Contents

Preface ................................................................. 11

Part 1 SQL Reference

OpenEdge SQL Statements ........................................ 19
OpenEdge SQL Functions ........................................... 129
OpenEdge SQL Reserved Words ................................ 201
OpenEdge SQL Error Messages ................................. 205
OpenEdge SQL System Limits .................................. 223
OpenEdge SQL System Catalog Tables ....................... 225
Data Type Compatibility ......................................... 251
OpenEdge SQL Language Elements ......................... 253
OpenEdge SQL Elements and Statements in Backus Naur Form ... 281
Compliance with Industry Standards ....................... 297
Syntax for ABL Attributes ...................................... 303
Contents

Part 2 JDBC Reference

Java Class Reference ................................................. 309
JDBC Conformance Notes ........................................ 337

Part 3 ODBC Reference

OpenEdge SQL and ODBC Data Types ............................. 355
SQLGetInfo ........................................................ 357
ODBC Scalar Functions ........................................... 377

Part 4 ESQL Reference

Embedded SQL ........................................................ 387
Third Party Acknowledgements ................................. 419
Index ............................................................... 455
Tables

Table 1: OpenEdge SQL reserved words .......................... 201
Table 2: OpenEdge SQL error codes and messages .................. 206
Table 3: OpenEdge SQL system limits .......................... 223
Table 4: System tables and descriptions .................. 226
Table 5: SYSTABLES core system table .................. 228
Table 6: SYSCOLUMNS core system table .................. 228
Table 7: SYSCOLUMNS system table .................. 229
Table 8: SYSCALLCTABLE system table .................. 230
Table 9: SYSNCHARSTAT system table .................. 230
Table 10: SYSCOLAUTH system table .................. 231
Table 11: SYSCOLSTAT system table .................. 231
Table 12: SYSCOLUMNS_FULL system table .................. 232
Table 13: SYSDATATYPES system table .................. 233
Table 14: SYSDATESTAT system table .................. 234
Table 15: SYSDBAUTH system table .................. 234
Table 16: SYSFLOATSTAT system table .................. 235
Table 17: SYSIDXSTAT system table .................. 235
Table 18: SYSINTSTAT system table .................. 236
Table 19: SYNUMSTAT system table .................. 236
Table 20: SYSPROCBIN system table .................. 237
Table 21: SYSPROCCOLUMNS system table .................. 237
Table 22: SYSPROCEDURES system table .................. 238
Table 23: SYSPROCTEXT system table .................. 238
Table 24: SYSPROCEXSTAT system table .................. 239
Table 25: SYSPROCEXSTAT system table .................. 239
Table 26: SYSSYNONYMS system table .................. 240
Table 27: SYSTABLES system table .................. 240
Table 28: SYSTABAUTH system table .................. 241
Table 29: SYSTABLES system table .................. 241
Table 30: SYSTBLSTAT system table .................. 243
Table 31: SYSTIMESTAT system table .................. 243
Table 32: SYSTINYINTSTAT system table .................. 244
Table 33: SYSTRIGCOLS system table .................. 244
Table 34: SYSTRIGGER system table .................. 245
Table 35: SYSTSTAT system table .................. 245
Table 36: SYSTSTZSTAT system table .................. 246
Table 37: SYNVARCHARSTAT system table .................. 246
Table 38: SYSVIEWS system table .................. 247
Table 39: SYS_CHKCOL_USAGE system table .................. 247
Table 40: SYS_CHK_CHK_CONSTS system table .................. 248
Table 41: SYS_CHK_CHK_USAGE system table .................. 248
Table 42: SYS_CHK_CHK_CONSTRS system table .................. 249
Table 43: SYS_CHK_CHK_CONSTRS system table .................. 249
Table 44: ABL and corresponding OpenEdge SQL data types .................. 251
Table 45: OpenEdge SQL Number Formats .................. 255
Table 46: Date formats and descriptions .................. 256
Table 47: Time formats and descriptions .................. 258
Table 48: Specification formats for binary values .................. 264
Table 49: Relational operators and resulting predicates .................. 273
Table 50: Compatibility of SQL-92 scalar functions .................. 297
Table 51: Compliance of SQL-92 DDL and DML statements .................. 301
Table 52: ABL table attributes used in OpenEdge SQL statements .................. 303
Table 53: ABL column attributes used in OpenEdge SQL statements .................. 305
Table 54: ABL index attributes used in OpenEdge SQL statements .................. 306
Table 55: Argument values for DhSQLException.getDiagnostics .................. 313
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>Allowable values for <code>fieldType</code> in <code>getParam</code></td>
<td>321</td>
</tr>
<tr>
<td>57</td>
<td>Allowable values for <code>fieldType</code> in <code>getValue</code></td>
<td>322</td>
</tr>
<tr>
<td>58</td>
<td>Allowable values for <code>fieldType</code> in <code>registerOutParam</code></td>
<td>325</td>
</tr>
<tr>
<td>59</td>
<td>Mapping between Java and JDBC data types</td>
<td>337</td>
</tr>
<tr>
<td>60</td>
<td>Mapping between JDBC and Java data types</td>
<td>338</td>
</tr>
<tr>
<td>61</td>
<td>Mapping between SQL-92 and Java data types</td>
<td>338</td>
</tr>
<tr>
<td>62</td>
<td>JDBC data type conversion</td>
<td>339</td>
</tr>
<tr>
<td>64</td>
<td>OpenEdge SQL and ODBC data types</td>
<td>355</td>
</tr>
<tr>
<td>65</td>
<td>Information the ODBC driver returns to SQLGetInfo</td>
<td>357</td>
</tr>
<tr>
<td>66</td>
<td>Scalar string functions</td>
<td>378</td>
</tr>
<tr>
<td>67</td>
<td>Scalar numeric functions</td>
<td>380</td>
</tr>
<tr>
<td>68</td>
<td>Date and time functions supported by ODBC</td>
<td>382</td>
</tr>
<tr>
<td>70</td>
<td>Compliance of SQL DDL and DML statements</td>
<td>417</td>
</tr>
</tbody>
</table>
Figures

Figure 1: ROUND digit positions ......................................................... 182
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Typographical conventions
- Examples of syntax diagrams (SQL)
Purpose

OpenEdge Data Management: SQL Reference provides specific information on the OpenEdge® SQL language. The reference contains information on SQL statements, functions, reserved words, error messages, data type compatibility, and the language’s compliance with industry standards. The book also provides reference information on the ODBC and JDBC drivers.

For the latest documentation updates, see the OpenEdge Product Documentation Overview page on PSDN: http://communities.progress.com/pcom/docs/DOC-16074.

Audience

The audience of this book is composed of two groups:

• **Database administrators will use the book to:**
  – Create and maintain databases
  – Create, modify, and revoke user privileges
  – Tune database performance
  – Perform installation and setup of servers and clients

• **Application developers will use the book to:**
  – Manage database connections and set up data sources
  – Create database queries
  – Tune database queries
  – Develop application business logic

Organization

Part 1, SQL Reference

*OpenEdge SQL Statements*

Describes the purpose and syntax of each OpenEdge SQL statement. A sample is provided for each statement.

*OpenEdge SQL Functions*

Describes the purpose and syntax of each OpenEdge SQL function. A sample is provided for each function.
OpenEdge SQL Reserved Words

Provides a list of words that have special syntactic meaning to OpenEdge SQL and cannot be used as identifiers for constants, variables, cursors, types, tables, records, subprograms or packages.

OpenEdge SQL Error Messages

Provides a list of error messages generated by the various components of OpenEdge SQL.

OpenEdge SQL System Limits

Provides a list of the maximum sizes for various attributes of the OpenEdge SQL database environment, and for elements of SQL queries addressed to this environment.

OpenEdge SQL System Catalog Tables

Provides a set of system tables for storing information about tables, columns, indexes, constraints, and privileges. This chapter describes those system catalog tables.

Data Type Compatibility

Addresses compatibility issues when using OpenEdge SQL and earlier versions of the database.

OpenEdge SQL Language Elements

Describes Standard SQL language elements that are common to OpenEdge SQL.

OpenEdge SQL Elements and Statements in Backus Naur Form

Presents OpenEdge SQL elements and statements in Backus Naur Form.

Compliance with Industry Standards

Addresses compatibility issues when using OpenEdge SQL and earlier versions of its database.

Syntax for ABL Attributes

Lists and describes SQL keywords to use with statements that allow you to define ABL attributes for tables and columns.

Part 2, JDBC Reference

Java Class Reference

Provides information on OpenEdge SQL Java classes and methods.

JDBC Conformance Notes

Provides information on mapping between JDBC and other data types and return values for database metadata.
Part 3, ODBC Reference

*OpenEdge SQL and ODBC Data Types*

Shows how the OpenEdge data types are mapped to the standard ODBC data types.

*SQLGetInfo*

Describes return values to SQL GetInfo from the ODBC driver.

*ODBC Scalar Functions*

Lists scalar functions that ODBC supports and are available to use in SQL statements.

Part 4, ESQL Reference

*Embedded SQL*

Provides reference information for an ESQL interface.

Appendix A, “Third Party Acknowledgements”

## Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td>SMALL, BOLD CAPITAL LETTERS</td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, <strong>GET</strong> and <strong>CTRL</strong>.</td>
</tr>
<tr>
<td>KEY1+KEY2</td>
<td>A plus sign between key names indicates a simultaneous key sequence: you press and hold down the first key while pressing the second key. For example, <strong>CTRL+X</strong>.</td>
</tr>
<tr>
<td>KEY1 KEY2</td>
<td>A space between key names indicates a sequential key sequence: you press and release the first key, then press another key. For example, <strong>ESCAPE H</strong>.</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td></td>
</tr>
<tr>
<td>Fixed width</td>
<td>A fixed-width font is used in syntax boxes, code examples, system output, and filenames.</td>
</tr>
<tr>
<td>Fixed-width italics</td>
<td>Fixed-width italics indicate variables in syntax boxes.</td>
</tr>
</tbody>
</table>
### Examples of syntax diagrams (SQL)

In this example, `GRANT`, `RESOURCE`, `DBA`, and `TO` are keywords. You must specify `RESOURCE`, `DBA`, or both, and at least one `user_name`. Optionally you can specify additional `user_name` items; each subsequent `user_name` must be preceded by a comma:

**Syntax**

```sql
GRANT { RESOURCE, DBA } TO user_name [, user_name ] ... ;
```

This excerpt from an ODBC application invokes a stored procedure using the ODBC syntax `{ call procedure_name ( param ) }`, where braces and parentheses are part of the language:

**Syntax**

```sql
proc1( param, *( call proc2 (param) )*, param);
```

In this example, you must specify a `table_name`, `view_name`, or `synonym`, but you can choose only one. In all SQL syntax, if you specify the optional `owner_name` qualifier, there must not be a space between the period separator and `table_name`, `view_name`, or `synonym`.

---

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fixed-width bold</strong></td>
<td>Fixed-width bold indicates variables with special emphasis.</td>
</tr>
<tr>
<td><strong>UPPERCASE fixed width</strong></td>
<td>Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.</td>
</tr>
<tr>
<td><img src="arrow_multi-step.png" alt="multi-step" /></td>
<td>This icon (three arrows) introduces a multi-step procedure.</td>
</tr>
<tr>
<td><img src="arrow_single-step.png" alt="single-step" /></td>
<td>This icon (one arrow) introduces a single-step procedure.</td>
</tr>
<tr>
<td><strong>lowercase</strong></td>
<td>Lowercase words are source language elements that are not SQL keywords.</td>
</tr>
<tr>
<td><code>[]</code></td>
<td>Large brackets indicate the items within them are optional.</td>
</tr>
<tr>
<td><code>{}</code></td>
<td>Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.</td>
</tr>
<tr>
<td>```</td>
<td>```</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Ellipses indicate repetition: you can choose one or more of the preceding items.</td>
</tr>
</tbody>
</table>
Syntax

```sql
CREATE [ PUBLIC ] SYNONYM synonym
    FOR [ owner_name.] { table_name | view_name | synonym } ;
```

In this example, you must specify `table_name` or `view_name`.

Syntax

```sql
DELETE FROM [ owner_name.] { table_name | view_name }
    [ WHERE search_condition ] ;
```

In this example, you must include one expression (`expr`) or column position (`posn`), and optionally you can specify the sort order as ascending (`ASC`) or descending (`DESC`). You can specify additional expressions or column positions for sorting within a sorted result set. The SQL engine orders the rows on the basis of the first `expr` or `posn`. If the values are the same, the second `expr` or `posn` is used in the ordering:

Syntax

```sql
ORDER BY { expr | posn } [ ASC | DESC ]
    [ , [ { expr | posn } [ ASC | DESC ] ] ] ...
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, `CREATE VIEW` is followed by several optional items:

Syntax

```sql
CREATE VIEW [ owner_name.] view_name
    [ ( column_name [, column_name ] ... ) ]
    AS [ ( ] query_expression [ ) ] [ WITH CHECK OPTION ] ;
```
Part 1

SQL Reference

OpenEdge SQL Statements
OpenEdge SQL Functions
OpenEdge SQL Reserved Words
OpenEdge SQL Error Messages
OpenEdge SQL System Limits
OpenEdge SQL System Catalog Tables
Data Type Compatibility
OpenEdge SQL Language Elements
OpenEdge SQL Elements and Statements in Backus Naur Form
Compliance with Industry Standards
Syntax for ABL Attributes
OpenEdge SQL Statements

This section provides detailed information on OpenEdge® SQL statements. A description for each statement provides the following information:

- A definition of the statement
- The syntax of the statement’s proper usage
- A code sample that shows how the statement works
- Any associated notes
- Authorization required in order to run the statement
- Related statements

**ALTER DATABASE SET PRO_ENABLE_LARGE_KEYS 'Y'**

This statement is used to enable large index database keys of up to 2,000 bytes.

**Syntax**

```
ALTER DATABASE SET PRO_ENABLE_LARGE_KEYS 'Y'
```

**Notes**

- You must have SQL DBA privileges to enable large keys.
- As of OpenEdge Release 10.1B, large index keys are supported for databases with 4,000 and 8,000 byte block sizes and are enabled by default.
In Releases 10.1B and later, large index key sizes are enabled by default. However, in Releases 10.1A and earlier, this statement must be used to enable large keys.

If large keys are already enabled, an error message indicating such will be returned when you use the statement.

This statement is used to enable large index keys in a database.

**Syntax**

```
ALTER DATABASE SET PRO_ENABLE_64BIT_SEQUENCES 'Y'
```

**Notes**

- You must have SQL DBA privileges to enable 64-bit sequences.
- As of OpenEdge Release 10.1B, OpenEdge databases support sequences with 64-bit maximums for positive and negative numbers.
- In releases 10.1B and later, this feature is enabled by default. However, in Releases 10.1A and earlier, this specific `ALTER DATABASE` statement must be used to enable 64-bit sequences.
- If 64-bit sequences are already enabled, an error message indicating such will be returned when you use the statement.

The `ALTER GROUP` statement allows you to add existing tenants to an existing multi-tenant group that is defined for a multi-tenant table. It also allows you to drop a tenant from an existing group that is defined for a multi-tenant table.

The tenant that you want to add to the group should not be a part of the group that is being altered. The tenant that is part of a group for a particular table cannot be part of another group for the same table.

To drop a tenant from the group that you want to alter, you must ensure that the tenant is a part of that group. When a tenant is dropped from a group, all the tenant partitions for that table are created with `NO SPACE`.

When the tenant of a specified table joins a group, the tenant’s existing partition for that table is removed. The tenant partition for the table (on which the current group being altered is defined) must have a `NO SPACE` attribute set.
Syntax

```
ALTER GROUP group_name {ADD | DROP} TENANT list_of_tenant_names
```

`group_name`

Specifies the name of the group that you want to alter.

`ADD TENANT`

Adds an existing tenant to an existing multi-tenant group that is defined for a multi-tenant table.

`DROP TENANT`

Drops an existing tenant from an existing multi-tenant group that is defined for a multi-tenant table.

`list_of_tenant_names`

Specifies the list of tenants that you want to add to or drop from an existing multi-tenant group.

Examples

The following example alters a group by adding tenants to it:

```
ALTER GROUP hyb-parkinglot-group ADD TENANT easy-cabs,green-cabs,city-cabs
```

The following example alters a group by dropping tenants from it:

```
ALTER GROUP hyb-parkinglot-group DROP TENANT easy-cabs,green-cabs,city-cabs
```

Authorization

To alter a group, you must have DBA privileges.

Related Statement

CREATE GROUP, DROP GROUP
The `ALTER SEQUENCE` statement can be used to change the current value of an existing sequence. The sequence can be in the current schema or a schema can be specified.

When altering a sequence of a multi-tenant table as a regular tenant, the `FOR TENANT` phrase is optional to modify its `CURRVAL`, and a regular tenant does not have the privileges to modify the global attributes of a sequence.

When altering the sequence of a multi-tenant table as a DBA or a super-tenant, the `FOR TENANT` phrase is mandatory for a DBA or a super-tenant to identify the tenant in the multi-tenant database that must be altered. A DBA or a super-tenant can alter all the sequence attributes of a multi-tenant table.

**Syntax**

```
ALTER SEQUENCE [schema_name]sequence_name
[SET
  { CURRVAL value [ FOR TENANT tenant_name ] | START WITH value |
    INCREMENT BY value | MAXVALUE value | NOMAXVALUE | |
    MINVALUE value | NOMINVALUE | CYCLE | NOCYCLE ]}]
```

- `schema_name`
  Specifies the name of the schema name that contains the sequence. If this is not specified, OpenEdge SQL alters the sequence.

- `sequence_name`
  Specifies the sequence to be altered.

- `CURRVAL`
  Returns the current value of the sequence.

- `FOR TENANT tenant_name`
  Specifies the name of the tenant corresponding to the `CURRVAL`.

- `INCREMENT BY`
  Specifies the interval between sequence numbers. The value can be a positive or negative integer (INTEGER data type for 32-bit sequences, BIGINT datatype for 64-bit sequences), but it cannot be 0. The value range for a 32-bit sequence is -2,147,483,648 to 2,147,483,647. The value range for a 64-bit sequence is -9223372036854775808 to 9223372036854775807. The value is positive for an ascending sequence and negative for a descending sequence. The default value is 1.
START WITH

Specifies the first number in a sequence generated. The value must be greater than or equal to the `MINVALUE` for an ascending sequence. The value must be less than or equal to the `MAXVALUE` for a descending sequence. The default value for an ascending sequence is `MINVALUE` and `MAXVALUE` for a descending sequence.

MAXVALUE

Specifies the maximum value that the sequence can generate. The default value is -1 for both 32-bit and 64-bit descending sequences. The default value is 2,147,483,647 for a 32-bit ascending sequence and 9223372036854775807 for a 64-bit ascending sequence.

NOMAXVALUE

Specifies -1 as the `MAXVALUE` for a 32-bit descending sequence and 2,147,483,647 as the `MAXVALUE` for a 32-bit ascending sequence. Specifies -1 as the `MAXVALUE` for a 64-bit descending sequence and 9223372036854775807 as the `MAXVALUE` for a 64-bit ascending sequence.

MINVALUE

Specifies the minimum value that the sequence can generate. The default value is 0 for an ascending sequence. The default value is -2,147,483,6480 for a 32-bit descending sequence and -9223372036854775808 for a 64-bit sequence.

NOMINVALUE

Specifies 0 as the `MINVALUE` for an ascending sequence. The `MINVALUE` is -2,147,483,648 for a 32-bit descending sequence and -9223372036854775808 for a 64-bit sequence.

CYCLE

Indicates that the sequence will continue to generate values after attaining the `MAXVALUE` (for an ascending sequence) or `MINVALUE` (for a descending sequence).

NOCYCLE

Indicates that the sequence cannot generate more values after attaining the `MAXVALUE` (for an ascending sequence) or `MINVALUE` (for a descending sequence). The SQL sequence generator uses `NOCYCLE` as the default if `CYCLE` is not specified.
Examples

The following example modifies a sequence by specifying a maximum value:

```
ALTER SEQUENCE customer RENAME TO Customers;
```

The following example when executed by a DBA or a super-tenant, modifies a sequence of a multi-tenant table by specifying a maximum value:

```
ALTER SEQUENCE pub.cust_num_seq SET MAXVALUE 99999;
```

The following example when executed by a DBA or a super-tenant, modifies a sequence by specifying a CURRVAL for the tenant AsiaPacificCust of a multi-tenant table:

```
ALTER SEQUENCE pub.cust_num_seq SET CURRVAL 521 FOR TENANT mtAsiaPacificCust;
```

Notes

- It is possible to set only one attribute of a sequence at a time.
- Attributes START WITH, INCREMENT BY, MAXVALUE, MINVALUE and CURRVAL can take either an INTEGER or BIGINT argument, depending on whether the sequence is a 32-bit or a 64-bit sequence.

```
ALTER SEQUENCE pub.customer
SET MAXVALUE 9000000000;
```

Authorization

To alter a sequence, you must have DBA privileges or ownership of the table.

Related Statement

DROP SEQUENCE

ALTER TABLE

The ALTER TABLE statement can be used to:

- Change the name of a table
- Change the name of a column within a table
- Add a column to a table
- Set (ABL) Advanced Business Language table, column and index attributes
- Convert a table to a multi-tenant table
- Add or drop LOB columns to or from a multi-tenant table
The syntax to alter a table is given as follows:

Syntax

```sql
ALTER TABLE [ owner_name ]table_name
{ADD column-definition
 |SET progress_table_attribute value
 |SET { ENCRYPT WITH cipher
 | DECRYPT
 | ENCRYPT REKEY }
 |BUFFER_POOL { PRIMARY | ALTERNATE }}
|ALTER [ COLUMN ]column_name { SET DEFAULT value
 | DROP DEFAULT
 | SET [NOT] NULL
 | SET progress_column_attribute value}
 |SET ENCRYPT WITH cipher
 |SET DECRYPT
 |SET ENCRYPT REKEY
 |SET BUFFER_POOL { PRIMARY | ALTERNATE }}
|DROP COLUMN column_name { CASCADE | RESTRICT }
|ADD [CONSTRAINT constraint_name ] { primary_key_definition
 |foreign_key_definition
 |uniqueness Constraint
 |check_constraint} [ AREA area_name ]
|DROP CONSTRAINT constraint_name { CASCADE | RESTRICT]
|ALTER INDEX index_name { SET progress_index_attribute value
 |SET ENCRYPT WITH cipher
 |SET DECRYPT
 |SET ENCRYPT REKEY
 |SET BUFFER_POOL { PRIMARY | ALTERNATE }}
|RENAME {table_name TO new_table_name
 |COLUMN column_name TO new_column_name
 |INDEX index_name TO new_index_name }
| [ SET MULTI_TENANT
 | ] FOR TENANT { [ owner_name ]tenant_name_1 | DEFAULT }
 | [ USING TABLE AREA table_area_name ]
 | [ USING INDEX AREA index_area_name ]
 | [ USING LOB AREA lob_area_name ]
 | ] ... 
| [ FOR TENANT { [ owner_name ]tenant_name_2 | DEFAULT } USING NO SPACE ] ]
;```

The syntax parameters are explained in the following sections.
The following syntax is used to define an **LOB** column in an `ALTER TABLE ADD COLUMN` statement:

**Syntax**

```
{ LVARCHAR | CLOB | LVARBINARY | BLOB } [ ( length ) ]
[ AREA area_name ]
[ ENCRYPT WITH cipher ]
[ BUFFER_POOL { PRIMARY | ALTERNATE } ]
```

The syntax used to convert a table to a multi-tenant table using the `ALTER TABLE` statement is given below:

**Syntax**

```
[ SET MULTI_TENANT
[ FOR TENANT { [ owner_name]:tenant_name_1 | DEFAULT } ]
[ USING TABLE AREA table_area_name ]
[ USING INDEX AREA index_area_name ]
[ USING LOB AREA lob_area_name ]
] ... 
[ FOR TENANT { [ owner_name]:tenant_name_2 | DEFAULT } USING NO SPACE ] ]
```

The following special syntax is for Database Administrators who handle problems while migrating database tables from an older release database to a newer release database. In some cases, there may be a CRC mismatch error (when dumping and loading data using `BINARY DUMP/LOAD`).

**Syntax**

```
ALTER TABLE table_name SET PRO_FIELD_INFO PRO_SCHEMA_REPAIR
```

**Note:** The `PRO_FIELD_INFO` attribute indicates that the incorrect values in the `_Field` table, corresponding to the given table name, need to be updated. The `PRO_SCHEMA_REPAIR` attribute provides additional authorization to DBAs that is required to perform the necessary updates to CRC specific rows in system tables.

Using the above statement for a particular table automatically repairs all the known schema errors in the table that are causing the CRC mismatch. If the error persists even after running this statement, report the problem to Progress Customer Support with the `_Field` system table information in the old and new databases.

Execution of the above statement sets `_DB-lang` to its default value (0) and updates the following data type names as shown below:

- `TIMESTAMP` to `DATETIME`
- `LVARBINARY` to `BLOB`
- `LVARCHAR` to `CLOB`
Executing the above statement also sets the case sensitivity value of the non-character type columns to 0.

**Notes**

- See the Notes section on page 301 for a list of ABL table, column and index attributes.

- The `ALTER TABLE ALTER INDEX` statement can use two index attributes, `PRO_DESCRIPTION` and `PRO_ACTIVE`. The `PRO_DESCRIPTION` attribute enables the index definition to accept free-form text in the same manner as ABL. The `PROC_ACTIVE` attribute takes only an `n` as an argument, thereby changing the index’s status from active to inactive. Changing an index’s status to inactive is an action that must be performed offline. For a description of the `PRO_DESCRIPTION` and `PRO_ACTIVE` attributes, see the Notes section on page 301.

- Table columns defined by OpenEdge SQL have default format values identical to those created by the Data Dictionary.

- For details on using the `ALTER TABLE ADD COLUMN` statement to designate objects for buffer pool assignments, including an alternate buffer pool, see *OpenEdge Data Management: Database Administration*.

- For details on using the `ALTER TABLE` statement to enable transparent data encryption, see *OpenEdge Getting Started: Core Business Services - Security and Auditing*.

**Examples**

In the following example, the `ALTER TABLE` statement is used to change the name of a table from `customer` to `Customers`:

```
ALTER TABLE customer RENAME TO Customers;
```

In this example, the `ALTER TABLE` statement is used to change the name of a column within a table and the column named `Address` changes to `Street`:

```
ALTER TABLE customer RENAME Address TO Street;
```

In this example, table `customer` adds the column `Region`:

```
ALTER TABLE customer ADD COLUMN Region;
```

In this example, table `customer` changes an existing 32-bit `INTEGER` column into a 64-bit `BIGINT` column:

```
ALTER TABLE OrderLine ALTER COLUMN Qty SET PRO_DATA_TYPE BIGINT;
```

Once the above statement is executed, the column will appear as a `BIGINT` column both internally and to applications.

A statement such as this executed against a column that is not 32-bit will result in an error.
In this example, `ALTER TABLE` adds an ABL description to a table and changes the ABL default data access index of the table:

```
ALTER TABLE pub.customer SET PRO_DESCRIPTION 'Sports 2000 Customers';
ALTER TABLE pub.customer SET PRO_DEFAULT_INDEX CustNumIdx;
```

In this example, `ALTER TABLE RENAME INDEX` is used to change an index named `CustNum` to `CustomerNumberIndex`:

```
ALTER TABLE Customers RENAME INDEX CustNum to CustomerNumberIndex;
```

The `ALTER TABLE` statement enables you to add columns or to set encryption while your database is online servicing other requests. Other changes performed by `ALTER TABLE` must occur offline.

The following example changes the `pub.mtcustomer` table to a multi-tenant table with default space allocations:

```
ALTER TABLE pub.mtcustomer SET MULTI_TENANT;
```

The following example alters the `pub.mtcustomer` table to be a multi-tenant table with overriding space allocations:

```
ALTER TABLE pub.mtcustomer SET MULTI_TENANT
FOR TENANT "Consolidated_Freightways"
    USING TABLE AREA "MMM Data area"
    USING INDEX AREA "MM CF Fin idx area"
FOR TENANT Mega_Media_Networks
    USING LOB AREA "MMM pix area" ;
FOR TENANT DEFAULT USING NO SPACE;
```

In general, the above example describes how to convert a regular table to a multi-tenant table. When a regular table is converted to a multi-tenant table, the table data is moved to the default partition. To move the data from the default partition to the tenant partitions, see the `INSERT` statement.

The following example repairs the CRC mismatch error:

```
ALTER TABLE Customer SET PRO_FIELD_INFO PRO_SCHEMA_REPAIR;
```

In the above example, the `Customer` table belongs to an old database. When you execute the above statement, the inconsistent field information is corrected and the table is updated with the new CRC.
Authorization

To alter a table, you must have DBA privileges, ownership of the table, or all the specified privileges on the table.

Related statements

ADD TABLE, DROP TABLE

ALTER TENANT

The ALTER TENANT statement can be used to:

- Modify the definition of a tenant
- Change the NO SPACE parameter specified for one or more existing multi-tenant tables
- Add an existing tenant to an existing multi-tenant group that is defined for a multi-tenant table

The table name and group name specified in the FOR TABLE JOIN clause should not be repeated. You can add the tenant to only one group for a given table. However, you can add it to multiple groups on different tables.

The current tenant partition for the table specified in the FOR TABLE clause should be allocated NO SPACE. When the tenant for the specified table joins a group, its existing table, index, and LOB partitions for that table is removed. No new partition is created for the tenant-table combination.

Syntax

```
ALTER TENANT tenant_name

   [FOR TABLE owner_name]table_name_1 [ { [USING TABLE AREA table_area_name ]
     [USING INDEX AREA index_area_name ]
     [USING LOB AREA lob_are_name ]
     |
     JOIN GROUP group_name } ] ...

;  
```  

`tenant_name`

Specifies the name of the tenant you want to modify.

`FOR TABLE`  

Indicates the default area, which can be overridden by a specific area for a table, or for its index, or for its LOB areas. You can specify an area of storage for data of tables and indexes.

`TABLE AREA area_name`

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE TABLE statement execution. The TABLE AREA specification is optional.
INDEX  AREA  index_area_name

Modifies the name of the default index storage area for a tenant.
If the storage area for an index is omitted in the syntax, then the default index area of the tenant is allocated to the specified index.

LOB AREA  lob_area_name

Modifies the name of the default LOB storage area for a tenant.
If the storage area for a LOB is omitted in the syntax, then the default LOB area of the tenant will be allocated to the specified LOB.

table_name_1

Specifies the name of the table. The referenced table name must be a valid table in the multi-tenant database.

owner_name

Specifies the name of the schema that qualifies the table.

JOIN GROUP

Adds the tenant to the specified group.

group_name

Specifies the name of the group that you want the tenant to join.

Examples

The following example modifies the table partition of the tenant to the allocated space and adds the tenant to a group:

```
ALTER TENANT Mega_Media_Networks
    FOR TABLE pub.finance_accts USING TABLE DEFAULT AREA
    FOR TABLE pub.new_customers USING TABLE AREA "MMN Data area"
        USING INDEX AREA "MMN_Index_Area"
    FOR TABLE pub.farm_location
    FOR TABLE pub.archive_10yr JOIN GROUP Joint_Archives
;
```

Authorization

To alter a tenant, you must have DBA privileges.

Related Statement

DROP TENANT
ALTER USER

Changes the password for the specified user.

Syntax

```sql
ALTER USER { 'username' | 'username@domain_name' }, 'old_password',
'new_password';
```

username

Specifies the username. The username must be enclosed in single quotes.

domain_name

Specifies the domain name that the user is associated with. The
`username@domain_name` must be enclosed in single quotes.

old_password

Specifies the current password for the user. The password must be enclosed in
quotes.

new_password

Specifies a new password for the user. The password must be enclosed in quotes.

Example

In this example, the `ALTER USER` statement Jasper changes the Jasper account
password from normandy to brittany:

```
ALTER USER 'Jasper', 'normandy', 'brittany' ;
```

The example displayed below shows how to change the password for a user
associated with a domain. The following `ALTER USER` statement changes the password
from normandy to brittany for the user Jasper associated with mtdomain domain:

```
ALTER USER 'Jasper@mtdomain', 'normandy', 'brittany' ;
```

Notes

- Used in conjunction with `CREATE USER` and `DROP USER`, the `ALTER USER` statement
  provides a way to change a user password.

- The `old_password` specification must match the current password for `username`.

Authorization

User specified in `username`.

Related statements

`CREATE USER`, `DROP USER`
AUDIT INSERT

Writes application audit events to an audit-enabled database.

For more information about auditing, see OpenEdge Getting Started: Core Business Services - Security and Auditing. For more information about enabling a database for auditing, see OpenEdge Data Management: Database Administration.

Syntax

```
AUDIT INSERT ( event_id,
                [ event_context | NULL ]
                [ event_detail | NULL ]
);
```

*event_id*

Positive integer value corresponding to an audit event record. The event_id must be a value greater than 32000.

*event_context*

Free-form character value that qualifies the event_id. May include non-ASCII characters.

*event_detail*

Free-form character value that supplies detailed information about the audit event. May include non-ASCII characters.

Notes

- Before inserting the specified application audit event into the database, the OpenEdge SQL engine determines the following:
  - The connected user has been granted the audit insert privilege.
  - The event id is valid and active.

If both of these conditions are true, the engine writes the application audit event to the database. If one or both are not true, the engine does not write the event.

- **AUDIT INSERT** always returns a success status. This prevents users from determining whether or not they have privileges to log application audit events.
AUDIT SET

Allows grouping of audit data by the client on a per-connection basis.

For more information about auditing, see OpenEdge Getting Started: Core Business Services - Security and Auditing. For more information about enabling a database for auditing, see OpenEdge Data Management: Database Administration.

Syntax

```
AUDIT SET { EVENT_GROUP | APPLICATION_CONTEXT } { string | NULL };
```

**EVENT_GROUP**

Indicates that subsequent audit records written by the database engine during the current connection will be marked as part of an event group.

**APPLICATION_CONTEXT**

Indicates that subsequent audit records written by the database engine during the current connection will be saved with application context information.

**string**

Free-form character value which is a unique string identifier for the group/application context event.

**NULL**

Clears an event group or application context string.

**string**

Free-form character value which provides additional application detail that describes the group/application context.

**NULL**

Clears an event group or application context string.

**Examples**

In this example, an application context is set:

```
AUDIT SET APPLICATION_CONTEXT 'app.name.checking' '06/02/2005 Deposits';
```

In this example, the application context is cleared:

```
AUDIT SET APPLICATION_CONTEXT NULL NULL;
```
CALL

Note
AUDIT SET always returns a success status. This prevents users from determining whether or not they have audit privileges.

Authorization
Must have AUDIT_ADMIN, AUDIT_ARCHIVE, or AUDIT_INSERT privileges.

CALL
Invokes a stored procedure.

Syntax
CALL proc_name ( [ parameter ] [ , ... ] );

proc_name
The name of the procedure to invoke.

parameter
Literal or variable value to pass to the procedure.

Example
This example shows an excerpt from an ODBC application that calls a stored procedure (order_parts) using the ODBC syntax {call procedure_name (param)}:

```sql
SQLINTEGER Part_num;
SQLINTEGER Part_numInd = 0;
// Bind the parameter.
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT,
        SQL_C_SLONG, SQL_INTEGER, 0, 0, &Part_num, 0, Part_numInd);
// Place the department number in Part_num.
    Part_num = 318;
// Execute the statement.
    SQLExecDirect(hstmt, "{call order_parts(?)}", SQL_NTS);
```

Authorization
Must have DBA or EXECUTE privileges.

Related statements
CREATE PROCEDURE, DROP PROCEDURE
Column constraints

Specifies a constraint for a column that restricts the values that the column can store. \texttt{INSERT}, \texttt{UPDATE}, or \texttt{DELETE} statements that violate the constraint fail. The database returns a constraint violation error with an SQLCODE of -20116.

Column constraints are similar to table constraints, but their definitions are associated with a single column rather than the entire table.

Syntax

\begin{verbatim}
CONSTRAINT constraint_name
  NOT NULL [ PRIMARY KEY | UNIQUE ]
  | REFERENCES [ owner_name.|table_name [ ( column_name ) ]
  | CHECK ( search_condition )
\end{verbatim}

\texttt{CONSTRAINT constraint_name}

Allows you to assign a name for the column constraint. This option facilitates making changes to the column definition. If you do not specify a \texttt{constraint_name}, the database assigns a name. These names can be long and unwieldy, and you must query system tables to retrieve the name.

\texttt{NOT NULL}

Restricts values in the column to values that are not null.

\texttt{NOT NULL PRIMARY KEY}

Defines the column as the primary key for the table. There can be at most one primary key for a table. A column with the \texttt{NOT NULL PRIMARY KEY} constraint should not contain null or duplicate values.

Other tables can name primary keys as foreign keys in their \texttt{REFERENCES} clauses. If they do, SQL restricts operations on the table containing the primary key in the following ways:

- \texttt{DROP TABLE} statements that delete the table fail
- \texttt{DELETE} and \texttt{UPDATE} statements that modify values in the column that match a foreign key's value also fail

\texttt{NOT NULL UNIQUE}

Defines the column as a unique key that cannot contain null or duplicate values. Columns with \texttt{NOT NULL UNIQUE} constraints defined for them are also called candidate keys.

Other tables can name unique keys in their \texttt{REFERENCES} clauses. If they do, SQL restricts operations on the table containing the unique key.
REFERENCES table_name [ ( column_name ) ]

Defines the column as a foreign key and specifies a matching primary or unique key in another table. The REFERENCES clause names the matching primary or unique key.

A foreign key and its matching primary or unique key specify a referential constraint. A value stored in the foreign key must either be null or be equal to some value in the matching unique or primary key.

You can omit the column_name argument if the table specified in the REFERENCES clause has a primary key and you want the primary key to be the matching key for the constraint.

CHECK ( search_condition )

Specifies a column-level check constraint. SQL restricts the form of the search condition. The search condition must not:

- Refer to any column other than the one with which it is defined
- Contain aggregate functions, subqueries, or parameter references

Examples

The following example shows the creation of a primary key column on the supplier table:

```sql
CREATE TABLE supplier (  
    supp_no     INTEGER CONSTRAINT supp_key_con NOT NULL PRIMARY KEY,  
    name        CHAR (30),  
    status      SMALLINT,  
    city        CHAR (20)  
) ;
```

The following example creates a NOT NULL UNIQUE constraint to define the column ss_no as a unique key for the employee table:

```sql
CREATE TABLE employee (  
    empno        INTEGER NOT NULL PRIMARY KEY,  
    ss_no        INTEGER NOT NULL UNIQUE,  
    ename        CHAR (19),  
    sal          NUMERIC (10, 2),  
    deptno       INTEGER NOT NULL  
) ;
```
The following example defines `order_item.orditem_order_no` as a foreign key that references the primary key `orders.order_no`:

```
CREATE TABLE orders (
    order_no INTEGER NOT NULL PRIMARY KEY,
    order_date DATE
) ;
CREATE TABLE order_item (
    orditem_order_no INTEGER REFERENCES orders ( order_no ),
    orditem_quantity INTEGER
) ;
```

The second `CREATE TABLE` statement in the previous example could have omitted the column name `order_no` in the `REFERENCES` clause, since it refers to the primary key of table `orders`.

The following example creates a check constraint:

```
CREATE TABLE supplier (
    supp_no     INTEGER NOT NULL,
    name        CHAR (30),
    status      SMALLINT,
    city        CHAR (20) CHECK (supplier.city <> 'BadApple')
) ;
```

If a column is defined with a `UNIQUE` column constraints, no error results if more than one row has a `NULL` value for the column.

**COMMIT**

Commits a transaction explicitly after executing one or more SQL statements. Committing a transaction makes permanent any changes made by the SQL statements.

**Syntax**

```
COMMIT [ WORK ] ;
```

**Notes**

- The SQL statements executed prior to executing the `COMMIT` statement are executed as one atomic transaction that is recoverable and durable. The transaction is serializable if you specify this isolation level.

- On a system failure and/or the execution of the `ROLLBACK`, the transaction is rolled back to its initial state. Any changes made by the transaction are undone, restoring the database to its initial state. In the event of a system failure, the transaction will be rolled back during crash recovery when the database is restarted.

- A `COMMIT` operation makes any database modifications made by that transaction permanent.
CONNECT AS CATALOG

- Once a COMMIT operation is executed, the database modifications cannot be rolled back.
- Once a COMMIT operation is executed, the transaction modifications are guaranteed durable regardless of any transient system failures.
- The atomicity applies only to the database modification and not to any direct I/O performed to devices such as the terminal, printer, and OS files by the application code.
- A COMMIT operation releases all locks implicitly or explicitly acquired by the transaction.

Related Statement

ROLLBACK

CONNECT AS CATALOG

Establishes a connection to an auxiliary read-only database.

Syntax

```sql
CONNECT 'database_path' AS CATALOG catalog_name;
```

- **database_path**
  - Full path to database directory and database name. This must be contained within quotes.
- **catalog_name**
  - Catalog name to be used as an alias for the database in schema, table and column references. This must be in the form of an SQL identifier.

Example

In this example, the database named *customer* in directory `/usr/databases` is connected as a catalog named *mydb1*:

```sql
CONNECT 'usr/databases/customer' AS CATALOG mydb1;
```

Notes

- Used to provide read-only access to multiple databases on a single client connection to an SQL server.
- Once connected, the catalog name for an auxiliary database may be used in SQL statements to qualify schema, table, and column access.
- The catalog name is visible, and usable, only in the client-server session in which it is defined.
- The catalog name of the primary database is the name of the primary database. The database name is the name by which the database is started (for example, by the proserve command) omitting all file system path information.
CREATE DOMAIN

Several client-server sessions may each connect to the same auxiliary database within an entire OpenEdge SQL Server process. Each such client-server session may use the same or different name when connecting to the same auxiliary database.

If you are connected to a primary database that is unencrypted and it was started using the -t startup parameter, you will be unable to simultaneously connect to an encrypted auxiliary database. An attempt to do so will result in an error.

Authorization

Any user allowed to execute this statement. However, authorization for access to the auxiliary database is governed by the same rules that govern access to the primary database. That is, the username and password of the current user must be authorized using access control information in the auxiliary database.

SQL Compliance

Progress Software Corporation specific extension.

Related statements

DISCONNECT CATALOG, SET CATALOG

CREATE DOMAIN

Defines a security domain to authenticate users for an existing tenant or a super-tenant. The domain name must be unique within the set of domain names that are defined for a database. A tenant can be associated with multiple domains.

Syntax

```
CREATE DOMAIN domain_name
FOR { TENANT tenant_name | SUPER_TENANT super-tenant_name }
[ PRO_DESCRIPTION value ]
;
```

domain_name

Specifies the name of the security domain. You must specify the domain name even if it is the same as the tenant name.

TENANT tenant_name

Specifies the name of an existing tenant. For a DEFAULT tenant, you must specify default as the tenant name.

SUPER_TENANT super-tenant_name

Specifies the name of the tenant defined as a super-tenant in the database.
CREATE GROUP

PRO_DESCRIPTION value

Allows you to enter an ABL description. The value attribute is an arbitrary character string.

Example

The following example illustrates how to create a tenant and a domain for the tenant.

To define a tenant called Mega_Media_Networks:

```sql
CREATE TENANT Mega_Media_Networks TABLE AREA "MMM Data area"
INDEX AREA "MM index area"
LOB AREA "MMM pix area" ;
```

To define a domain for the tenant Mega_Media_Networks:

```sql
CREATE DOMAIN dom_MMNet FOR TENANT Mega_Media_Networks;
```

To specify users who can securely connect to the domain dom_MMNet, you must create users and associate them with the domain dom_MMNet.

Authorization

To create a domain, you must have Database Administrator (DBA) privileges.

Related Statement

DROP DOMAIN

CREATE GROUP

Creates a group for the specified table in a multi-tenant database. A multi-tenant group allows multiple tenants to map to the same table partition for a multi-tenant table and therefore share the same data for that table. All data within the table partition is shared by the multiple tenants that are part of the group.

You can create multiple groups for the same table. The areas specified in the command prompt must be pre-existing as valid Type II areas in the database.

Syntax

```sql
CREATE GROUP group_name FOR TABLE table_name TABLE AREA area_name_1
INDEX AREA area_name_2
LOB AREA area_name_3
[ PRO_DESCRIPTION value ]
```
CREATE GROUP

group_name

Specifies the name of the group for the multi-tenant table. The group name must be unique across the database.

table_name

Specifies the table name. The name of a referenced table must be a valid multi-tenant table. The table name can be qualified by an owner (schema) name.

TABLE AREA  area_name

Is used for database space allocation for the table, during the CREATE GROUP statement execution. The TABLE AREA must be specified.

INDEX AREA  index_area_name

Is used for database space allocation for the index, during the CREATE GROUP statement execution. The INDEX AREA specification is optional. The name of the storage area must be specified within double quotes.

If the storage area for an index is omitted in the syntax, the TABLE AREA acts as its default area.

LOB AREA  lob_area_name

Is used for database space allocation for LOB, during the CREATE GROUP statement execution. The LOB AREA specification is optional. The name of the storage area must be specified within double quotes.

If the storage area for a LOB is omitted in the syntax, the TABLE AREA acts as its default area.

PRO_DESCRIPTION  value

Allows you to enter the ABL description. The value attribute is an arbitrary character string.

FOR TABLE

Specifies the name of the table for which the group is defined.

Examples

The following example shows how to create a group for a table and allocate partition space for the table, index, and LOB:

```
CREATE GROUP carz_second FOR TABLE pub.customer
  [ TABLE AREA "CUSTTABAREA"
  | INDEX AREA "CUSTIDAREA"
  | LOB AREA "CUSTTLOBAREA"
  | PRO_DESCRIPTION "Second Group for CARZ"
```

Authorization

To create a group, you must have DBA privileges.

Related Statement

SHOW GROUP

CREATE INDEX

Creates an index on the specified table using the specified columns of the table. An index improves the performance of SQL operations whose predicates are based on the indexed column. However, an index slows performance of INSERT, DELETE, and UPDATE operations.

A DBA must use the FOR TENANT phrase to over-ride the default index storage areas and allocate new database storage areas for the index. A multi-tenant index is generated for a multi-tenant table.

Syntax

```
CREATE [ UNIQUE ] INDEX index_name
   ON table_name
   ( { column_name [ ASC | DESC ] } [ , ... ] )
   [ AREA area_name ]
   [ ENCRYPT WITH cipher ]
   [ BUFFER_POOL { PRIMARY | ALTERNATE } ]
   [ PRO_DESCRIPTION value ]
   [ PRO_ACTIVE {'N' | 'n'} ]
   [ FOR TENANT { tenant_name_1 | DEFAULT } USING INDEX AREA area_name | USING NO SPACE ] ...
```

**UNIQUE**

Does not allow the table to contain rows with duplicate column values for the set of columns specified for that index.

```
index_name
```

Must be unique for the given table.

```
table_name
```

The name of the table on which the index is built.

```
column_name [ , ... ]
```

The columns on which search and retrieval is ordered. These columns are called the index key. When more than one column is specified in the CREATE INDEX statement, a concatenated index is created.
ASC | DESC

Allows the index to be ordered as either ascending (ASC) or descending (DESC) on each column of the concatenated index. The default is ASC.

AREA area_name

The name of the storage area where the index and its entries are stored.

ENCRIPT WITH cipher

Allows the index to be encrypted by designating an appropriate cipher.

BUFFER_POOL { PRIMARY | ALTERNATE }

Allows the index to be assigned to a primary or alternate buffer pool.

PRO_DESCRIPTION value

Allows you to enter an ABL description. value is an arbitrary character string.

PRO_ACTIVE {'N'|'n'}

Indicates the creation of an inactive index. Inactive indexes can be created for an online database.

FOR TENANT tenant_name

Specifies the name of a tenant. The FOR TENANT phrase can be repeated for as many existing tenants as required.

The FOR phrase cannot be used, if it is not a multi-tenant table.

USING INDEX AREA area_name

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE INDEX statement execution. The INDEX AREA specification is optional.

If this phrase is not specified, the index will be allocated in the default index area for the particular tenant.

Note: The area name is ignored for a multi-tenant index.

USING NO SPACE

Indicates that no space is allocated for index data storage for the table. This phrase is valid only if the tenant partition for the table is allocated NO SPACE.

Note: For a particular tenant, the table and the Index partitions must have the same allocation state.
The following example illustrates how to create a unique index on a table:

```
CREATE UNIQUE INDEX custindex ON customer (cust_num);
```

The following example shows how to create an inactive word index with the specified description field:

```
CREATE PRO_WORD INDEX CommentsWordIdx ON customer (cust_num)
    on pub.customer
    PRO_DESCRIPTION 'Word index on comments field'
    PRO_ACTIVE 'n';
```

The following example shows how to define a multi-tenant index:

```
CREATE PRO_WORD INDEX CommentsWordIdx ON mtcustomer (cust_num)
    MULTI_TENANT
    FOR TENANT DEFAULT USING INDEX AREA "Misc_Index_Area"
    ;
```

Notes

- The first index you create on a table should be the fundamental key of the table. This index cannot be dropped except by dropping the table.
- An index slows down the performance of `INSERT, DELETE, and UPDATE` operations.
- Use `PROUTIL` to activate indexes.
- Use `CREATE INDEX` without the `PRO_ACTIVE {‘N’ | ‘n’}` attribute to create an inactive index. Active indexes can be created for an online database if the following conditions are met:
  - Run `CREATE INDEX` immediately after creating a table.
  - The index is created on the table.
  - Both the `CREATE TABLE` and `CREATE INDEX` are performed within the same transaction (no commit is performed after `CREATE TABLE` is run).

Authorization

To create a index, you must have `DBA` privileges or the `INDEX` privilege on the table.

Related statements

`ALTER TABLE, CREATE TABLE, DROP INDEX`
CREATE PROCEDURE

Creates a stored procedure. Stored procedures contain a Java code snippet that is processed into a Java class definition and stored in the database in text and compiled form. SQL applications invoke stored procedures through the SQL CALL statement or the procedure-calling mechanisms of ODBC and JDBC.

Syntax

```
CREATE PROCEDURE [ owner_name. ] procname
    ( [ parameter_decl [ , ... ] ] )
    [ RESULT ( column_name data_type [ , ... ] ) ]
    [ IMPORT
        java_import_clause ]
    BEGIN
    java_snippet
    END
```

`parameter_decl`

This is the syntax for `parameter_decl`:

```
{ IN | OUT | INOUT } parameter_name data_type
```

`owner_name`

Specifies the owner of the procedure. If the name is different from the user name of the user executing the statement, then the user must have DBA privileges.

`procname`

Names the stored procedure. DROP PROCEDURE statements specify the procedure name defined here. SQL also uses `procname` in the name of the Java class that it creates from the Java snippet.

`IN | OUT | INOUT`

Specifies whether following parameter declaration is input, output, or both.

Calling applications pass values for input parameters in the CALL statement or CALL escape sequence.

Stored procedures assign values to output parameters as part of their processing.

`INOUT` parameters have both a value passed in and receive a new value during procedure processing.

`parameter_name data_type`

Names a parameter and associates an SQL data type with it. The data type must be one supported by OpenEdge.
CREATE PROCEDURE

RESULT ( column_name data_type [ , ... ] )

Specifies columns in the result set the procedure returns. If the CREATE PROCEDURE statement includes this clause, the Java snippet must explicitly insert rows into the result set using the Java class SQLResultSet.

Note that the column_name argument is not used in the body of the stored procedure. Instead, methods of the Java classes refer to columns in the result set by ordinal number, not by name. The IMPORT keyword must be uppercase and on a separate line. The body is a sequence of Java statements between the BEGIN and END keywords. The Java statements become a method in a class that SQL creates and submits to the Java compiler. The BEGIN and END keywords must be uppercase and on separate lines.

The following example illustrates the use of the CREATE PROCEDURE statement:

Example

```
CREATE PROCEDURE get_sal ()
IMPORT
import java.math.*;
BEGIN
StringBuffer ename = new StringBuffer (20) ;
BigDecimal esal = new BigDecimal (2) ;
SQLCursor empcursor = new SQLCursor ("SELECT name, sal FROM emp ");
empcursor.open () ;
empcursor.fetch ();
while (empcursor.found ()){
  ename = (StringBuffer) empcursor.getValue (1, CHAR);
esal = (BigDecimal) empcursor.getValue (2, NUMERIC);
  // do something with the values here
}
empcursor.close () ;
END
```

Note

See OpenEdge Data Management: SQL Development for more information on using the CREATE statement and stored procedures.

Authorization

Must have DBA privilege, RESOURCE privilege, or ownership of procedure.

Related statements

CALL, DROP PROCEDURE
CREATE SEQUENCE

A sequence is an object for creating an incremental number series. Sequences can generate sequential values within any integer range with either positive or negative increments. The database holds the sequence definition and keeps track of the next available value.

To create a multi-tenant sequence, use the MULTI_TENANT attribute after defining all the other sequence attributes in the CREATE SEQUENCE syntax.

Syntax

CREATE SEQUENCE [schema_name]sequence_name
  [INCREMENT BY value],
  [START WITH value],
  [MAXVALUE value | NOMAXVALUE],
  [MINVALUE value | NOMINVALUE],
  [CYCLE | NOCYCLE]
  [MULTI_TENANT] ;

schema_name

Specifies the schema to contain the sequence. If schema_name is not specified, the sequence generator creates the sequence in the current schema. OpenEdge supports only the PUBLIC (PUB) schema.

sequence_name

Specifies the name of the sequence to be created.

INCREMENT BY

Specifies the interval between sequence numbers. The value can be a positive or negative integer (INTEGER data type for 32-bit sequences, BIGINT datatype for 64-bit sequences), but it cannot be 0. The value range for a 32-bit sequence is -2,147,483,648 to 2,147,483,647. The value range for a 64-bit sequence is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The value is positive for an ascending sequence and negative for a descending sequence. The default value is 1.

START WITH

Specifies the first number in the sequence generated. The value must be greater than or equal to the MINVALUE for an ascending sequence. The value must be less than or equal to the MAXVALUE for a descending sequence. The default value for an ascending sequence is MINVALUE and for a descending sequence it is MAXVALUE.
MAXVALUE

Specifies the maximum value to be generated for the sequence. The default value is -1 for a 32-bit and a 64-bit descending sequence. The default value is 2,147,483,647 for a 32-bit ascending sequence and 9223372036854775807 for a 64-bit ascending sequence.

NOMAXVALUE

Specifies -1 as the MAXVALUE for a 32-bit descending sequence and 2,147,483,647 as the MAXVALUE for a 32-bit ascending sequence. Specifies -1 as the MAXVALUE for a 64-bit descending sequence and 9223372036854775807 as the MAXVALUE for a 64-bit ascending sequence.

MINVALUE

Specifies the minimum value that the sequence can generate. The default value is 0 for an ascending sequence. The default value is -2,147,483,648 for a 32-bit descending sequence and -9223372036854775808 for a 64-bit sequence.

NOMINVALUE

Specifies 0 as the MINVALUE for an ascending sequence. The MINVALUE is -2,147,483,648 for a 32-bit descending sequence and -9223372036854775808 for a 64-bit sequence.

CYCLE

Indicates that the sequence will continue to generate a value after reaching the MAXVALUE (for an ascending sequence) or MINVALUE (for a descending sequence).

NOCYCLE

Indicates that the sequence cannot generate a value after reaching the MAXVALUE (for an ascending sequence) or MINVALUE (for a descending sequence). The SQL sequence generator uses NOCYCLE as the default, if CYCLE is not specified.

MULTI_TENANT

Specifies the sequence defined as a multi-tenant sequence.

Examples

The following example shows how a sequence generates unique customer numbers when a new customer is inserted into the pub.customer table:

```sql
CREATE SEQUENCE pub.customer_sequence
    START WITH 100,
    INCREMENT BY 1,
    NOCYCLE;
```
The following example shows how to define a multi-tenant sequence:

```
CREATE SEQUENCE pub.mtcustomer_sequence
    START WITH 100,
    INCREMENT BY 1,
    NOCYCLE
    MULTI_TENANT
;
```

**Authorization**

To create a sequence, you must have **DBA** privileges.

**Related Statements**

ALTER SEQUENCE, DROP SEQUENCE

---

**CREATE SUPER-TENANT**

Defines a tenant for a multi-tenant database. It also allocates database resources in the existing multi-tenant tables.

Defines a super-tenant for a multi-tenant database. A super-tenant is a user who can access and maintain data for any tenant defined in the multi-tenant database. There can be multiple super-tenant users created within a single database.

The multi-tenant database resources are not allocated automatically when you create a super-tenant. Privileges of a regular tenant, such as creating, selecting, inserting, updating, or deleting database objects are also applicable to the super-tenant.

The syntax for creating a super-tenant is given below:

**Syntax**

```
CREATE SUPER_TENANT super_tenant_name [ PRO_DESCRIPTION value ];
```

**super_tenant_name**

Specifies the name of the super-tenant. You must ensure that the name is unique within the set of tenants in a database.

**PRO_DESCRIPTION value**

Allows you to enter an ABL description. The value attribute is an arbitrary character string.
CREATE SYNONYM

Example

The following example shows how to create a SUPER_TENANT tenantadmin:

```
CREATE SUPER_TENANT tenantadmin;
```

Authorization

To create a super-tenant, you must have DBA privileges.

Related Statement

CREATE TENANT

CREATE SYNONYM

Creates a synonym for the specified table, view, or synonym. A synonym is an alias that SQL statements can use instead of the name specified when the table, view, or synonym was created.

Syntax

```
CREATE [ PUBLIC ] SYNONYM synonym
FOR [ owner_name.] {table_name | view_name | synonym };
```

PUBLIC

Specifies that the synonym is public: all users can refer to the name without qualifying it. By default, the synonym is private: other users must qualify the synonym by preceding it with the user name of the user who created it.

Users must have the DBA privilege to create public synonyms.

SYNONYM synonym

Name for the synonym.

FOR [ owner_name.] {table_name | view_name | synonym }

Table, view, or synonym for which SQL creates the new synonym.

Example

The following example demonstrates the use of the CREATE SYNONYM statement:

```
CREATE SYNONYM customer FOR smith.customer ;
CREATE PUBLIC SYNONYM public_suppliers FOR smith.suppliers ;
```
Authorization

Must have DBA privilege or RESOURCE privilege.

Related Statement

DROP SYNONYM

CREATE TABLE

Creates a table definition. A table definition consists of a set of named column definitions for data values that will be stored in rows of the table. SQL provides two forms of the CREATE TABLE statement.

The first syntax form explicitly specifies the definition of a column. The second syntax form, with the AS query_expression clause, implicitly defines the columns using the columns in a query expression. The third syntax form defines the new table as a multi-tenant table, and allocates storage area of a database to the tenants.

Syntax

```
CREATE TABLE [ [owner_name] ] table_name
    ( { column_definition | table_constraint }, ... )
    [ AREA area_name ]
    [ ENCRYPT WITH cipher ]
    [ BUFFER_POOL { PRIMARY | ALTERNATE } ]
    [ [ progress_table_attribute_keyword value ] ];

CREATE TABLE [ [owner_name] ] table_name
    [ (column_name [ NOT NULL ], ... ) ]
    [ AREA area_name ]
    [ ENCRYPT WITH cipher ]
    [ BUFFER_POOL { PRIMARY | ALTERNATE } ]
    AS query_expression
    ;

CREATE TABLE [ [owner_name] ] table_name
    ( { column_definition | table_constraint }, ... )
    [ AREA area_name ]
    [ ENCRYPT WITH cipher ]
    [ BUFFER_POOL { PRIMARY | ALTERNATE } ]
    [ [ progress_table_attribute_keyword value ] ]
    MULTI_TENANT
    [ FOR TENANT { tenant_name_1 | DEFAULT } ]
    [ USING TABLE AREA table_area_name ]
    [ USING INDEX AREA index_area_name ]
    [ USING LOB AREA lob_area_name ]
    [ ... ]
    [ FOR TENANT { tenant_name_2 | DEFAULT } [ USING NO SPACE ] ]
    ;
```
column_definition

Defines a column. The syntax for column_definition is given below:

Syntax

```
column_name  data_type
  [ COLLATE  case_insensitive | case_sensitive ]
  [ DEFAULT { literal | NULL | SYSDATE | SYSTIME | SYSTIMESTAMP } ]
  [ column_constraint [ column_constraint, ... ] ]
  [ progress_column_attribute_keyword value ], ...]
```

column_name  data_type

Names a column and associates a data type to it. The name of the column must be different from other column names already defined in the table. The data_type must be supported by OpenEdge.

When a table contains more than one column, a comma separator should be used after each column_definition, except for the final column_definition.

COLLATE

Indicates the case sensitivity of the column. The default value for the COLLATE element is case_sensitive.

case_insensitive

Indicates that the column will be case-insensitive. The word case_insensitive itself cannot be used as a valid input. The value for the case_insensitive clause can only be _I, I, or the default database collation with the suffix _I (For example: COLLATE_I, COLLATE I, or COLLATE BASIC_I).

case_sensitive

Indicates that the column will be case-sensitive. The word case_sensitive itself cannot be used as a valid input. The value for the case_sensitive clause can only be _S, S, or the default database collation with the suffix _S (For example: COLLATE_S, COLLATE S, or COLLATE BASIC_S).
DEFAULT

Specifies an explicit default value for a column. The column takes on the
default value if an INSERT statement does not include a value for the column.
If a column definition omits the DEFAULT clause, NULL value gets assigned.

The DEFAULT clause accepts the arguments shown in the table below:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>literal</td>
<td>An integer, numeric, or a string constant.</td>
</tr>
<tr>
<td>NULL</td>
<td>A null value.</td>
</tr>
</tbody>
</table>
| SYSDATE  | Displays the current date. Valid only for columns
defined with the DATE data type. SYSDATE is
equivalent to the Progress default keyword TODAY. The argument represents a DATE value. |
| SYSTIME  | Displays the current time. The argument represents a
TIME value. |
| SYSTIMESTAMP | Displays the current date and time. The argument
represents a TIMESTAMP value. |

column_constraint

Specifies a constraint that is applied when you insert or update a value in the
associated column.

progress_column_attribute_keyword value

ABL column attribute keyword and value.

The syntax used to define a LOB column is shown below:

Syntax

```
{ LVARCHAR | CLOB | LVARBINARY | BLOB } [ ( length ) ]
[ AREA area_name ]
[ ENCRYPT WITH cipher ]
[ BUFFER_POOL { PRIMARY | ALTERNATE } ]
```

table_constraint

Specifies a constraint that is applied when you insert or update a row in the table.

AREA area_name

Specifies the name of the storage area where data of the table is to be stored. The
name of the storage area must be specified within double quotes.

If the specified area does not exist, the database returns an error. If you do not
specify an area, the default area is used.
CREATE TABLE

ENCRIPT WITH cipher

Allows the table to be encrypted by designating an appropriate cipher.

BUFFER_POOL { PRIMARY | ALTERNATE }

Allows the table to be assigned to a primary or alternate buffer pool.

progress_table_attribute_keyword value

ABL table attribute keyword and value.

AS query_expression

Specifies a query expression to be used for the data type and data values for the columns of the table. The type and length of the columns of the query_expression result become the type and length of the respective columns in the table you created. The rows in the resultant set of the query_expression are inserted into the table after its creation. The column names are optional in this form of the CREATE TABLE statement. If it is omitted, the names of the columns of the table are taken from the column names of the query_expression.

MULTI_TENANT

Defines the table as a multi-tenant table. The MULTI_TENANT phrase can occur only after the column_definition and the progress_table_attribute_keyword, and before the AS query_expression part of the CREATE TABLE syntax.

FOR TENANT tenant_name

Specifies the name of a tenant. The FOR TENANT phrase can be repeated for as many existing tenants as required.

TABLE AREA area_name

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE TABLE statement execution. Specifying the TABLE AREA is optional.

Specifies the name of the default storage area allocated for the tenant.

INDEX AREA index_area_name

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE TABLE statement execution. Specifying the INDEX AREA is optional.

If the storage area for an index is omitted in the syntax, then any index will be allocated in the table area.

To define INDEX AREA, you must define the TABLE AREA.
LOB AREA lob_area_name

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE TABLE statement execution. Specifying the LOB AREA is optional.

If the storage area for a LOB is omitted in the syntax, then any LOB will be allocated in the table area.

To define the LOB AREA, you must define the TABLE AREA.

PRO_DESCRIPTION value

Allows you to enter ABL description. value is an arbitrary character string.

USING NO SPACE

Indicates that no space is allocated for index data storage in the table. This phrase is valid only if the tenant partition for the table is allocated NO SPACE.

Note: For a particular tenant, the table and the Index partitions must have the same allocation state.

Examples

The following example shows that the user who issues the CREATE TABLE statement must have REFERENCES privilege on the itemno column of the table john.item in CREATE TABLE supplier_item:

```
CREATE TABLE supplier_item (    supp_no INTEGER NOT NULL PRIMARY KEY,    item_no INTEGER NOT NULL REFERENCES john.item (itemno),    qty INTEGER ) ;
```

The table is created in the current owner schema.

The following CREATE TABLE statement explicitly specifies a table owner, gus:

```
CREATE TABLE gus.account (    account integer,    balance numeric (12,2),    info char (84) ) ;
```

The following example shows the AS query_expression form of CREATE TABLE to create and load a table with a subset of the data in the customer table:

```
CREATE TABLE dealer (name, street, city, state) AS SELECT name, street, city, state FROM customer WHERE state IN ('CA', 'NY', 'TX') ;
```
CREATE TABLE

The following example includes a **NOT NULL** column constraint and **DEFAULT** clause for definition of a column:

```sql
CREATE TABLE emp (  
  empno integer NOT NULL,
  deptno integer DEFAULT 10,
  join_date date DEFAULT NULL
) ;
```

The following example shows how to create a table with two columns. Both of them have ABL descriptions and specified column labels:

```sql
CREATE TABLE emp (  
  empno INTEGER NOT NULL UNIQUE
  PRO_DESCRIPTION 'A unique number for each employee'
  PRO_COL_LABEL 'Employee No.'
  deptno INTEGER DEFAULT 21 NOT NULL
  PRO_DESCRIPTION 'The department number of the employee'
  PRO_COL_LABEL 'Dept. No.'
)  
PRO_HIDDEN 'Y' PRO_DESCRIPTION 'All Employees';
```

A hidden table is created and it has a specified description.

The following example shows how to create a multi-tenant table that overrides areas of the selected tenant table partition:

```sql
CREATE TABLE pub.acct_payable (acct_num    integer, debit_date   date)
MULTI_TENANT
FOR TENANT Consolidated_Freightways
  USING TABLE AREA "CF Financial area"
  USING INDEX AREA "CF Fin idx area"
FOR TENANT "Mega Media Networks"
  USING TABLE AREA "MMN Data area"
  USING INDEX AREA "MMN index area"
  USING LOB AREA "MMN pix area" ;
FOR TENANT DEFAULT USING NO SPACE ;
```

The following example shows how to create a multi-tenant table which uses the default area for all the table partition of a tenant. In this case, the **DEFAULT** tenant is allocated no space in the database storage area:

```sql
CREATE TABLE pub.mtcustomer (cust_num    integer, hire_date   date)
MULTI_TENANT;
```

The following example shows how to create a multi-tenant table which uses the default area for all areas tenants except the **DEFAULT** tenant:

```sql
CREATE TABLE pub.mtcustomer (cust_num    integer, hire_date   date)
MULTI_TENANT
  FOR TENANT DEFAULT USING TABLE AREA "Scratch_Data_Area"
  USING INDEX AREA "Misc_Index_Area";
```
CREATE TENANT

Note

Table columns defined in OpenEdge SQL have default format values identical to those created by the Data Dictionary. Thus, columns created by SQL will have the same default format as columns created by ABL tools.

Authorization

To create a table, you must have DBA privileges, and the RESOURCE privilege.

Related Statement

DROP TABLE

CREATE TENANT

The CREATE TENANT statement can be used to:

- Define a tenant for a multi-tenant database
- Allocate database resources in the existing multi-tenant tables
- Add a new tenant to an existing multi-tenant group that is defined for a multi-tenant table

The table name and group name specified in the FOR TABLE clause of the CREATE TENANT statement cannot be repeated in another instance of the FOR TABLE clause of the same statement. You can add the tenant to only one group for a given table. However, you can add it to multiple groups on different tables. No new partition is created for the table specified in the FOR TABLE clause.

Syntax

```
CREATE TENANT tenant_name TABLE AREA area_name
    [ INDEX AREA area_name_2 ]
    [ LOB AREA area_name_3 ]
    [ PRO_DESCRIPTION value ]
    [ FOR TABLE [ owner_name ]table_name_1 { [ USING TABLE AREA table_area_name ]
        [ USING INDEX AREA index_area_name ]
        [ USING LOB AREA lob_area_name ]
        [ USING NO SPACE ]
        [ JOIN GROUP group_name ] } ] ...
    [ FOR TABLE [ owner_name ]table_name_2 { [ USING NO SPACE ] } ] ]
;
```

tenant_name

Specifies the name of the tenant for the multi-tenant table.

TABLE AREA  area_name

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE TENANT statement execution. The TABLE AREA specification is optional. The name of the storage area must be specified within double quotes.
INDEX AREA  index_area_name

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE TENANT statement execution. The INDEX AREA specification is optional. The name of the storage area must be specified within double quotes.

If the storage area for an index is omitted in the syntax, then any index will be allocated in the table area.

LOB AREA  lob_area_name

Is used to over-ride the default storage areas associated with the particular tenant. This area is used for database space allocation during the CREATE TENANT statement execution. The LOB AREA specification is optional. The name of the storage area must be specified within double quotes.

If the storage area for a LOB is omitted in the syntax, then any LOB will be allocated in the table area.

PRO_DESCRIPTION  value

Allows you to enter the ABL description. The value attribute is an arbitrary character string.

FOR TABLE

Indicates the default area, which can be overridden by a specific area for a particular table, or for its index, or for its LOB areas. An area can be specified for the storage of data of table and index.

USING NO SPACE

Indicates that no space is allocated for index data storage for the table. This phrase is valid only if the tenant partition for the table is allocated NO SPACE.

Note: For a particular tenant, the table and the Index partitions must have the same allocation state.

table_name

Specifies the table name. The name of a referenced table must be valid for a multi-tenant database.

owner_name

Specifies the name of the schema that qualifies the table.

JOIN GROUP

Adds the tenant to the specified group.

group_name

Specifies the name of the group that you want the tenant to join.
Example

The following example shows how to create a tenant by overriding an area for the selected table partition and how to add the tenant to a group:

```
CREATE TENANT Consolidated_Freightways TABLE AREA "CF Data area"
  INDEX AREA "CF index area"
FOR TABLE pub.customer USING TABLE AREA "CF cust area"
  USING INDEX AREA "CF cust idx area"
FOR TABLE pub.acct_payable USING TABLE AREA "CF Financial area"
  USING INDEX AREA "CF Fin idx area"
FOR TABLE pub.farm_location USING NO SPACE
FOR TABLE pub.archive_10yr JOIN GROUP Joint_Archives;
```

The following two examples use the default area for the partition of all the tenant tables:

```
CREATE TENANT Mega_Media_Networks TABLE AREA "MMM Data area"
  INDEX AREA "MMM index area"
  LOB AREA "MMM pix area"

CREATE TENANT Mega_Media_Networks TABLE AREA "MMM Data area"
  INDEX AREA "MMM index area"
  LOB AREA "MMM pix area"
FOR TABLE pub.archive_10yr USING NO SPACE;
```

Authorization

To create a tenant, you must have DBA privileges.

Related Statement

DROP TENANT
CREATE TRIGGER

Creates a trigger for the specified table. A trigger is a special type of automatically executed stored procedure that helps ensure referential integrity for a database.

Triggers contain Java source code that can use SQL Java classes to carry out database operations. Triggers are automatically activated when an `INSERT`, `UPDATE`, or `DELETE` statement changes the trigger’s target table. The Java source code details what actions the trigger takes when it is activated.

Syntax

```
CREATE TRIGGER [ owner_name ] trigname
  { BEFORE | AFTER }
  { INSERT | DELETE | UPDATE [ OF column_name [ , ... ] ] }
  ON table_name
  [ REFERENCING { OLDROW [ ,NEWROW ] | NEWROW [ ,OLDROW ] } ]
  [ FOR EACH { ROW | STATEMENT } ]
  [ IMPORT java_import_clause ]
  BEGIN
    java_snippet
  END
```

`owner_name`

Specifies the owner of the trigger. If the name is different from the user name of the user executing the statement, then the user must have `DBA` privileges.

`trigname`

Names the trigger. `DROP TRIGGER` statements specify the trigger name defined here. SQL also uses `trigname` in the name of the Java class that it creates from the Java snippet.

`BEFORE` | `AFTER`

Denotes the trigger action time. The trigger action time specifies whether the triggered action, implemented by `java_snippet`, executes `BEFORE` or `AFTER` the invoking `INSERT`, `UPDATE`, or `DELETE` statement.

`INSERT | DELETE | UPDATE [ OF column_name [ , ... ] ]`

Denotes the trigger event. The trigger event is the statement that activates the trigger.

If `UPDATE` is the triggering statement, this clause can include an optional column list. Only updates to any of the specified columns will activate the trigger. If `UPDATE` is the triggering statement and does not include the optional column list, then any `UPDATE` on the table will activate the trigger.
CREATE TRIGGER

ON table_name

Identifies the name of the table where the trigger is defined. A triggering statement that specifies table_name causes the trigger to execute. table_name cannot be the name of a view.

REFERENCING OLDROW [ , NEWROW ] | NEWROW [ , OLDROW ]

Provides a mechanism for SQL to pass row values as input parameters to the stored procedure implemented by java_snippet. The code in java_snippet uses the getValue method of the NEWROW and OLDROW objects to retrieve values of columns in rows affected by the trigger event and store them in procedure variables. This clause is allowed only if the trigger specifies the FOR EACH ROW clause.

The meaning of the OLDROW and NEWROW arguments of the REFERENCING clause depends on whether the trigger event is INSERT, UPDATE, or DELETE. For example:

- INSERT...REFERENCING NEWROW means the triggered action can access values of columns of each row inserted. SQL passes the column values specified by the INSERT statement.

- INSERT...REFERENCING OLDROW is meaningless, since there are no existing values for a row being inserted. INSERT...REFERENCING OLDROW generates a syntax error.

- UPDATE...REFERENCING OLDROW means the triggered action can access the values of columns, before they are changed, of each row updated. SQL passes the column values of the row as it exists in the database before the update operation.

- DELETE...REFERENCING OLDROW means the triggered action can access values of columns of each row deleted. SQL passes the column values of the row as it exists in the database before the delete operation.

- DELETE...REFERENCING NEWROW is meaningless, since there are no new existing values to pass for a row being deleted. DELETE...REFERENCING NEWROW generates a syntax error.

- UPDATE is the only triggering statement that allows both NEWROW and OLDROW in the REFERENCING clause.

- UPDATE...REFERENCING NEWROW means the triggered action can access the values of columns, after they are changed, of each row updated. SQL passes the column values specified by the UPDATE statement.

- The trigger action time (BEFORE or AFTER) does not affect the meaning of the REFERENCING clause. For instance, BEFORE UPDATE...REFERENCING NEWROW still means the values of columns after they are updated will be available to the triggered action.

- The REFERENCING clause generates an error if the trigger does not include the FOR EACH ROW clause.
CREATE TRIGGER

FOR EACH { ROW | STATEMENT }

Controls the execution frequency of the triggered action implemented by java_snippet.

FOR EACH ROW means the triggered action executes once for each row being updated by the triggering statement. CREATE TRIGGER must include the FOR EACH ROW clause if it also includes a REFERENCING clause.

FOR EACH STATEMENT means the triggered action executes only once for the whole triggering statement. FOR EACH STATEMENT is the default.

IMPORT java_import_clause

Specifies standard Java classes to import. The IMPORT keyword must be uppercase and on a separate line.

BEGIN

java_snippet

END

Denotes the body of the trigger or the triggered action. The body contains the Java source code that implements the actions to be completed when a triggering statement specifies the target table. The Java statements become a method in a class that SQL creates and submits to the Java compiler.

The BEGIN and END keywords must be uppercase and on separate lines.

Notes

• Triggers can take action on their own table so that they invoke themselves. SQL limits such recursion to five levels.

• You can have multiple triggers on the same table. Multiple UPDATE triggers on the same table must specify different columns. SQL executes all triggers applicable to a given combination of table, trigger event, and action time.

• The actions carried out by a trigger can fire another trigger. When this happens, the other trigger’s actions execute before the rest of the first trigger finishes executing.

• If a constraint and trigger are both invoked by a particular SQL statement, SQL checks constraints first, so any data modification that violates a constraint does not also fire a trigger.

• To modify an existing trigger, you must delete it and issue another CREATE TRIGGER statement. You can query the systrigger system table for information about the trigger before you delete it.

• The code in java_snippet uses the getValue method of the NEWROW and OLDROW objects. The getValue method is valid on OLDROW before or after an update or delete and NEWROW before or after an update or insert; the setValue method is only valid on NEWROW before an insert or update.
**Example**

The following code segment illustrates how to use the `CREATE TRIGGER` statement:

```sql
CREATE TRIGGER TRG_TEST04 BEFORE INSERT ON tst_trg_01
REFERENCING NEWROW
FOR EACH ROW
IMPORT
    import java.sql.*;
BEGIN
    //Inserting Into tst_trg_03
    Integer new_value=newInteger(0);
    new_value=(Integer)NEWROW.getValue(1,INTEGER);
    SQLIStatement insert_tst3=new SQLIStatement("INSERT INTO tst_trg_03 values (?)");
    insert_tst3.setParam(1,new_value);
    insert_tst3.execute();
END
```

The following code segment illustrates how to set values for a new row in the `CREATE TRIGGER` statement:

```sql
CREATE TRIGGER trg1403
BEFORE INSERT ON tbl1401
REFERENCING NEWROW
FOR EACH ROW
IMPORT
    import java.sql.*;
BEGIN
    INTEGER n2 = new INTEGER(12345);
    NEWROW.setValue(2, n2);
END
```

For more information on creating and using triggers, see *OpenEdge Data Management: SQL Development*.

**Authorization**

Must have the **DBA** privilege or **RESOURCE** privilege.

**Related statements**

DROP TRIGGER
CREATE USER

Creates the specified user.

Syntax

```
CREATE USER { 'username' | 'username@domain_name' }, 'password' ;
```

`username`

Specifies the username. The username must be enclosed in single quotes.

`domain_name`

Specifies the domain name that the user is associated with. The `username@domain_name` must be enclosed in single quotes.

`password`

Specifies the password for the user. The password must be enclosed in single quotes.

Example

In this example an account with DBA privileges creates the `username` 'Jasper' with password 'spaniel':

```
CREATE USER 'Jasper', 'spaniel' ;
```

In the following example, `CREATE USER` creates the user Jasper to connect to mtdomain domain with the password spaniel:

```
CREATE USER 'Jasper@mtdomain', 'spaniel' ;
```

The user Jasper should be associated with the tenant identified by the domain mtdomain.

Notes

- You are strongly advised to NOT create a user named PUB. A user named PUB is inherently the owner of all tables created in the ABL and all schema tables, since these are all in the PUB schema. As the owner, a user PUB has full access to those tables, including the ability to read and write data, and the ability to drop the application table. Therefore, the existence of a user PUB creates a very serious security risk for the database. For these reasons, please do NOT create a user named PUB.

- Used in conjunction with `BEGIN-END DECLARE SECTION` and `DROP USER` statement, the `CREATE USER` statement provides a way to manage user records through SQL.

- The user name and password must be enclosed in quotes.

- Before issuing the `CREATE USER` statement, there are no users defined in the user table and any user can log into the database.
• After issuing the `CREATE USER` statement, only users defined in the user table can log into the database.

• If you are creating a user in a multi-tenant table, the created user should be associated with the tenant identified by the domain name defined in the syntax.

• The `username@domainame` can be equivalently defined in OpenEdge ABL. For more information on defining a user name in ABL, see *OpenEdge Development: ABL Reference*.

**Authorization**

Must have **DBA** privileges.

**Related statements**

`BEGIN-END DECLARE SECTION, DROP USER`

---

**CREATE VIEW**

Creates a view with the specified name on existing tables or views.

**Syntax**

```
CREATE VIEW [ owner_name.]view_name
            [ ( column_name, column_name,... ) ]
    AS [ ( ] query_expression [ ) ]
    [ WITH CHECK OPTION ];
```

`owner_name`

Owner of the created view.

`{ column_name, column_name,... }`

Specifies column names for the view. These names provide an alias for the columns selected by the query specification. If the column names are not specified, then the view is created with the same column names as the tables or views on which it is based.

**WITH CHECK OPTION**

Checks that the updated or inserted row satisfies the view definition. The row must be selectable using the view. The `WITH CHECK OPTION` clause is only allowed on an updatable view.
**Notes**

- A view is deletable if deleting rows from that view is allowed. For a view to be deletable, the view definition must satisfy the following conditions:
  - The first `FROM` clause contains only one table reference or one view reference.
  - There are no aggregate functions, `DISTINCT` clause, `GROUP BY` clause, or `HAVING` clause in the view definition.
  - If the first `FROM` clause contains a view reference, then the view referred to is deletable.

- A view is updatable if updating rows from that view is allowed. For a view to be updatable, the view has to satisfy the following conditions:
  - The view is deletable (it satisfies all the previously specified conditions for deletability).
  - All the select expressions in the first `SELECT` clause of the view definition are simple column references.
  - If the first `FROM` clause contains a view reference, then the view referred to is updatable.

- A view is insertable if inserting rows into that view is allowed. For a view to be insertable, the view has to satisfy the following conditions:
  - The view is updatable (it satisfies all the previously specified conditions for update ability).
  - If the first `FROM` clause contains a table reference, then all `NOT NULL` columns of the table are selected in the first `SELECT` clause of the view definition.
  - If the first `FROM` clause contains a view reference, then the view referred to is insertable.

**Example**

The following examples illustrate `CREATE VIEW` statements defined by query expressions:

```
CREATE VIEW ne_customers AS
  SELECT name, address, city, state
  FROM customer
  WHERE state IN ( 'NH', 'MA', 'ME', 'RI', 'CT', 'VT' )
  WITH CHECK OPTION ;

CREATE VIEW OrderCount (custnum, numorders) AS
  SELECT CustNum, COUNT(*)
  FROM Order
  GROUP BY CustNum;
```
Authorization

Must have DBA privilege, RESOURCE privilege, or SELECT privilege.

Related statements

DROP VIEW

DELETE

Deletes zero, one, or more rows from the specified table that satisfy the search condition specified in the WHERE clause. If the optional WHERE clause is not specified, then the DELETE statement deletes all rows of the specified table.

When deleting row(s) of a multi-tenant table, a regular tenant can only delete rows in its partition, but a super-tenant or a DBA can delete rows in all the tenant partitions and group partitions. A super-tenant or a DBA may refine which tenants’ rows must be affected by using the tenantid_tbl() or the tenantName_tbl() function in the WHERE clause search_condition.

Syntax

```
DELETE FROM [ owner_name ] { table_name | view_name }
 [ WHERE search_condition ] ;
```

Example

The following example illustrates the DELETE statement:

```
DELETE FROM Customer WHERE Name = 'Surf and Sport';
```

The following example deletes the row from a multi-tenant table using custnum and tenantid_tbl() in the WHERE clause.

```
DELETE FROM mtcustomer
WHERE custnum = 1428 AND tenantName_tbl (pub.mtcustomer) = 'SNCSoftware';
```

Note

If the table has primary or candidate keys and there are references from other tables to the rows to be deleted, the statement is rejected.

Authorization

To delete a row from the table, you must have DBA privileges, ownership of the table, or DELETE permission of the table.

A user mapped to a super-tenant can view data for all the tenants only if the user has the required permissions on the tables referenced in the statement.

Related statements

WHERE clause
DISCONNECT CATALOG

Removes a connection to an auxiliary read-only database.

Syntax

```sql
DISCONNECT CATALOG catalog_name;
```

catalog_name

Catalog name to be used as an alias for the database in schema, table and column references. This must be in the form of an SQL identifier.

Example

In this example, the auxiliary database connection identified by the catalog named `mydb1` is removed:

```sql
DISCONNECT CATALOG mydb1;
```

Note

Used to remove auxiliary connections established by executing the `CONNECT AS CATALOG` statement.

Authorization

Any user is allowed to execute this statement.

SQL Compliance

Progress Software Corporation specific extension

Related statements

`CONNECT AS CATALOG`, `SET CATALOG`

DROP DOMAIN

Using the drop domain, you can delete a security domain.

Syntax

```sql
DROP DOMAIN domain_name;
```

domain_name

Specifies the name of the security domain.

Note: To drop a domain, you must first drop all the users associated with the domain.
Example

The syntax shown below shows how to drop a domain.

```
DROP DOMAIN jasper;
```

Note

Each database contains tenants, users, and domains. A DBA must complete the steps specified below to drop a tenant from the database:

1. DROP USER for every user in the domain. For more information on dropping a user.
2. DROP DOMAIN for every domain defined for the tenant.
3. DROP TENANT.

Authorization

To drop a domain, you must have DBA privileges.

Related Statement

CREATE DOMAIN

---

**DROP GROUP**

Deletes a group from a multi-tenant database and deallocates the database resources in the group partition of the multi-tenant table for which the group is defined.

Syntax

```
DROP GROUP group_name;
```

*group_name*

Specifies the name of the group that you want to delete from the multi-tenant database.

Example

The following example illustrates the `DROP GROUP` statement:

```
DROP GROUP carz_group;
```

Note

The specified group must be an existing valid group in the database. All the tenants must be disassociated from the group before dropping it.

Authorization

You must have DBA privilege to drop a group.

Related statements

CREATE GROUP, SHOW GROUP
DROP INDEX

Deletes an index on the specified table.

If a multi-tenant index is dropped, then the DROP INDEX statement deletes the index for every tenant defined in the table.

**Syntax**

```
DROP INDEX [ index_owner_name ] index_name 
  ON [ table_owner_name ] table_name ;
```

*index_owner_name*

Specifies the name of the index owner. If *index_owner_name* is specified and is different from the name of the user executing the statement, then the user must have DBA privileges.

*index_name*

Verifies the *index_name* corresponding to the table.

**Example**

The following example illustrates the DROP INDEX statement:

```
DROP INDEX custindex ON customer;
```

You cannot drop the first index created on a table, except by dropping the table.

**Authorization**

Must have DBA privilege or ownership of the index.

**Related statements**

CREATE INDEX

---

DROP PROCEDURE

Deletes a stored procedure.

You can use the DROP PROCEDURE statement as an online and offline operation. For databases enabled for OpenEdge Replication the DROP PROCEDURE statement is an offline operation. For databases without OpenEdge Replication the DROP PROCEDURE statement is an online operation.

**Syntax**

```
DROP PROCEDURE [ owner_name. ] procedure_name ;
```
owner_name

Specifies the owner of the procedure.

procedure_name

Name of the stored procedure to delete.

Example

The following example illustrates the DROP PROCEDURE statement:

```
DROP PROCEDURE new_sal;
```

Authorization

Must have DBA privilege or owner of a stored procedure.

Related statement

CALL, CREATE PROCEDURE

---

DROP SEQUENCE

The DROP SEQUENCE statement removes a sequence from a schema. The sequence can be in a user’s schema or another schema may be specified. You must have DBA privileges to remove a sequence in a schema other than your own.

If a multi-tenant sequence is dropped, then the DROP SEQUENCE statement drops the sequence instance for every tenant defined in the multi-tenant table.

Syntax

```
DROP SEQUENCE [ schema_name ] sequence_name;
```

Authorization

To delete a sequence, you must have DBA privileges or ownership of the table.

Example

The following is an example of the DROP SEQUENCE statement:

```
DROP SEQUENCE pub.customer;
```
DROP SYNONYM

Related Statement

CREATE SEQUENCE

DROP SYNONYM

Drops the specified synonym.

Syntax

```
DROP [ PUBLIC ] SYNONYM synonym ;
```

PUBLIC

Specifies that the synonym was created with the PUBLIC argument.

SYNONYM synonym

Name for the synonym.

Example

The following is an example of the DROP SYNONYM statement:

```
DROP SYNONYM customer ;
DROP PUBLIC SYNONYM public_suppliers ;
```

Notes

- If DROP SYNONYM specifies PUBLIC and the synonym was not a public synonym, SQL generates the “base table not found” error.
- If DROP SYNONYM does not specify PUBLIC and the synonym was created with the PUBLIC argument, SQL generates the “base table not found” error.

Authorization

Must have DBA privilege or ownership of the synonym (for DROP SYNONYM).

Related statement

CREATE SYNONYM
DROP TABLE

Deletes the specified table.

If a multi-tenant table is dropped, then, the DROP TABLE statement deletes the table partition for every tenant defined in the table.

You can use the DROP TABLE statement as an online and offline operation. For further information, see the Notes section below.

Syntax

```
DROP TABLE [ owner_name ] table_name ;
```

**owner_name**

Specifies the owner of the table.

**table_name**

Name of the table to be dropped.

Example

The following is an example of the DROP TABLE statement:

```
DROP TABLE customer ;
```

Notes

- If owner_name is specified and is different from the name of the user executing the statement, then the user must have DBA privileges.
- When a table is dropped, the indexes on the table and the privileges associated with the table are dropped automatically.
- Views dependent on the dropped table are not automatically dropped, but become invalid.
- If the table is part of another table’s referential constraint (if the table is named in another table’s REFERENCES clause), the DROP TABLE statement fails. You must DROP the referring table first.
- When the DROP TABLE statement is applied to a multi-tenant table, it deletes the table for every tenant defined in the table.
- The DROP TABLE statement is an online operation for the following cases:
  - All data in the Type II area

  **Note:** If tables are in the Type II area but have no indexes created on them, then the DROP TABLE statement for those tables will be an offline operation.
  - Tables that are not multi-tenant
  - Data in schema other than PUB
DROP TENANT

- Databases without JTA transactions
- Tables without OpenEdge Replication

- The **DROP TABLE** statement is an offline operation for the following cases:
  - Table objects (indexes and LOBs) and tables in areas other than the Type II area
  - Multi-tenant tables
  - Tables in PUBLIC (PUB) schema
  - JTA transactions
  - Tables with OpenEdge Replication

**Authorization**

To delete a table, you must have **DBA** privileges or ownership of the table.

**Related statement**

CREATE TABLE

---

**DROP TENANT**

Deletes a tenant from a multi-tenant database.

**Syntax**

```
DROP TENANT [ owner_name]tenant_name;
```

- **tenant_name**
  Specifies the name of the tenant you want to delete. The name of the tenant must exist in the database.

- **owner_name**
  Specifies the name of the schema which qualifies the table.
### Example

The following example shows how to drop the tenant, `mtcustomer`, from the multi-tenant table:

```
DROP TENANT pub.mtcustomer;
```

### Note

Each database contains users, domains, and tenants. A DBA must follow the order specified below to drop a tenant from the database:

1. DROP USER for every user in the domain.
2. DROP DOMAIN for every domain defined for the tenant.
3. DROP TENANT.

If a tenant that is being dropped belongs to one or more groups, it is first disassociated from the group(s) before being dropped.

### Authorization

To drop a tenant, you must have **DBA** privileges.

### Related Statement

**CREATE TENANT**

---

### DROP TRIGGER

Deletes a trigger.

#### Syntax

```
DROP TRIGGER [ owner_name.]trigger_name ;
```

- **owner_name**
  - Specifies the owner of the trigger.

- **trigger_name**
  - Names the trigger to drop.

#### Example

The following is an example of the **DROP TRIGGER** statement:

```
DROP TRIGGER sal_check ;
```
Authorization

Must have DBA privilege or ownership of the trigger.

Related statement

CREATE TRIGGER

DROP USER

Deletes the specified user.

Syntax

```
DROP USER { 'username' | 'username@domain_name' };
```

username

Specifies the user name to be deleted. The username must be enclosed in single quotes.

domain_name

Specifies the domain name that the user is associated with. The `username@domain_name` must be enclosed in single quotes.

Example

In this example, an account with DBA privileges drops the username 'Jasper':

```
DROP USER 'Jasper';
```

The example displayed below shows how to drop a user associated with a domain. An user with DBA privileges deletes the user `Jasper` associated with the domain name `mtdomain`:

```
DROP USER 'Jasper@mtdomain';
```

Note

In a database with tenants, users, and domains, a DBA must complete the following steps to drop a tenant from the database:

1. DROP USER for every user in the domain.
2. DROP DOMAIN for every domain defined for the tenant. For more information on dropping a user.
3. DROP TENANT.
Authorization

Must have DBA privileges.

Related statements

BEGIN-END DECLARE SECTION, CREATE USER

DROP VIEW

Deletes the view from the database.

Syntax

```
DROP VIEW [ owner_name. ]view_name ;
```

owner_name

Specifies the owner of the view.

view_name

Names the view to drop.

Example

The following is an example of the DROP VIEW statement:

```
DROP VIEW newcustomers ;
```

Notes

- If owner_name is specified and is different from the name of the user executing the statement, then the user must have DBA privileges.
- When a view is dropped, other views that are dependent on this view are not dropped. The dependent views become invalid.

Authorization

Must have DBA privilege or ownership of the view.

Related statement

CREATE VIEW
GRANT

Grants various privileges to the specified users of the database. There are two forms of the **GRANT** statement:

- Grant database-wide privileges, such as system administration (**DBA**), general creation (**RESOURCE**), audit administration (**AUDIT_ADMIN**), audit archive (**AUDIT_ARCHIVE**), or audit insert (**AUDIT_INSERT**).
- Grant various privileges on specific tables and views. Privilege definitions are stored in the system tables **SYSDBAUTH**, **SYSTABAUTH**, and **SYSCOLAUTH** for the database, tables, and columns, respectively.

**Note:** You must use separate commands to grant **DBA** or **RESOURCE** privileges with any of the **AUDIT** privileges. Using the same command to grant a user with **DBA** or **RESOURCE** privileges and any of the **AUDIT** privileges results in an error.

You must use the fully qualified username to grant privileges to tenant data in a multi-tenant table.

**Syntax**

```
GRANT { RESOURCE, DBA, AUDIT_ADMIN, AUDIT_ARCHIVE, AUDIT_INSERT }
    TO user_identifier [, user_identifier ] , ...
    [ WITH GRANT OPTION ];
```

**user_identifier**

Identifies a username. For a tenant user, you must mention the fully qualified user name, `username@domain_name`, to grant access to a user.

**Syntax**

```
username | username@domain_name
```

**Note:** You must not use the commands, such as:

- GRANT **DBA** to 'user' WITH GRANT OPTION
- GRANT **RESOURCE** to 'user' WITH GRANT OPTION.

These commands return syntax errors.

In this example, audit administration and audit archive privileges are granted to bsmith:

```
GRANT AUDIT_ADMIN, AUDIT_ARCHIVE TO bsmith WITH GRANT OPTION;
```

Because these privileges are granted to bsmith **WITH GRANT OPTION**, bsmith may now grant these two privileges to other users.
This is the syntax to grant privileges on specific tables and views:

**Syntax**

```
GRANT { privilege [, privilege ] , . . . | ALL [ PRIVILEGES ] } 
    ON table_name 
TO { username [, username ] , . . . | PUBLIC } 
[ WITH GRANT OPTION ] ;
```

This is the syntax for the `privilege` variable:

**Syntax**

```
{ SELECT | INSERT | DELETE | INDEX 
    | UPDATE [ ( column , column , . . . ) ] 
    | REFERENCES [ ( column , column , . . . ) ] } 
```

Use the following syntax to assign sequence privileges:

**Syntax**

```
GRANT { SELECT | UPDATE } 
ON SEQUENCE schema.sequence 
TO user_name [, user_name] ... 
```

**SELECT**

Allows specified user to read data from the sequence.

**UPDATE**

Allows specified user to modify data for the sequence.

In this example, the sequence generator grants user slsadmin the ability to modify the customer number sequence:

```
GRANT UPDATE 
ON SEQUENCE pub.customer_sequence 
TO slsadmin; 
```

The following syntax is a variation on the `GRANT` statement that enables the user to execute stored Java procedures:

**Syntax**

```
GRANT EXECUTE ON StoredJavaProcedureName () TO { username [, username ] , . . . | PUBLIC } 
[ WITH GRANT OPTION ] ;
```
GRANT

RESOURCE

Allows the specified users to issue CREATE statements.

DBA

Allows the specified users to create, access, modify, or delete any database object, and to grant other users any privileges.

TO username [, username], ...

Grants the specified privileges on the table or view to the specified list of users.

SELECT

Allows the specified users to read data from the table or view.

INSERT

Allows the specified users to add new rows to the table or view.

DELETE

Allows the specified users to delete rows from the table or view.

INDEX

Allows the specified users to create an index on the table or view.

UPDATE [ ( column , column , ... ) ]

Allows the specified users to modify existing rows in the table or view. If followed by a column list, the users can modify values only in the columns named.

REFERENCES [ ( column , column , ... ) ]

Allows the specified users to refer to the table from other tables’ constraint definitions. If followed by a column list, constraint definitions can refer only to the columns named.

For more detail on constraint definitions, see the Column constraints and Table constraints entries of this section.

ALL

Grants all privileges for the table or view.

TO PUBLIC

Grants the specified privileges on the table or view to any user with access to the system.

WITH GRANT OPTION

Allows the specified users to grant their privileges or a subset of their privileges to other users.
Examples  The following example illustrates the **GRANT** statement:

```
GRANT DELETE ON cust_view TO dbuser1;
GRANT SELECT ON newcustomers TO dbuser2;
```

If the **username** specified in a **RESOURCE** or **DBA** **GRANT** operation does not already exist, the **GRANT** statement creates a row in the **SYSDBAUTH** system table for the new **username**. This row is not deleted by a subsequent **REVOKE** operation.

In the following example, a DBA grants DELETE and SELECT privileges to the user **dbuser1** associated with **domuser1** of a multi-tenant table:

```
GRANT DELETE ON cust_view TO dbuser1@domuser1;
GRANT SELECT ON newcustomers TO dbuser2@domuser1;
```

The user **dbuser1**, **dbuser2** must be associated with the domain name **domuser1** in the multi-tenant table.

For more information on SQL database user authentication and **GRANT** statement, see *OpenEdge Getting Started: Identity Management*.

**Authorization**

To grant permissions for a user and to transfer the permissions from one user to others, you must have **DBA** privilege, ownership of the table, or all the specified privileges on the table (granted with the **WITH GRANT OPTION** clause). You must have **DBA** or **AUDIT_ADMIN WITH GRANT** privileges to grant auditing privileges.

**Related statement**

**REVOKE**

**INSERT**

Inserts new rows into the specified table or view that will contain either the explicitly specified values or the values returned by the query expression.

When you insert rows in a multi-tenant table, the data is inserted into the appropriate tenant partition. The **INSERT** statement is supplemented with a **TENANT** clause for SQL to understand which table partition receives the inserted row.

For a regular tenant in a multi-tenant table, the **TENANT** clause is optional. However, the tenant name must match the user’s tenancy. Otherwise, the **INSERT** statement returns an error.

For a super-tenant or a DBA in a multi-tenant table, the **TENANT** clause must be the name of an existing tenant, or a default tenant. This conveys the information to the **INSERT** statement about the point of insertion of row(s) in an existing or default tenant partition of the multi-tenant table.
When the INSERT statement is applied to the group partition for a tenant, the TENANT clause names the tenant that is part of the multi-tenant table that is being inserted with rows. Any indexes for the table are updated using the partition information for the group.

Syntax

```
INSERT INTO [ owner_name] {table_name|view_name} 
[ TENANT tenant_name ]
[( column_name [, column_name ] , ... ) ]
[( column_name [, column_name ] , ... ) ]
{ VALUES ( value [, value ] , ... ) | query_expression } ;
```

Examples

The following provides examples of the INSERT statement:

```
INSERT INTO customer (cust_no, name, street, city, state) 
VALUES 
  (1001, 'RALPH', '#10 Columbia Street', 'New York', 'NY') ;

INSERT INTO neworders (order_no, product, qty)
SELECT order_no, product, qty
FROM orders
WHERE order_date = SYSDATE ;

INSERT INTO pub.mtcustomer TENANT SNCSoftware 
(custnum, name) 
VALUES 
{9999, 'West Side Sports'};

INSERT INTO pub.mtcustomer TENANT SNCSoftware
SELECT * 
FROM pub.mtcustomer AS mtc
WHERE tenantName_tbl (mtc) = 'default'
AND MOD (custnum, 2) = 0;

INSERT INTO pub.mtcustomer TENANT OEDProducts
SELECT * 
FROM pub.mtcustomer AS mtc
WHERE tenantName_tbl (mtc) = 'default'
AND MOD (custnum, 2) <> 0;

DELETE 
FROM pub.mtcustomer AS mtc
WHERE tenantName_tbl (mtc) = 'default';
```
Notes

• If the optional list of column names is specified, then only the values for those columns are required. The rest of the columns of the inserted row will contain NULL values, provided that the table definition allows NULL values and there is no DEFAULT clause for the columns. If a DEFAULT clause is specified for a column and the column name is not present in the optional column list, then the column is given the default value.

• If the optional list is not specified, then the column values must be either explicitly specified or returned by the query expression. The order of the values should be the same as the order in which the columns are declared in the declaration of the table or view.

• The VALUES (...) form for specifying the column values inserts one row into the table. The query expression form inserts all the rows from the query results.

• A SELECT statement utilizing a NOLOCK hint can be used within an INSERT statement. For example:

```sql
INSERT INTO PUB.CUSTOMER
SELECT * FROM PUB.ARCHIVE_CUST WHERE ...
WITH (NOLOCK);
```

For more information using the NOLOCK hint in a SELECT statement, see Notes section on page 88.

• can be used if If the table contains a foreign key and there is no corresponding primary key that matches the values of the foreign key in the record being inserted, then the insert operation is rejected.

Authorization

To insert a row, you must have DBA privileges, ownership of the table, INSERT privilege on the table, or SELECT privilege on all the tables or views referred to in the query_expression, if it is specified.

A user mapped to a super-tenant can view data for all the tenants only if the user has the required permissions on the tables referenced in the statement.

Related statements

REVOKE
LOCK TABLE

Explicitly locks one or more specified tables for shared or exclusive access.

Syntax

```
LOCK TABLE table_name [, table_name ] , ... 
IN { SHARE | EXCLUSIVE } MODE ;
```

**table_name**

The table in the database that you want to lock explicitly. You can specify one table or a comma-separated list of tables.

**SHARE MODE**

Allows all transactions to read the tables. Prohibits all other transactions from modifying the tables. After you acquire an explicit lock on a table in SHARE MODE, any SELECT statements in your transaction can read rows and do not implicitly acquire individual record locks. Any INSERT, UPDATE, and DELETE statements do acquire record locks.

**EXCLUSIVE MODE**

Allows the current transaction to read and modify the tables, and prohibits any other transactions from reading or modifying the tables. After you acquire an explicit lock on a table in EXCLUSIVE MODE, you can SELECT, INSERT, UPDATE, and DELETE rows, and your transaction does not implicitly acquire individual record locks for these operations.

**Examples**

Unless another transaction holds an EXCLUSIVE lock on the teratab and megatab tables, the SHARE MODE example explicitly locks the tables. The shared lock allows all transactions to read the tables. Only the current transaction can modify the tables, as shown in the following example:

```
LOCK TABLE teratab, megatab IN SHARE MODE ;
```

Unless another transaction holds a lock on the teratab table, the EXCLUSIVE MODE example locks the teratab table for exclusive use by the current transaction. No other transactions can read or modify the teratab table, as shown in the following example:

```
LOCK TABLE teratab IN EXCLUSIVE MODE ;
```
Without a table lock, the first `SELECT` statement in the following example could exceed the limits of the record lock table, while the `LOCK TABLE` statement prevents the subsequent `SELECT` statement from consuming the record lock table:

```
-- Without a table lock, this SELECT statement creates an entry in the record lock table for every row in teratab.
SELECT COUNT (*) FROM teratab ;

-- The LOCK TABLE IN SHARE MODE operation preserves the record lock table resource.
LOCK TABLE teratab IN SHARE MODE ;
SELECT COUNT (*) FROM teratab ;
```

**Notes**

- The `LOCK TABLE` statement might encounter a locking conflict with another transaction.

- The `SHARE MODE` option detects a locking conflict if another transaction:
  - Locked the table in `EXCLUSIVE MODE` and has not issued a `COMMIT` or `ROLLBACK`
  - Inserted, updated, or deleted rows in the table and has not issued a `COMMIT` or `ROLLBACK`

- The `EXCLUSIVE MODE` option detects a locking conflict if another transaction:
  - Locked the table in `SHARE MODE` or `EXCLUSIVE MODE` and has not issued a `COMMIT` or `ROLLBACK`
  - Read from, inserted, updated, or deleted rows and has not issued a `COMMIT` or `ROLLBACK`

- When there is a locking conflict, the transaction is suspended and the database returns an error. You might configure the time at which the transaction is suspended. The default is five seconds.

- You can use explicit table locking to improve the performance of a single transaction, at the cost of decreasing the concurrency of the system and potentially blocking other transactions. It is more efficient to lock a table explicitly if you know that the transaction will be updating a substantial part of a table. You gain efficiency by decreasing the overhead of the implicit locking mechanism, and by decreasing any potential wait time for acquiring individual record locks on the table.

- You can use explicit table locking to minimize potential deadlocks in situations where a transaction is modifying a substantial part of a table. Before making a choice between explicit or implicit locking, compare the benefits of table locking with the disadvantages of losing concurrency.

- The database releases explicit and implicit locks only when the transaction ends with a `COMMIT` or `ROLLBACK` operation.
Authorization

Must have **DBA** privilege or **SELECT** privilege on the table.

Related statements

**COMMIT, ROLLBACK, SET TRANSACTION ISOLATION LEVEL**

**REVOKE**

Revoke various privileges from the specified users of the database. There are two forms of the **REVOKE** statement:

- Revoke database-wide privileges, either system administration (**DBA**), general creation (**RESOURCE**), audit administration (**AUDIT_ADMIN**), audit archive (**AUDIT_ARCHIVE**), or audit insert (**AUDIT_INSERT**)
- Revoke various privileges on specific tables and views

**Syntax**

```
REVOKE { RESOURCE, DBA, AUDIT_ADMIN, AUDIT_ARCHIVE, AUDIT_INSERT}
FROM { user_identifier [, user_identifier ] , ... }
[ RESTRICT | CASCADE ]
[ GRANTED BY ANY_USER ];
```

**RESOURCE**

Revoke the privilege to issue **CREATE** statement for specified users.

**DBA**

Revoke the privilege to create, access, modify, or delete any database object, and revokes the privilege to grant other users any privileges for specified users.

**AUDIT_ADMIN**

Revoke the privilege to administer and maintain a database auditing system for specified users.

**AUDIT_ARCHIVE**

Revoke the privilege to read and delete audit records for specific users.

**AUDIT_INSERT**

Revoke the privilege to insert application audit records for specified users.

```
FROM user_identifier [, user_identifier ] , ...
```

Revoke the specified privileges on the table or view from the specified list of users.
REVOKE

RESTRICT | CASCADE

Prompts SQL to check if the privilege being revoked was passed onto other users. If the original privilege was included the WITH GRANT OPTION clause, then the REVOKE statement fails and generates an error. If the privilege was not passed on, the REVOKE statement succeeds.

If the REVOKE statement specifies CASCADE, revoking the access privileges from a user also revokes the privileges from all users who received the privilege from that user.

If the REVOKE statement does not specify either RESTRICT or CASCADE, then the behavior follows only CASCADE.

user_identifier

Identifies a username. For a tenant user, you must mention the fully qualified user name, username@domain_name, to revoke access from a user.

Syntax

```
username | username@domain_name
```

**Note:** CASCADE is not supported for AUDIT_ADMIN, AUDIT_ARCHIVE, and AUDIT_INSERT privileges. The only user who can revoke an audit privilege is the user who granted it.

GRANTED BY ANY_USER

A DBA can use this phrase to revoke all access privileges to a table for a specified user, even if the user was the creator of the table. This phrase is only applicable to users with DBA privileges.

Example

In this example, the audit administration privilege is revoked from bsmith:

```
REVOKE AUDIT_ADMIN FROM bsmith RESTRICT;
```

The syntax to revoke privileges on specific tables and views is shown below:

Syntax

```
REVOKE [ GRANT OPTION FOR ]
{ privilege [, privilege ] , ... | ALL [ PRIVILEGES ] }
on table_name
FROM { user_identifier [, user_identifier ] , ... | PUBLIC }
[ RESTRICT | CASCADE ];
```
REVOKE

GRANT OPTION FOR

Revoke the GRANT option for the privilege from the specified users. The actual privilege itself is not revoked. If specified with RESTRICT, and the privilege is passed on to other users, the REVOKE statement fails and generates an error. Otherwise, GRANT OPTION FOR implicitly revokes any privilege the user might have provided to other users.

privilege

The syntax for the privilege item is shown below:

Syntax

```
{ SELECT | INSERT | DELETE | INDEX
  | UPDATE [ ( column , column , ... ) ]
  | REFERENCES [ ( column , column , ... ) ] };
```

privilege [ , privilege ] , ... | ALL [ PRIVILEGES ]

List of privileges to be revoked. Refer to the description in the GRANT statement. Revoking RESOURCE and DBA privileges can only be done by the administrator or a user with DBA privileges.

If more than one user grants access to the same table to a user, then all the grantors must perform a revoke for the user to lose access to the table.

Using the keyword ALL revokes all the privileges granted on the table or view.

FROM PUBLIC

Revoke the specified privileges on the table or view from any user with access to the system.

RESTRICT | CASCADE

Prompts SQL to check to see if the privilege being revoked was passed on to other users. If the original privilege included the clause WITH GRANT OPTION, then the REVOKE statement fails and generates an error. If the privilege was not passed on, the REVOKE statement succeeds.

If the REVOKE statement specifies CASCADE, revoking the access privileges from a user also revokes the privileges from all users who received the privilege from that user.

If the REVOKE statement does not specify either RESTRICT nor CASCADE, the behavior follows only CASCADE.
Example

In the following example, **REVOKE** is used on **INSERT** and **DELETE** privileges:

```
REVOKE INSERT ON customer FROM dbuser1;
REVOKE DELETE ON cust_view FROM dbuser2;
```

If the **username** specified in a **GRANT DBA** or **GRANT RESOURCE** operation does not already exist in the authorization tables, the **GRANT** statement creates a row in the **SYSDBAUTH** system table for the new **username**. This row is not deleted by a subsequent **REVOKE** operation.

In the following example, a **DBA** revokes privileges from **dbuser1** associated with **domuser1** domain of a multi-tenant table:

```
REVOKE INSERT ON customer FROM dbuser1@domuser1;
REVOKE DELETE ON cust_view FROM dbuser2@domuser1;
```

The usernames **dbuser1** and **dbuser2** must be associated with the domain name **domuser1** in the multi-tenant table.

For more information on SQL database user authentication and **REVOKE** statement, see **OpenEdge Getting Started: Identity Management**.

Authorization

To revoke privileges, you must have **DBA** privilege or ownership of the table. To revoke audit privileges, the user must have **DBA** or **AUDIT ADMINISTRATION WITH GRANT** privileges and you must be the user who granted the audit privilege.

Related statement

**GRANT**

**ROLLBACK**

Ends the current transaction and undoes any database changes performed during the transaction.

**Syntax**

```
ROLLBACK [ WORK ] ;
```

**Notes**

- Under certain circumstances, SQL marks a transaction for abort but does not actually roll it back immediately. Without an explicit **ROLLBACK**, any subsequent updates do not take effect. A **COMMIT** statement causes SQL to recognize the transaction as marked for abort and instead implicitly rolls back the transaction.

- SQL marks a transaction for abort in the event of a hardware or software system failure. This transaction is rolled back during recovery.
Authorization

None

Related statements

COMMIT

**SELECT**

Selects the specified column values from one or more rows contained in the tables or views specified in the query expression. The selection of rows is restricted by the `WHERE` clause. You can also use clauses like `GROUP BY`, `HAVING`, `ORDER BY` with the `SELECT` statement. These clauses are described in detail in the further sections. To limit the number of rows that are returned, you can implement a query paging solution using the `OFFSET` and `FETCH` clauses. The temporary table derived through the clauses of a select statement is called a result table.

When selecting rows in a multi-tenant table, a regular tenant can only view the rows in its partition, but a DBA or a super-tenant can view all the tenant partitions in the multi-tenant tables being accessed.

A super-tenant can also view a list of tenant IDs and tenant names that are associated with data from a tenant partition or from a group partition, by using the `tenantid_tbl()` and the `tenantName_tbl()` functions, respectively.

These functions logically evaluate to the set of IDs and names of the tenants that are associated with a tenant partition or with a group partition. They also enable the query to specify the tenants for which data should be selected.

**Syntax**

```sql
SELECT [ ALL | DISTINCT ] [ TOP n ]
{ *
| { table_name | alias} * [ , { table_name | alias} * ] ... |
| expr [ [ AS ] [ ' ] column_title [ ' ] ]
| [ , expr [ [ AS ] [ ' ] column_title [ ' ] ] ] ] ... |
}
FROM table_ref [ , table_ref ] ... [ { NO REORDER } ] [ WITH ( NOLOCK ) ]
[ WHERE search_condition ]
[ GROUP BY [ table]column_name [ . [ table]column_name ] ... ]
[ HAVING search_condition ];
[ ORDER BY ordering_condition ]
[ OFFSET offset_value { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } fetch_value { ROW | ROWS } ONLY ]
[ WITH locking_hints ]
[ FOR UPDATE update_condition ]
;```
SELECT column_list

See the Notes section on page 90.

TOP n search_condition

See the Notes section on page 93.

FROM table_list

See the Notes section on page 94.

WHERE search_condition

See the Notes section on page 96.

GROUP BY grouping_condition

See the Notes section on page 97.

HAVING search_condition

See the Notes section on page 99.

ORDER BY ordering_condition

See the Notes section on page 99.

OFFSET offset_value

See the Notes section on page 101.

FETCH { FIRST | NEXT }

See the Notes section on page 101.

WITH locking_hints

See the Notes section on page 103.

FOR UPDATE update_condition

See the Notes section on page 104.

Examples

The following example displays the different forms of the SELECT statement:

The example below illustrates the use of SELECT statement.

```
SELECT * FROM pub.customer;
```

Assuming the user is mapped to a DBA or a super-tenant, the example below directs the SELECT statement to join three multi-tenant tables.
When joining three multi-tenant tables, the DBA or the super-tenant must make sure that data from one tenant, `pub.mtcustomer`, is joined only with the data owned by the same tenant in the other tables, `pub.mtorder` and `pub.mtorderline`; especially when primary keys, `custnum` and `ordernum`, are unique for a tenant, but not unique across all the tenants.

```
SELECT
    tenantName_tbl (c) as ten_name,
    c.name as c_name,
    COUNT (*)
FROM
    Pub.mtcustomer AS c
INNER JOIN pub.mtorder AS o
    ON tenantId_tbl (c) = tenantId_tbl (o)
    AND c.custnum = o.custnum
INNER JOIN pub.mtorderline AS ol
    ON tenantId_tbl (o) = tenantId_tbl (ol)
    AND o.ordernum = ol.ordernum
GROUP BY
    ten_name, c_name;
```

**Authorization**

To select a specified column, you must have **DBA privileges** or **SELECT** permission on all the tables or views referred to in the **query_expression**.

A user mapped to a super-tenant can view data for all the tenants only if the user has the required permissions on the tables referenced in the statement.

**Related Statements**

**WHERE clause, GROUP BY clause, HAVING clause, ORDER BY clause, FOR UPDATE clause.**

**COLUMN_LIST clause**

Specifies which columns to retrieve by the **SELECT** statement.

**Syntax**

```
[ ALL | DISTINCT ]
{ * | { table_name | alias.}*: [ , { table_name | alias.}*: ]... |
    expr [[ AS ]| ''] column_title [ ''] ]
    [ expr [[ AS ]| ''] column_alias [ ''] ]... ]
    [ table | alias.]column_name , ... ]
```

```
[ ALL | DISTINCT ]
```

Indicates whether a result table omits duplicate rows. **ALL** is the default and specifies that the result table includes all rows. **DISTINCT** specifies that a table omits duplicate rows.
* \{table_name. | alias. \} *

Specifies that the result table includes all columns from all tables named in the FROM clause.

* expr \[\ [ AS ]\ [ ' ] column_alias [ ' ] \]

Specifies a list of expressions, called a select list, whose results will form columns of the result table. Typically, the expression is a column name from a table named in the FROM clause. The expression can also be any supported mathematical expression, scalar function, or aggregate function that returns a value.

The optional column_alias argument specifies a new heading for the associated column in the result table. You can also use the column_title in an ORDER BY clause. Enclose the new title in single or double quotation marks if it contains spaces or other special characters, including hyphens.

**Note:** A table alias cannot be used to qualify a column alias. A column alias can only be used without a qualifier because it is not a part of any table definition.

[ table | alias. ]column_name , ... ]

Specifies a list columns from a particular table or alias.

**Examples**

Both these statements return all the columns in the customer table to the select list:

```
SELECT * FROM Customer;
SELECT Customer.* FROM Customer;
```

The table_name.* syntax is useful when the select list refers to columns in multiple tables and you want to specify all the columns in one of those tables. For example:

```
SELECT Customer.CustNum, Customer.Name, Invoice.*
    FROM Customer, Invoice;
```
The following example illustrates using the `column_alias` option to change the name of the column:

```sql
-- Illustrate optional 'column_title' syntax
SELECT
    FirstName AS 'First Name',
    LastName AS 'Last Name',
    state AS 'New England State'
FROM Employee
WHERE state = 'NH' OR state = 'ME' OR state = 'MA'
OR state = 'VT' OR state = 'CT' OR state = 'RI';
```

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>New England State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Justine</td>
<td>Smith</td>
<td>MA</td>
</tr>
<tr>
<td>Andy</td>
<td>Davis</td>
<td>MA</td>
</tr>
<tr>
<td>Marcy</td>
<td>Adams</td>
<td>MA</td>
</tr>
<tr>
<td>Larry</td>
<td>Dawson</td>
<td>MA</td>
</tr>
<tr>
<td>John</td>
<td>Burton</td>
<td>NH</td>
</tr>
<tr>
<td>Mark</td>
<td>Hall</td>
<td>NH</td>
</tr>
<tr>
<td>Stacey</td>
<td>Smith</td>
<td>MA</td>
</tr>
<tr>
<td>Scott</td>
<td>Abbott</td>
<td>MA</td>
</tr>
<tr>
<td>Meredith</td>
<td>White</td>
<td>NH</td>
</tr>
<tr>
<td>Heather</td>
<td>White</td>
<td>NH</td>
</tr>
</tbody>
</table>

You must qualify a column name if it occurs in more than one table specified in the `FROM` clause, as shown:

```sql
SELECT Customer.CustNum FROM Customer;
```

```sql
-- Table name qualifier required
-- Customer table has city and state columns
-- Billto table has city and state columns
SELECT
    Customer.CustNum,
    Customer.City AS 'Customer City',
    Customer.State AS 'Customer State',
    Billto.City AS 'Bill City',
    Billto.State AS 'Bill State'
FROM Customer, Billto
WHERE Customer.City = 'Clinton';
```

<table>
<thead>
<tr>
<th>CustNum</th>
<th>Customer City</th>
<th>Customer State</th>
<th>Bill City</th>
<th>Bill State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1272</td>
<td>Clinton</td>
<td>MS</td>
<td>Montgomery</td>
<td>AL</td>
</tr>
<tr>
<td>1272</td>
<td>Clinton</td>
<td>MS</td>
<td>Atlanta</td>
<td>GA</td>
</tr>
<tr>
<td>1421</td>
<td>Clinton</td>
<td>SC</td>
<td>Montgomery</td>
<td>AL</td>
</tr>
<tr>
<td>1421</td>
<td>Clinton</td>
<td>SC</td>
<td>Atlanta</td>
<td>GA</td>
</tr>
<tr>
<td>1489</td>
<td>Clinton</td>
<td>OK</td>
<td>Montgomery</td>
<td>AL</td>
</tr>
<tr>
<td>1489</td>
<td>Clinton</td>
<td>OK</td>
<td>Atlanta</td>
<td>GA</td>
</tr>
</tbody>
</table>
When there is a conflict between a `SELECT` list alias and an actual database column, OpenEdge SQL interprets the reference as the database column. Note the following example:

```sql
SELECT substring (state, 1, 2) state, sum (balance)
FROM pub.customer
GROUP BY state;
```

In the above query, `state` is ambiguous because it can refer to either database column `pub.customer.state` or the result of the substring scalar function in the `SELECT` list. The ANSI standard requires that `state` refers unambiguously to the database column, therefore, the query groups the result by the database column. The same principle holds true for ambiguous references that appear in `WHERE`, `ON`, and `HAVING` clauses.

### TOP clause

Limits the rows returned by an OpenEdge SQL query at the statement level.

**Syntax**

```
TOP n [ [column_name [, column_name ] ....] * ]
```

- `n` Indicates the number of records per table.
- `[ column_name [, column_name ] ....] * ]` Indicates the columns within a table.

When the `TOP` clause is specified, the OpenEdge SQL server returns the maximum number of rows specified in the clause. The maximum number allowed for the `TOP` clause is 2,147,483,647.

**Example**

In the following example, the `SELECT` statement returns the names of the five customers with the highest account balance:

```sql
SELECT TOP 5 * FROM pub.customer
ORDER BY balance DESC;
```

The `TOP` clause is only allowed in a top-level `SELECT` statement. Therefore, the `TOP` clause cannot be used in the following instances:

- As part of a subquery
- When derived tables are used in the query
- Within the `CREATE TABLE`, `CREATE VIEW`, `UPDATE`, and `INSERT` statements
- In queries used with set operators such as `UNION`, `INTERSECT`, and `MINUS`
In instances when the server performs aggregation on the result set (i.e., through an aggregate function such as \texttt{SUM} or \texttt{MAX}, a \texttt{GROUP BY} clause, or the \texttt{DISTINCT} keyword) the \texttt{TOP} clause should be interpreted as being applied last. When there is no aggregation in the \texttt{SELECT} statement and the result set is also sorted, then SQL will optimize sorting in order to increase query performance.

\texttt{SELECT TOP} is the functional equivalent of the Oracle \texttt{ROWNUM} functionality. Note that \texttt{SELECT TOP} is defined simply in terms of a limit on the result set size, and the optimizer determines how to use this limit for best data access. Thus, \texttt{SELECT TOP} does not have all the "procedural rules" used to define the meaning of the Oracle \texttt{ROWNUM} phrase.

\textbf{FROM clause}

Specifies one or more table references. Each table reference resolves to one table (either a table stored in the database or a virtual table resulting from processing the table reference) whose rows the query expression uses to create the result table.

\textbf{Syntax}

\begin{verbatim}
FROM table_ref [ , table_ref ] ... [ { NO REORDER } ]
\end{verbatim}

\textit{table_ref}

There are three forms of table references:

- A direct reference to a table, view, or synonym
- A derived table specified by a query expression in the \texttt{FROM} clause
- A joined table that combines rows and columns from multiple tables

If there are multiple table references, SQL joins the tables to form an intermediate result table that is used as the basis for evaluating all other clauses in the query expression. That intermediate result table is the Cartesian product of rows in the tables in the \texttt{FROM} clause, formed by concatenating every row of every table with all other rows in all tables, as shown in the following syntax:

\textbf{Syntax}

\begin{verbatim}
table_name [ AS ] [ alias [ ( column_alias [ ... ] ) ] ]
| ( query_expression ) [ AS ] alias [ (column_alias [ ... ] ) ]
| [ [ ] joined_table [ ] ]
\end{verbatim}

FROM table_name [ AS ] [ alias [ ( column_alias [ ... ] ) ] ]

Explicitly names a table. The name can be a table name, a view name, or a synonym.
**alias**

A name used to qualify column names in other parts of the query expression. Aliases are also called correlation names.

If you specify an alias, you must use it, and not the table name, to qualify column names that refer to the table. Query expressions that join a table with itself must use aliases to distinguish between references to column names.

Similar to table aliases, the column_alias provides an alternative name to use in column references elsewhere in the query expression. If you specify column aliases, you must specify them for all the columns in table_name. Also, if you specify column aliases in the FROM clause, you must use them, and not the column names, in references to the columns.

```sql
FROM ( query_expression ) [ AS ] [ alias [ ( column_alias [ ... ] ) ] ]
```

Specifies a derived table through a query expression. With derived tables, you must specify an alias to identify the derived table.

Derived tables can also specify column aliases. Column aliases provide alternative names to use in column references elsewhere in the query expression. If you specify column aliases, you must specify them for all the columns in the result table of the query expression. Also, if you specify column aliases in the FROM clause, you must use them, and not the column names, in references to the columns.

```sql
FROM [ ( ] joined_table [ ) ]
```

Combines data from two table references by specifying a join condition, as shown in the following syntax:

**Syntax**

```sql
{ table_ref CROSS JOIN table_ref
  | table_ref [ INNER | LEFT [ OUTER ] ] JOIN
  table_ref ON search_condition
}
```

The syntax currently allowed in the FROM clause supports only a subset of possible join conditions:

- **CROSS JOIN** specifies a Cartesian product of rows in the two tables. Every row in one table is joined to every row in the other table.
- **INNER JOIN** specifies an inner join using the supplied search condition.
- **LEFT OUTER JOIN** specifies a left outer join using the supplied search condition.
- **LEFT JOIN** specifies the same conditions as an inner join.

You can also specify these and other join conditions in the WHERE clause of a query expression.
Disables join order optimization for the \texttt{FROM} clause. Use \texttt{NO REORDER} when you choose to override the join order chosen by the optimizer. The braces are part of the syntax for this optional clause.

\[
\textbf{[WITH} \texttt{(NOLOCK)} \textbf{]}\]

Allows a dirty read to occur in the event records are locked by another user.

**Example**

For customers with orders, retrieve their names and order info, as shown in the following example:

\[
\begin{verbatim}
SELECT Customer.CustNum,Customer.Name, Order.OrderNum,Order.OrderDate
FROM Customer, Order
WHERE Customer.CustNum = Order.CustNum;
\end{verbatim}
\]

**WHERE clause**

Specifies a search condition that applies conditions to restrict the number of rows in the result table. If the query expression does not specify a \texttt{WHERE} clause, the result table includes all the rows of the specified table reference in the \texttt{FROM} clause.

**Syntax**

\[
\texttt{WHERE search\_condition}
\]

\[
\textit{search\_condition}
\]

Applied to each row of the result table set of the \texttt{FROM} clause. Only rows that satisfy the conditions become part of the result table. If the result of the \textit{search\_condition} is \texttt{NULL} for a row, the row is not selected. Search conditions can specify different conditions for joining two or more tables.

**Example**

For customers with orders, retrieve their names and order info:

\[
\begin{verbatim}
SELECT Name, City, State
FROM Customer
WHERE State = 'NM';
\end{verbatim}
\]
GROUP BY clause

Specifies grouping of rows in the result table. The results may be grouped by column, alias or expression.

Grouping by column

The result set of a query may be ordered by one or more columns specified in the GROUP BY clause.

Syntax

GROUP BY [ table_name.]column_name ...  

Notes

- For the first column specified in the GROUP BY clause, SQL arranges rows of the result table into groups whose rows all have the same values for the specified column.
- If you specify a second GROUP BY column, SQL groups rows in each main group by values of the second column.
- SQL groups rows for values in additional GROUP BY columns in a similar fashion.
- All columns named in the GROUP BY clause must also be in the select list of the query expression. Conversely, columns in the select list must also be in the GROUP BY clause or be part of an aggregate function.

Example

This example retrieves name and order info for customers with orders:

```
SELECT DeptCode, LastName
FROM Employee
GROUP BY DeptCode;
```

Grouping by alias

The GROUP BY clause orders the result set according to an alias specified in the SELECT statement.

Syntax

GROUP BY [ alias ] ...

Note

In this instance, the alias may be used as a simple column reference to a database table, or an actual expression composed of arithmetic expressions, character operators, date operators, or scalar functions. The alias is essentially an alternate name.
Example In the following example, the GROUP BY clause refers to the “CityState” phrase of the SELECT statement:

```
SELECT CONCAT (State, City) AS "CityState",
       COUNT (city)
FROM Pub.Customer
GROUP BY "CityState";
```

Grouping by expression

The GROUP BY clause orders the result set according to an expression used in the SELECT statement.

Syntax

```
GROUP BY [ expression ] ...
```

Note

The GROUP BY clause can contain any scalar expression which produces a value that is used as a grouping key. An individual column, when it is part of a larger expression in a GROUP BY list, cannot by itself be referenced in the SELECT list. Only the entire expression, which is the grouping key, can be used in the statement’s SELECT list. Note that a GROUP BY expression cannot contain an aggregate expression such as SUM.

The GROUP BY clause does not support set differencing operations such as MINUS and INTERSECT.

Example In the following example, the GROUP BY clause refers to the concatenation expression used in the SELECT statement:

```
SELECT CONCAT (State, City),
       COUNT (city)
FROM Pub.Customer
GROUP BY CONCAT (State, City);
```
HAVING clause

Allows you to set conditions on the groups returned by the GROUP BY clause. If the HAVING clause is used without the GROUP BY clause, the implicit group against which the search condition is evaluated is all the rows returned by the WHERE clause.

Syntax

```
HAVING search_condition
```

Note

A condition of the HAVING clause can compare one aggregate function value with another aggregate function value or a constant.

Example

The HAVING clause in the following example compares the value of an aggregate function (COUNT(*)) to a constant (10):

```
SELECT CustNum, COUNT(*)
FROM Order
WHERE OrderDate < TO_DATE ('3/31/2004')
GROUP BY CustNum
HAVING COUNT(*) > 10;
```

The query returns the customer number and number of orders for all customers who had more than 10 orders before March 31.

ORDER BY clause

Allows ordering of the rows selected by the SELECT statement. Unless an ORDER BY clause is specified, the rows of the result set might be returned in an unpredictable order as determined by the access paths chosen and other decisions made by the query optimizer. The decisions made will be affected by the statistics generated from table and index data examined by the UPDATE STATISTICS command.

Syntax

```
ORDER BY { expr | posn } [ ASC | DESC ]
[ , { expr | posn } [ ASC | DESC ] , ... ]
```

**expr**

Expression of one or more columns of the tables specified in the FROM clause of the SELECT statement.

**posn**

Integer column position of the columns selected by the SELECT statement.

**ASC | DESC**

Indicates whether to order by ascending order (ASC) or descending order. The default is ASC.
The following example demonstrates the `ORDER BY` clause in the `SELECT` statement:

### Examples

```sql
-- Produce a list of customers sorted by name.
SELECT Name, Address, City, State, PostalCode
FROM Customer
ORDER BY Name ;
```

The following examples demonstrate the `ORDER BY` clause in query expressions containing set operators:

```sql
-- Produce a merged list of customers and suppliers with the column position specifying the sort key.
SELECT Name, Address, State, PostalCode
FROM Customer
UNION
SELECT Name, Address, State, PostalCode
FROM Supplier
ORDER BY 1;

-- Produce a merged list of customers and suppliers sorted by name.
SELECT Name, Address, State, PostalCode
FROM Customer
UNION
SELECT Name, Address, State, PostalCode
FROM Supplier
ORDER BY Name;

-- Produce a merged list of customers and suppliers sorted by name.
SELECT cust_name as Name, cust_address, cust_state, cust_postalcode
FROM Customer
UNION
SELECT sup_name, sup_address, sup_state, sup_postalcode
FROM Supplier
ORDER BY Name;
```

### Notes

- The `ORDER BY` clause, if specified, should follow all other clauses of the `SELECT` statement.

- The selected rows are ordered on the basis of the first `expr` or `posn`. If the values are the same, then the second `expr` or `posn` is used in the ordering.

- A query expression can be followed by an optional `ORDER BY` clause. If the query expression contains set operators (such as, `UNION`), then the `ORDER BY` clause can specify column names, aliases, and positions in the following ways:
  - An alias from the first `SELECT` statement of the set operator query expression can be used as the `ORDER BY` key. In this case, the alias indicates the ordering based on the corresponding column from all the query expressions (`SELECT` statements) in the set operator.
  - A column name from the first `SELECT` statement of the set operator query expression can be explicitly used as an `ORDER BY` key. In this case, the column name indicates the ordering based on the corresponding column from all the query expressions (`SELECT` statements) in the set operator.
  - An ordinal position number can be specified as an `ORDER BY` key.
A query expression is considered invalid if the ORDER BY clause:

- Has an expression
- Refers to a column name or alias that is not from the first SELECT statement
- Has a position number that is greater than the number of columns projected

**OFFSET and FETCH clauses**

The OFFSET clause specifies the number of rows to skip, before starting to return rows from the query expression. The FETCH clause specifies the number of rows to return, after processing the OFFSET clause.

**Syntax**

```
[ORDER BY { expr | posn } [ ASC | DESC ]
[ , { expr | posn } [ ASC | DESC ] , ... ]]
[OFFSET offset_value { ROW | ROWS }]
[FETCH {FIRST | NEXT} fetch_value { ROW | ROWS | ONLY }]
```

**Note:** You can use ROW and ROWS interchangeably. Similarly, FIRST and NEXT can be used interchangeably.

**expr**

Expression of one or more columns of the tables specified in the FROM clause of the SELECT statement.

**posn**

Integer column position of the columns selected by the SELECT statement.

**ASC | DESC**

Indicates whether to order by ascending order (ASC) or descending order. The default is ASC.

**offset_value**

Specifies the number of rows to skip, before starting to return the result rows of the query. This value must be a constant: literal or parameter. It does not support expressions or self-contained subqueries. The value must not be negative and the maximum number allowed is 9,223,372,036,854,775,807.

**fetch_value**

Specifies the number of rows to return, after processing the OFFSET clause. This value must be a constant: literal or parameter. It does not support expressions or self-contained subqueries. The value must not be negative and the maximum number allowed is 9,223,372,036,854,775,807.
Examples

The following example demonstrates the OFFSET clause in the SELECT statement:

```sql
SELECT OrderID,OrderDate,custID,filler
FROM dbo.Orders OFFSET 10;
```

The above query skips the first 10 rows and returns the rest of the qualified rows.

The following example demonstrates the FETCH clause in the SELECT statement:

```sql
SELECT OrderID,OrderDate,custID,filler
FROM dbo.Orders
ORDER BY OrderDate DESC, OrderID DESC
FETCH FIRST 10 ROWS ONLY;
```

The above query returns the first 10 rows without skipping any.

The following example demonstrates both the OFFSET and FETCH clauses in the SELECT statement:

```sql
SELECT OrderID,OrderDate,custID,filler
FROM dbo.Orders
ORDER BY OrderDate DESC, OrderID DESC
OFFSET 50 ROWS FETCH NEXT 10 ROWS ONLY;
```

The above query returns rows 51 through 60 in the result set of the query.

Notes

- Both the clauses are optional. However, if present, the OFFSET clause must come before the FETCH clause.
- If the OFFSET clause is specified without the FETCH clause, the SQL engine skips the specified number of rows and returns all other qualifying rows without an upper limit.
- If the FETCH clause is specified without the OFFSET clause, the SQL engine does not skip any rows and the initial value of the OFFSET is 0. This is similar to the TOP clause.
- The OFFSET and FETCH clauses need not be preceded by the ORDER BY clause. It is a good idea to use an ORDER BY clause that constrains the result rows into a unique order. If the ORDER BY clause is specified, the OFFSET and FETCH clauses must appear right after the ORDER BY clause of a query.
- The TOP clause cannot be combined with the OFFSET and FETCH clauses in the same query expression (in the same query scope).
- The TOP and OFFSET/FETCH clauses are mutually exclusive.
The `OFFSET` and `FETCH` clauses are only allowed in a top-level `SELECT` statement. Therefore, these clauses cannot be used in the following instances:

- As part of a subquery
- Within the derived tables
- Within the `CREATE TABLE`, `CREATE VIEW`, `UPDATE`, `DELETE`, and `INSERT` statements.
- In queries used with set operators such as `UNION`, `INTERSECT`, and `MINUS`
- In views

The `OFFSET` and `FETCH` clauses are interpreted as being applied last, after all the other clauses specified in the query are applied.

**WITH clause**

Enables table-level locking when a finer control of the types of locks acquired on an object is required. These locking hints override the current transaction isolation level for the session.

The locking hint clause, such as for `READPAST`, can only be specified in the main `SELECT` statement, but not in the subquery `SELECT` statement in the "search condition" of the `WHERE` clause.

**Syntax**

```
[ WITH ( READPAST NOLOCK [ WAIT timeout | NOWAIT ] ) ]
```

**search_condition**

The `READPAST` locking hint skips locked rows. This option causes a transaction to skip rows locked by other transactions that would ordinarily appear in the result set, rather than block the transaction waiting for the other transactions to release their locks on these rows. The `READPAST` lock hint applies only to transactions operating at `READ COMMITTED` isolation and will read only past row-level locks. Applies only to the `SELECT` statement.

The `NOLOCK` locking hint ensures records are not locked during the execution of a `SELECT` statement when the transaction isolation level is set to `READ COMMITTED`. When `NOLOCK` is invoked, a dirty read is possible. This locking hint only works with the `SELECT` statement.

**WAIT timeout**

Override the default lock-wait time out. The timeout value is in seconds and can be 0 or any positive number.
NOWAIT

Causes the SELECT statement to skip (read past) the row immediately if a lock cannot be acquired on a row in the selection set because of the lock held by some other transaction. The default behavior is for the transaction to wait until it obtains the required lock or until it times out waiting for the lock.

The following example demonstrates the WITH clause in the SELECT statement:

Example

```
SELECT * FROM Customer WHERE "CustNum" < 100 ORDER BY "CustNum" FOR UPDATE
WITH (READPAST WAIT 1);
```

FOR UPDATE clause

Specifies update intention on the rows selected by the SELECT statement.

Syntax

```
FOR UPDATE [ OF [ table.]column_name , ... ] [ NOWAIT ]
```

OF [ table.]column_name , ...

Specifies the table’s column name to be updated.

NOWAIT

Causes the SELECT statement to return immediately with an error if a lock cannot be acquired on a row in the selection set because of the lock held by some other transaction. The default behavior is for the transaction to wait until it obtains the required lock or until it times out waiting for the lock.

Note

If you specify FOR UPDATE, the database acquires exclusive locks on all the rows satisfying the SELECT statement. The database does not acquire row level locks if there is an exclusive lock on the table. See the LOCK TABLE statement for information on table locking.

SET CATALOG

Changes the default catalog name to be used for schema, table, and column references. The default catalog name is initially the name of the primary database.

Syntax

```
SET CATALOG catalog_name;
```

catalog_name

Catalog name to be used as an alias for the database in schema, table and column references. This must be in the form of an SQL identifier of up to 32 bytes in length.
Example

In this example, the auxiliary database connection identified by the catalog named `mydb1` is specified as the default catalog:

```
SET CATALOG mydb1;
```

Notes

- The `SET CATALOG` statement is used to specify the default database catalog name to be used for schema, table, and column references.

- The primary database connection is automatically given a catalog name which is the name of the primary database. For example, if the primary database is at `/usr/progress/sports2000`, then the catalog name for the primary database is `sports2000`.

- The `SET CATALOG` statement may be used to set the default catalog to an auxiliary database catalog or to the primary database catalog.

- The specified catalog must identify a current catalog name.

- If an auxiliary database catalog is set as the default catalog, disconnecting from the auxiliary catalog will not change the name of the default catalog. Thus setting the default catalog to an auxiliary database may cause failures of statements when the default catalog is not set to an active catalog. In other words, shutting down an auxiliary database identified as the default catalog will cause any query using a three-part specification to fail. The failure produces an error indicating that the catalog is not connected.

Authorization

Any user is allowed to execute this statement.

SQL Compliance

Progress Software Corporation specific extension.

Related statements

CONNECT AS CATALOG, DISCONNECT CATALOG

---

**SET PRO_CONNECT LOG**

Controls logging for the current SQL Server connection.

Syntax

```
SET PRO_CONNECT LOG [ ON | OFF ] [ WITH ( { STATEMENT, QUERY_PLAN } ) ];
```

- **ON**
  
  Indicates that logging is turned on.
SET PRO_CONNECT QUERY_TIMEOUT

OFF

Indicates that logging is turned off.

STATEMENT

Indicates that statement tracing information is written to the log file.

QUERY_PLAN

Indicates that query plan information is written to the log file.

Notes

• When logging is set ON, the current SQL connection begins logging to a file named as SQL_server_<server-id>_<ddmmyyyy>_<hhmmss>.log.

  For example: SQL_server_1_05MAY2005_112609.log

• The server-id corresponds to the server ID shown in database_name.lg.

• Logging files are located in the server's work directory. The work directory corresponds to the value of the \WRLDIR environment variable on UNIX systems and the applicable registry settings in Windows systems.

• The maximum size of each logging file is 500 MB. When SQL_server_<server-id>_<ddmmyyyy>_<hhmmss>.log reaches 500 MB, the server logs a message indicating the file was closed due to reaching the maximum size. After this message is written, all logging stops, logging is set to the OFF state, and SQL_server_<server-id>_<ddmmyyyy>_<hhmmss>.log automatically closes.

• When logging commences to a new file, the file contents begin with information about the SQL Server environment, including:
  – Environment variable settings
  – Parameter values passed to the server at startup
  – Logging control values (such as size limits)
  – The SQL Server process ID

  • Each section of information written to the log file begins with the string

    DDMMYYYY HH:MM:SS <user-id>

  For example, 19AUG2005 12:00:00 1:
Indicates the maximum number of seconds during which a query should execute before it is automatically cancelled by the SQL server.

**Notes**

- The number of seconds specified is the maximum time allowed for the execution of the following protocol messages:
  - Query statement prepare
  - Query statement execution
  - Query fetch
- The value specified by \( n \) applies to all subsequent protocol messages of these types until the timeout value is cleared. This may be accomplished simply by specifying a value of 0 on subsequent execution of the statement.

**Example**

This example sets the query timeout to 30 seconds:

```java
Statement stmt = connection.createStatement();

String MySetQueryTimeout;
String MyClearQueryTimeout;

MySetTimeout = "SET PRO_CONNECT QUERY_TIMEOUT 30";
MyClearTimeout = "SET PRO_CONNECT QUERY_TIMEOUT 0";

// Set SQL Server timeout for query execute and fetch
stmt.executeUpdate( MySetQueryTimeout );

// Add code here to perform queries

// Set SQL Server timeout for query execute and fetch
stmt.executeUpdate( MyClearQueryTimeout );
```

**SET PRO_SERVER QUERY_TIMEOUT**

Defines the maximum number of seconds during which a query should execute for the current SQL Server connection.

**Syntax**

```sql
SET PRO_SERVER QUERY_TIMEOUT \( n \);
```

\( n \)

Indicates the maximum number of seconds during which a query should execute before it is automatically cancelled by the SQL server. Setting an \( n \) value of 0 disables a previously set query timeout.
Notes

- Execution of this command is restricted to DBAs. Any value set with this command is in effect for the duration that the database is up and running.

- Should a query timeout value be set for an individual connection via the command `SET PRO_CONNECT QUERY_TIMEOUT` the lower of the timeout values for the connection and the server takes precedence.

This example sets the query timeout to 30 seconds:

```java
Statement stmt = connection.createStatement();
String MySetQueryTimeout;
String MyClearQueryTimeout;
MySetTimeout = "SET PRO_SERVER QUERY_TIMEOUT 30";
MyClearTimeout = "SET PRO_SERVER QUERY_TIMEOUT 0";
// Set SQL Server timeout for query execute and fetch
stmt.executeUpdate(MySetQueryTimeout);
// Add code here to perform queries
// Clear SQL Server timeout for query execute and fetch
stmt.executeUpdate(MyClearQueryTimeout);
```

**SET PRO_SERVER LOG**

Controls logging for all connections to all OpenEdge SQL Servers.

**Syntax**

```
SET PRO_SERVER LOG [ ON | OFF ] [ WITH ( { STATEMENT, QUERY_PLAN } ) ];
```

**ON**

Indicates that logging is turned on.

**OFF**

Indicates that logging is turned off.

**STATEMENT**

Indicates that statement tracing information is written to each log file.

**QUERY_PLAN**

Indicates that query plan information is written to the log file.

**Notes**

- When logging is set ON, each SQL Server begins logging to a file named as `SQL_server_<server-id>_<ddmmmyyyy>_<hhmmss>.log`

  **For example**: `SQL_server_1_05MAY2005_112609.log`

- The `server-id` corresponds to the server ID shown in `database_name.lg`. 

OpenEdge® Data Management: SQL Reference
• Logging files are located in the server’s work directory. The work directory corresponds to the value of the `WRKDIR` environment variable on UNIX systems and the applicable registry settings in Windows systems.

• The maximum size of each logging file is 500 MB. When `SQL_server_<server-id>_<ddmmmyyyy>_<hhmmss>.log` reaches 500 MB, the server logs a message indicating the file was closed due to reaching the maximum size. After this message is written, all logging stops, logging is set to the `OFF` state, and `SQL_server_<server-id>_<ddmmmyyyy>_<hhmmss>.log` automatically closes.

• When logging commences to a new file, the file contents begin with information about the SQL Server environment, including:
  – Environment variable settings
  – Parameter values passed to the server at startup
  – Logging control values (such as size limits)
  – The SQL Server process ID.

• Each section of information written to the log file begins with the string `DDMMYYYY HH:MM:SS <user-id>`:

For example, 19AUG2005 12:00:00 1:

### SET ROWCOUNT

Limits the result set to a specified number of rows. This functionality of restricting the size of the result set is similar to that of the `TOP` clause. However, the `TOP` clause only applies to queries at the statement level, while `SET ROWCOUNT` is a session-wide parameter and is not statement-specific.

#### Syntax

```
SET ROWCOUNT rowcount_number;
```

- `rowcount_number` 
  Specifies the number of rows to be restricted from the result set. This value must be a constant literal.

#### Examples

The following examples demonstrate the `SET ROWCOUNT` statement:

```
SET ROWCOUNT 10;
SELECT * FROM Customer;
```

The above query returns the first ten rows in the result set.
The following example demonstrates the `SET ROWCOUNT` statement with `OFFSET` and `FETCH` clauses in the `SELECT` statement:

```
SET ROWCOUNT 5;
SELECT * FROM Customer
ORDER BY Name
OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

The above query further restricts the result set, after applying the `OFFSET` and `FETCH` clauses, and returns rows 11 through 15.

The following example demonstrates the `SET ROWCOUNT` statement, where the row count value is set to 3:

```
SET ROWCOUNT 3;
SELECT TOP 5 * FROM Customer;
```

The above query returns only the first 3 rows in the result set.

Consider running the same query with the row count value set to 0 (the default value for `SET ROWCOUNT`), as shown below:

```
SET ROWCOUNT 0;
SELECT TOP 5 * FROM Customer;
```

The above query returns the first 5 rows in the result set without restricting any rows.

The following example demonstrates the `SET ROWCOUNT` statement in a stored procedure containing more than one row:

```
SET ROWCOUNT 1;
CALL get_sal ();
```

The above query returns only the first row in the result set.

**Notes**

- The default value of the row count is 0, which means that the number of rows in the result set is not restricted, as shown in the example above. When the row count value is greater than 0, the maximum size of the result set is that of the row count value.

- You can change the row count value at any point of time in a session. The new value affects the subsequent `SELECT` statements and stored procedures in the session.

- `SET ROWCOUNT` restricts the result set after all the statement level clauses (such as, `OFFSET`, `FETCH`, and `TOP` clauses) have been applied. This implies that after the restriction of `OFFSET`, `FETCH`, and `TOP` clauses on the result set, `SET ROWCOUNT` further restricts the result set.
The `SET ROWCOUNT` statement has no effect on the following:

- DDL statements such as `CREATE`, `ALTER`, and `DROP`
- DML statements such as `INSERT`, `DELETE`, and `UPDATE`
- `SHOW` statements
- Triggers

The `SET ROWCOUNT` statement has no effect on the internal `SELECT` statements used in stored procedures. However, the overall result set of the stored procedures is restricted by `SET ROWCOUNT`, as shown in the example above.

The `SET ROWCOUNT` statement affects the overall query result and not the intermediate stages of a query execution as the following:

- Intermediate result set in a subquery
- Intermediate result set in views or derived tables
- Intermediate queries used as set operators such as `UNION`, `INTERSECT`, and `MINUS`

Related Statement

`SHOW ROWCOUNT`

---

**SET SCHEMA**

Sets the default owner, also known as schema, for unqualified table references.

**Syntax**

```
SET SCHEMA { 'string_literal' | ? | USER }
```

`'string_literal'`

Specifies the name for the default owner as a string literal, enclosed in single or double quotes.

`?`

Indicates a parameter marker to contain the default owner. The actual replacement value for the owner name is supplied in a subsequent SQL operation.

`USER`

Directs the database to set the default owner back to the `username` that established the session.
Example

This example sets the default schema name to White:

```sql
SET SCHEMA 'White';
COMMIT;
SELECT * from customer;
```

Subsequent SQL statements with unqualified table references will use the owner name White. The SELECT statement in this example returns all rows in the ‘White.customer’ table. The username establishing the original session is still the current user.

Notes

- For authorization purposes, invoking `SET SCHEMA` does not change the `username` associated with the current session.
- You can set the default schema name to the `username` associated with the session by using a `SET SCHEMA USER` statement.

Authorization

None

SET TRANSACTION ISOLATION LEVEL

Explicitly sets the isolation level for a transaction. Isolation levels specify the degree to which one transaction can modify data or database objects in use by another concurrent transaction.

Syntax

```
isolation_level_name
```

The following is the syntax for `isolation_level_name`:

Syntax

```
READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE
```

READ UNCOMMITTED

Also known as a dirty read. When this isolation level is used, a transaction can read uncommitted data that later might be rolled back. The standard requires that a transaction that uses this isolation level can only fetch data but cannot update, delete, or insert data.

READ COMMITTED

Dirty reads are not possible with this isolation level. However, if the same row is repeatedly read during the same transaction, its contents can be changed or the entire row can be deleted by other transactions.
REPEATABLE READ

This isolation level guarantees that a transaction can read the same row many times and it will remain intact. However, if a query with the same search criteria (the same \texttt{WHERE} clause) is executed more than once, each execution can return different sets of rows. This can happen because other transactions are allowed to insert new rows that satisfy the search criteria or update some rows in such a way that they now satisfy the search criteria.

SERIALIZABLE

This isolation level guarantees that none of the above happens. Transactions that use this level will be completely isolated from other transactions.

Notes

- See the \texttt{LOCK TABLE} for information on record locking schemes used by each isolation level.
- For more information on transactions, see \textit{OpenEdge Data Management: SQL Development}.

Authorization

None

Related statements

\texttt{COMMIT, LOCK TABLE, ROLLBACK}

SHOW CATALOGS

Returns a list of available catalog information with catalog name, catalog type (primary or auxiliary), and catalog status (default or not default).

Syntax

\begin{verbatim}
SHOW CATALOGS \{ ALL | \{ PRO_NAME | PRO_TYPE | PRO_STATUS \} \}
\end{verbatim}

\texttt{ALL}

Return all attributes.

\texttt{PRO_NAME}

List of catalog names.

\texttt{PRO_TYPE}

List of catalog types (primary or auxiliary).

\texttt{PRO_STATUS}

List of catalog statuses (default or notdefault)
SHOW GROUP

Example

In this example, the names of the currently available catalogs are returned.

```
SHOW CATALOGS PRO_NAME;
```

Notes

- This statement is useful for obtaining the catalog names of databases currently connected for the user, for identifying the catalog name of the primary database (automatically connected) and the current default catalog.

- The primary database connection is automatically given a catalog name which is the name of the primary database. For example, if the database is at `/usr/progress/sports2000`, then the catalog name for the primary database is `sports2000`.

- Three columns of information are returned by the `SHOW CATALOG` statement. These are the catalog name, catalog type (primary or auxiliary) and status (default or notdefault).

Authorization

Any user is allowed to execute this statement.

SQL Compliance

Progress Software Corporation specific extension.

Related statements

CONNECT AS CATALOG, DISCONNECT CATALOG, SET CATALOG

SHOW GROUP

Returns a result set with information about the database tables and the existing tenants that are associated with one or more groups.

Syntax

```
SHOW { ALL GROUP [ { FOR TABLE table_name | FOR TENANT tenant_name } ] | GROUP FOR group_name } [ FULL | PRO_NAME ];
SHOW ALL GROUP
```

SHOW ALL GROUP

Returns information about all the groups defined in the multi-tenant database.

FOR TABLE table_name

Returns information about all the groups defined for the specified table. Returns zero rows if the specified table is not a multi-tenant table.
SHOW GROUP

FOR TENANT tenant_name

Returns information about all the groups that the tenant belongs to.

SHOW GROUP FOR group_name

Returns information about the specified group.

FULL

Returns all the information about all the groups defined for the specified table.

PRO_NAME

Returns only the names of the groups defined for the specified table.

Examples

The following example shows how to return the basic information about all the groups defined in the database:

```
SHOW ALL GROUP;
```

The following example shows how to return all the group names defined for a table:

```
SHOW ALL GROUP FOR TABLE pub.mtcustomer PRO_NAME;
```

The following example shows how to get full information about all the groups defined for a table:

```
SHOW ALL GROUP FOR TABLE pub.mtcustomer FULL;
```

The following example shows how to get information about all the groups that a tenant belongs to:

```
SHOW ALL GROUP FOR TENANT Consolidated_Freightways;
```

Authorization

Execution of the SHOW GROUP statement requires DBA privileges.

Related Statement

CREATE GROUP
SHOW ROWCOUNT

Displays the current value of the row count.

Syntax

```sql
SET ROWCOUNT rowcount_number;
SHOW ROWCOUNT;
```

Example

The following example demonstrates the `SHOW ROWCOUNT` statement:

```sql
SET ROWCOUNT 10;
SHOW ROWCOUNT;
```

The above query returns the row count value as 10.

Related Statement

`SET ROWCOUNT`

SHOW TENANT

Returns specific information for a tenant. The information includes type and status of the tenant. If the tenant type is default, the `SHOW TENANT` statement displays the tenant ID as zero.

Syntax

```
SHOW [ ALL TENANT | TENANT FOR tenant_name ] [ FOR TABLE table_name ]
[ FULL | PRO_NAME | TABLE | TABLE_INDEX ]
;  
```

`SHOW ALL TENANT`

Returns information about the database objects for the defined tenants.

`SHOW TENANT FOR tenant_name`

Returns information about the database objects for the specified tenant.

`FOR TABLE table_name`

Returns information about the tenant in the specified table. If the specified table is not a multi-tenant table, it does not return any rows in the result set.

`FULL`

Returns complete information about the database objects for the specified tenant.

`PRO_NAME`

Returns the names of tenants in the multi-tenant table.
TABLE

Returns information about the table associated with the specified tenant.

TABLE INDEX

Returns information about the table and index defined for the multi-tenant tables associated with a specified tenant.

Example

The example shows how to return the available tenants of a table mtcustomer:

```sql
SHOW ALL TENANT FOR TABLE pub.mtcustomer;
```

Authorization

All users are allowed to execute the SHOW TENANT statement.

Related Statement

DROP TENANT

SHOW ENCRYPT ON

The `SHOW ENCRYPT` statement provides encryption policy information on the primary database. It can be used only by security administrators or DBAs.

Syntax

```sql
SHOW ENCRYPT ON { ALL | [ TABLE | INDEX | LOB ]
   | TABLE table_name [ WITH INDEX| WITH LOB]
   | TABLE table_name ON INDEX index_name }
```

When run, the statement returns a result set with eight columns:

- Database object type (AREA, TABLE, INDEX, LOB)
- Database object name
- Object’s table name (blank for area)
- Database object name (blank for area)
- Database object identification
- Object policy state (CURRENT or PREVIOUS)
- Object policy cipher name
- Object policy version number

Notes

- Only active policies are returned by the statement.
• The only option which shows Type I area encryption information is the SQL statement `SHOW ENCRYPTION ON ALL`. Other options on `SHOW ENCRYPT` show encryption information only for Type II area database objects.

**Authorization**

Security Administrator or DBA.

**SQL Compliance**

Progress Software Corporation specific extension.

**Related statements**

`ALTER TABLE, CREATE INDEX, CREATE TABLE`
Table constraints

Table constraints specifies a constraint for a table that restricts the values that the table can store. INSERT, UPDATE, or DELETE statements that violate the constraint fail. SQL returns a constraint violation error.

Table constraints have syntax and behavior similar to column constraints. Note the following differences:

- The definitions of the table constraints are separated from the column definitions by commas.
- Table constraint definitions can include more than one column, and SQL evaluates the constraint based on the combination of values stored in all the columns.

Syntax

```sql
CONSTRAINT constraint_name
   PRIMARY KEY ( column [ , ... ] )
| UNIQUE ( column [ , ... ] )
| FOREIGN KEY ( column [ , ... ] )
   REFERENCES [ owner_name. ]table_name [ ( column [ , ... ] ) ]
| CHECK ( search_condition )
```

CONSTRAINT constraint_name

Allows you to assign a name that you choose to the table constraint. While this specification is optional, this facilitates making changes to the table definition, since the name you specify is in your source CREATE TABLE statement. If you do not specify a constraint_name, the database assigns a name. These names can be long and unwieldy, and you must query system tables to determine the name.

PRIMARY KEY ( column [ , ... ] )

Defines the column list as the primary key for the table. There can be at most one primary key for a table.

All the columns that make up a table level primary key must be defined as NOT NULL, or the CREATE TABLE statement fails. The combination of values in the columns that make up the primary key must be unique for each row in the table.

Other tables can name primary keys in their REFERENCES clauses. If they do, SQL restricts operations on the table containing the primary key in the following ways:

- DROP TABLE statements that delete the table fail
- DELETE and UPDATE statements that modify values in the combination of columns that match a foreign key’s value also fail
Table constraints

**UNIQUE ( column [ , ... ] )**

Defines the column list as a unique, or candidate, key for the table. Unique key table-level constraints have the same rules as primary key table-level constraints, except that you can specify more than one **UNIQUE** table-level constraint in a table definition.

**FOREIGN KEY ( column [, ... ] ) REFERENCES [ owner_name ]:table_name [ ( column [, ... ] ) ]**

Defines the first column list as a foreign key and, in the **REFERENCES** clause, specifies a matching primary or unique key in another table.

A foreign key and its matching primary or unique key specify a referential constraint. The combination of values stored in the columns that make up a foreign key must either:

- Have at least one of the column values be null.
- Be equal to some corresponding combination of values in the matching unique or primary key.

You can omit the column list in the **REFERENCES** clause if the table specified in the **REFERENCES** clause has a primary key and you want the primary key to be the matching key for the constraint.

**CHECK (search_condition)**

Specifies a table level check constraint. The syntax for table level and column level check constraints is identical. Table level check constraints must be separated by commas from surrounding column definitions.

SQL restricts the form of the search condition. The search condition must not:

- Refer to any column other than columns that precede it in the table definition
- Contain aggregate functions, subqueries, or parameter references

**Examples**

In the following example, which shows creation of a table level primary key, note that its definition is separated from the column definitions by a comma:

```sql
CREATE TABLE SupplierItem (  SuppNum INTEGER NOT NULL,  ItemNum INTEGER NOT NULL,  Quantity INTEGER NOT NULL DEFAULT 0,  PRIMARY KEY (SuppNum, ItemNum)) ;
```
The following example shows how to create a table with two `UNIQUE` table level constraints:

```
CREATE TABLE OrderItem {
    OrderNum    INTEGER NOT NULL,
    ItemNum     INTEGER NOT NULL,
    Quantity         INTEGER NOT NULL,
    Price       INTEGER NOT NULL,
    UNIQUE (OrderNum, ItemNum),
    UNIQUE (Quantity, Price));
```

The following example defines the combination of columns `student_courses.teacher` and `student_courses.course_title` as a foreign key that references the primary key of the `courses` table:

```
CREATE TABLE Courses {
    Instructor        CHAR (20) NOT NULL,
    CourseTitle   CHAR (30) NOT NULL,
    PRIMARY KEY (Instructor, CourseTitle));

CREATE TABLE StudentCourses {
    StudentID     INTEGER,
    Instructor        CHAR (20),
    CourseTitle   CHAR (30),
    FOREIGN KEY (Instructor, CourseTitle) REFERENCES Courses);
```

Note that this `REFERENCES` clause does not specify column names because the foreign key refers to the primary key of the `courses` table.

SQL evaluates the referential constraint to see if it satisfies the following search condition:

```
(StudentCourses.Ieacher IS NULL
 OR StudentCourses.CourseTitle IS NULL)
 OR EXISTS (SELECT * FROM StudentCourses WHERE
 (StudentCourses.Instructor = Courses.Instructor AND
 StudentCourses.CourseTitle = Courses.CourseTitle)
 )
```

**Note:** INSERT, UPDATE, or DELETE statements that cause the search condition to be false violate the constraint, fail, and generate an error.
In the following example, which creates a table with two column level check constraints and one table level check constraint, each constraint is defined with a name:

```
CREATE TABLE supplier (
    SuppNum   INTEGER NOT NULL,
    Name      CHAR (30),
    Status    SMALLINT CONSTRAINT StatusCheckCon
               CHECK (Supplier.Status BETWEEN 1 AND 100 ),
    City      CHAR (20) CONSTRAINT CityCheckCon CHECK
               (Supplier.City IN ('New York', 'Boston', 'Chicago')),
    CONSTRAINT SuppTabCheckCon CHECK (Supplier.City <> 'Chicago'
               OR Supplier.Status = 20)) ;
```

**UPDATE**

Updates the rows and columns of the specified table with the given values for rows that satisfy the `search_condition`.

When updating row(s) of a multi-tenant table, a regular tenant can only update rows in its partition, and the rows remain in the same tenant partition, but a super-tenant or a DBA can update rows in all the tenant partitions and group partitions. A super-tenant or a DBA may refine which tenants' rows must be affected by using the `tenantid_tbl()` or the `tenantName_tbl()` function in the `WHERE` clause `search_condition`.

**Syntax**

```
UPDATE table_name
    SET assignment [, assignment ] , ...
    [ WHERE search_condition ]
;
```

**assignment:**

The syntax for `assignment` is given below:

**Syntax**

```
column = { expr | NULL }
| { column [ , column ] , ... } = ( expr [ , expr ] )
| { column [ , column ] , ... } = ( query_expression )
```

**Notes**

- If you specify the optional `WHERE` clause, only rows that satisfy the `search_condition` are updated. If you do not specify a `WHERE` clause, all rows of the table are updated.
- If the expressions in the `SET` clause are dependent on the columns of the target table, the expressions are evaluated for each row of the table.
- If a `query_expression` is specified on the right-hand side of an `assignment`, the number of expressions in the first `SELECT` clause must be the same as the number of columns listed on the left-hand side of the `assignment`. 
• If a query _expression is specified on the right-hand side of an assignment, it must return a single row.

• If a table has check constraints and if the columns to be updated are part of a check expression, then the check expression is evaluated. If the result of the evaluation is **FALSE**, then, the **UPDATE** statement fails.

• If a table has a primary or candidate key, and if the columns to be updated are a part of the primary or candidate key, SQL checks to determine if there is a corresponding row in the referencing table. If there is no corresponding row, then, the **UPDATE** statement fails.

Column names in the **SET** clause do not need a table_name qualifier. Since an **UPDATE** statement affects a single table, columns in the **SET** clause are implicitly qualified to the table name identified in the **UPDATE** clause. The following is an example of an **UPDATE** statement:

**Examples**

The following examples illustrate the **UPDATE** statement:

The example below illustrates the different forms of **UPDATE** statement.

```
UPDATE Orderline
SET Qty = 186
WHERE Ordernum = 22;

Update Orderline
SET (Itemnum) =
(Select Itemnum
FROM Item
WHERE Itemname = 'Tennis balls')
WHERE Ordernum = 20;

UPDATE Orderline
SET (Qty) = (200 * 30)
WHERE OrderNum = 19;

UPDATE OrderLine
SET (ItemNum, Price) =
(SELECT ItemNum, Price * 3
FROM Item
WHERE ItemName = 'gloves')
WHERE OrderNum = 21;
```

The example below updates the postal code to '99999' for a customer '1428' for the tenant **SNCSoftware**.

```
UPDATE pub.mtcustomer
SET postalcode = '99999'
WHERE custnum = 1428 AND tenantName_tbl (pub.mtcustomer) = 'SNCSoftware';
```
The example below updates the postal code to '99999' for the customer '1428' for all tenants:

```sql
UPDATE pub.mtcustomer
SET postalcode = '99999'
WHERE custnum = 1428;
```

Authorization

To update rows and columns of a particular table, you must have DBA privileges or UPDATE privileges on all the specified columns of the target table, and SELECT privilege on all the other tables referred to in the statement.

A user mapped to a super-tenant can view data for all the tenants only if the user has the required permissions on the tables referenced in the statement.

Related statements

SELECT, OPEN, FETCH

---

**UPDATE STATISTICS**

Queries data tables and updates the following statistics:

- Table cardinality
- Index statistics
- Column data distribution for columns that are index components
- Column data distribution for columns that are not index components

**Syntax**

```sql
UPDATE ( [ TABLE | INDEX | ALL ] COLUMN ) STATISTICS
[ AND ]; ... [ FOR table_name ];
```

**Examples**

The following example shows default commands for table cardinality and data distribution for index component columns:

```sql
UPDATE STATISTICS FOR Customer;
```

The following example shows commands for table cardinality only:

```sql
UPDATE TABLE STATISTICS FOR Customer;
```
The following example shows commands for new index statistics:

```
UPDATE INDEX STATISTICS FOR Customer;
```

The following example shows commands for updating column statistics for index columns only:

```
UPDATE COLUMN STATISTICS FOR Customer;
```

The following example shows commands for updating statistics for all columns:

```
UPDATE ALL COLUMN STATISTICS FOR Customer;
```

The following example shows commands to obtain table cardinality and new index statistics and column statistics for all columns:

```
UPDATE TABLE STATISTICS AND INDEX STATISTICS AND ALL COLUMN STATISTICS FOR Customer;
```

**Notes**

- All statistics are obtained online. Obtaining statistics does not require an exclusive lock on the schema or any table locks. Rows written to statistics tables will be exclusively locked, as in every transaction performing updates. Therefore, statistics can be obtained while normal database operations continue.

- Specifying `TABLE STATISTICS` obtains table cardinality only. Table cardinalities are stored in the `SYSTABLSTAT` system catalog table.

- Specifying `INDEX STATISTICS` obtains statistics on the number of unique values in each index. Index statistics are stored in the `SYSIDXSTAT` system catalog table.

- Specifying `COLUMN STATISTICS` (without `ALL`) obtains statistics on the data distribution of values for each column that is an index key component.

- Specifying `ALL COLUMN STATISTICS` obtains statistics on the data distribution of values for all columns.

- The `STATISTICS` phrase can be repeated so that up to three statistics can be requested by a single `UPDATE STATISTICS` statement.

- By default, for the simple statement `UPDATE STATISTICS`, where the type of statistics is not specified, SQL will obtain table and index column statistics. This is equivalent to the statement `UPDATE TABLE STATISTICS AND COLUMN STATISTICS`.

- A table containing `LONG` data types can get table, index, and/or column statistics. The columns that are `LONG` data types cannot get statistics.

- Obtaining table statistics runs in time proportional to the table’s primary index.
• Obtaining column statistics runs in time proportional to the table’s primary index, plus an additional amount proportional to the number of columns in the table.

• Obtaining index statistics runs in time proportional to the total size for all indexes for the table.

• Table statistics are often the most useful statistic, as they influence join order substantially.

• Index statistics are important when a table has five or more indexes. This is especially true if some of the indexes are similar to one another.

• Column statistics are the most useful when applications use range predicates, such as BETWEEN and the operators <, <=, > and >=.

**Note:** To get the best SQL query performance, or if a SQL performance problem occurs, be sure that the database has a full set of SQL statistics. To get a full set of SQL statistics, execute this SQL statement:

```
UPDATE TABLE STATISTICS AND INDEX STATISTICS AND COLUMN STATISTICS;
```

At a slightly longer execution time, you can get even better SQL statistics by doing:

```
UPDATE TABLE STATISTICS AND INDEX STATISTICS AND ALL COLUMN STATISTICS;
```

**Authorization**

Must have **DBA** privilege, **SELECT** privilege, or ownership of table.
OpenEdge SQL Functions

This section provides detailed information on each SQL function. A description for each function provides the following information:

- A definition of the function
- The syntax of the function’s proper usage
- A code sample that shows how the function works
- Any associated notes

About OpenEdge SQL functions

A function is an SQL expression that returns a value based on arguments supplied. OpenEdge® SQL supports five aggregate functions and 90 scalar functions.

Aggregate functions

Aggregate functions calculate a single value for a collection of rows in a result table. If the function is in a statement with a GROUP BY clause, it returns a value for each group in the result table. Aggregate functions are also called set or statistical functions. Aggregate functions cannot be nested. The aggregate functions are:

- AVG
- COUNT
- MAX
- MIN
- SUM
Scalar functions

 Scalar functions calculate a value based on another single value. Scalar functions are also called value functions and can be nested.

**ABS**

Computes the absolute value of expression.

**Syntax**

```sql
ABS ( expression )
```

**Example**

This example illustrates the ABS function:

```sql
SELECT ABS (MONTHS_BETWEEN (SYSDATE, order_date))
FROM orders
WHERE ABS (MONTHS_BETWEEN (SYSDATE, order_date)) > 3 ;
```

**Notes**

- The argument to the function must be of type TINYINT, SMALLINT, INTEGER, NUMERIC, REAL, or FLOAT.
- The result is of type NUMERIC.
- If the argument expression evaluates to NULL, the result is NULL.

**Compatibility**

ODBC compatible
ACOS

Returns the arccosine of expression.

Syntax

ACOS ( expression )

Example

In this example, which illustrates two ways to use the ACOS function, the first SELECT statement returns the arccosine in radians, and the second returns the arccosine in degrees:

```
select acos (.5) 'Arccosine in radians' from syscalctable;

ARCCOSINE IN RADIANS
----------------------
1.047197551196598
1 record selected

select acos (.5) * (180/ pi()) 'Arccosine in degrees' from syscalctable;

ARCCOSINE IN DEGREES
----------------------
59.999999999999993
1 record selected
```

Notes

- ACOS takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

- The result is expressed in radians and is in the range -Pi/2 to Pi/2 radians. To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

- The expression must be in the range -1 to 1.

- The expression must evaluate to an approximate numeric data type.

Compatibility

ODBC compatible
ADD_MONTHS

Adds to the date value specified by the `date_expression`, the given number of months specified by `integer_expression`, and returns the resultant date value.

**Syntax**

```
ADD_MONTHS ( date_expression , integer_expression )
```

**Example**

This example illustrates the `ADD_MONTHS` function:

```
SELECT *  
FROM  customer  
WHERE ADD_MONTHS (start_date, 6) > SYSDATE ;
```

**Notes**

- The first argument must be of `DATE` type.
- The second argument to the function must be of `NUMERIC` type.
- The result is of type `DATE`.
- If any of the arguments evaluates to `NULL`, the result is `NULL`.

**Compatibility**

Progress extension

---

ASCII

Returns the ASCII value of the first character of the given character expression.

**Syntax**

```
ASCII ( char_expression )
```

**Example**

The following example shows how to use the `ASCII` function:

```
SELECT ASCII ( PostalCode )  
FROM Customer;
```

**Notes**

- The argument to the function must be of type `CHARACTER`.
- The result is of type `INTEGER`.
- If the argument `char_expression` evaluates to `NULL`, the result is `NULL`.
• The `ASCII` function is character-set dependent and supports multi-byte characters. The function returns the character encoding integer value of the first character of `char_expression` in the current character set. If `char_expression` is a literal string, the result is determined by the character set of the SQL client. If `char_expression` is a column in the database, the character set of the database determines the result.

**Compatibility**

ODBC compatible

---

### ASIN

Returns the arcsine of `expression`.

**Syntax**

```
ASIN ( expression )
```

**Example**

In the following example, which shows how to use the `ASIN` function, the first `SELECT` statement returns the arcsine in degrees, and the second returns the arcsine in radians:

```
SELECT ASIN (1) * (180/ pi()) 'Arcsine in degrees' FROM SYSPROGRESS.SYSCALCTABLE;

ARCSINE IN DEGREES
-------------------
90.000000000000000

1 record selected

SELECT ASIN (1) 'Arcsine in radians' FROM SYSPROGRESS.SYSCALCTABLE;

ARCSINE IN RADIANS
------------------
1.570796326794897

1 record selected
```

`ASIN` takes the ratio (`expression`) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result is expressed in radians and is in the range `-pi/2` to `pi/2` radians. To convert degrees to radians, multiply degrees by `pi/180`. To convert radians to degrees, multiply radians by `180/pi`.
ATAN

Notes

- The expression must be in the range -1 to 1.
- The expression must evaluate to an approximate numeric data type.

Compatibility

ODBC compatible

ATAN

Returns the arctangent of expression.

Syntax

ATAN ( expression )

Example

The following example illustrates two ways to use the ATAN function:

```
select atan (1) * (180/ pi()) 'Arctangent in degrees' from syscalctable;

ARCTANGENT IN DEGREES
-----------------------
45.000000000000000
1 record selected

select atan (1) 'Arctangent in radians' from syscalctable;

ARCTANGENT IN RADIANS
-----------------------
0.785398163397448
1 record selected
```

ATAN takes the ratio (expression) of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

The result is expressed in radians and is in the range -Pi/2 to Pi/2 radians. To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

Notes

- The expression must be in the range -1 to 1.
- The expression must evaluate to an approximate numeric data type.

Compatibility

ODBC compatible
ATAN2

Returns the arctangent of the x and y coordinates specified by expression1 and expression2.

Syntax

```
ATAN2 ( expression1 , expression2 )
```

Example

The following example illustrates two ways to use the ATAN2 function:

```
select atan2 (1,1) * (180/ pi()) 'Arctangent in degrees' from syscalctable;

ARCTANGENT IN DEGREES
-----------------------
45.000000000000000
1 record selected

select atan2 (1,1) 'Arctangent in radians' from syscalctable;

ARCTANGENT IN RADIANS
-----------------------
0.785398163397448
1 record selected
```

Notes

- **ATAN2** takes the ratio of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.
- **expression1** and **expression2** specify the x and y coordinates of the end of the hypotenuse opposite the angle.
- The result is expressed in radians and is in the range -pi/2 to pi/2 radians. To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.
- Both **expression1** and **expression2** must evaluate to approximate numeric data types.

Compatibility

ODBC compatible
AVG

Computes the average of a collection of values. The keyword `DISTINCT` specifies that the duplicate values are to be eliminated before computing the average.

**Syntax**

```
AVG ( { [ ALL ] expression } | { DISTINCT column_ref } )
```

**Example**

This example illustrates the `AVG` function:

```
SELECT AVG (salary)
FROM employee
WHERE deptno = 20;
```

**Notes**

- NULL values are eliminated before the average value is computed. If all the values are NULL, the result is NULL.
- The argument to the function must be of type `SMALLINT`, `INTEGER`, `NUMERIC`, `REAL`, or `FLOAT`.
- The result is of type `NUMERIC`.

CASE

Specifies a series of search conditions and associated result expressions. The general form is called a searched case expression. SQL returns the value specified by the first result expression whose associated search condition evaluates as true. If none of the search conditions evaluates as true, the `CASE` expression returns a `NULL` value, or the value of some other default expression if the `CASE` expression includes the `ELSE` clause.

`CASE` also supports syntax for a shorthand notation, called a simple case expression, for evaluating whether one expression is equal to a series of other expressions.

**Syntax**

```
CASE
WHEN search_condition THEN { result_expr | NULL }
[ ... ]
[ ELSE expr | NULL ]
END
```

```
searched_case_expr | simple_case_expr
```
**CASE**

Specifies a searched case expression. It must be followed by one or more WHEN-THEN clauses, each specifying a search condition and corresponding expression.

**WHEN search_condition THEN { result_expr | NULL }**

Specifies a search condition and corresponding expression. SQL evaluates search_condition. If search_condition evaluates as true, CASE returns the value specified by result_expr, or NULL, if the clause specifies THEN NULL.

If search_condition evaluates as false, SQL evaluates the next WHEN-THEN clause, if any, or the ELSE clause, if it is specified.

**CASE primary_expr**

Specifies a simple case expression. In a simple case expression, one or more WHEN-THEN clauses specify two expressions.

**WHEN expr THEN { result_expr | NULL }**

Prompts SQL to evaluate expr and compare it with primary_expr specified in the CASE clause. If they are equal, CASE returns the value specified by result_expr (or NULL, if the clause specifies THEN NULL).

If expr is not equal to primary_expr, SQL evaluates the next WHEN-THEN clause, if any, or the ELSE clause, if it is specified.

**ELSE { expr | NULL }**

Specifies an optional expression whose value SQL returns if none of the conditions specified in WHEN-THEN clauses are satisfied. If the CASE expression omits the ELSE clause, it is the same as specifying ELSE NULL.

**Examples**

A simple case expression can always be expressed as a searched case expression. This example illustrates a simple case expression:

```
CASE primary_expr
  WHEN expr1 THEN result_expr1
  WHEN expr2 THEN result_expr2
  ELSE expr3
END
```
The simple case expression in the preceding `CASE` example is equivalent to the following searched case expression:

```
CASE
  WHEN primary_expr = expr1 THEN result_expr1
  WHEN primary_expr = expr2 THEN result_expr2
  ELSE expr3
END
```

The following example shows a searched case expression that assigns a label denoting suppliers as 'In Mass' if the state column value is 'MA':

```
SELECT name, city,
  CASE
    WHEN state = 'MA' THEN 'In Mass' ELSE 'Not in Mass'
  END
FROM supplier;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>searched_case(State,MA,In Mass,)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GolfWorld Suppl</td>
<td>Boston</td>
<td>In Mass</td>
</tr>
<tr>
<td>Pool Swimming S</td>
<td>Valkeala</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>Nordic Ski Whol</td>
<td>Hingham</td>
<td>In Mass</td>
</tr>
<tr>
<td>Champion Soccer</td>
<td>Harrow</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>ABC Sports Supp</td>
<td>Boston</td>
<td>In Mass</td>
</tr>
<tr>
<td>Seasonal Sports</td>
<td>Bedford</td>
<td>In Mass</td>
</tr>
<tr>
<td>Tennis Supplies</td>
<td>Boston</td>
<td>In Mass</td>
</tr>
<tr>
<td>Boating Supplie</td>
<td>Jacksonville</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>Aerobic Supplie</td>
<td>Newport Beach</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>Sports Unlimite</td>
<td>Irving</td>
<td>Not in Mass</td>
</tr>
</tbody>
</table>

The following example shows the equivalent simple case expression:

```
SELECT name, city,
  CASE state
    WHEN 'MA' THEN 'In Mass' ELSE 'Not in Mass'
  END
FROM supplier;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>simple_case(State,MA,In Mass,)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GolfWorld Suppl</td>
<td>Boston</td>
<td>In Mass</td>
</tr>
<tr>
<td>Pool Swimming S</td>
<td>Valkeala</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>Nordic Ski Whol</td>
<td>Hingham</td>
<td>In Mass</td>
</tr>
<tr>
<td>Champion Soccer</td>
<td>Harrow</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>ABC Sports Supp</td>
<td>Boston</td>
<td>In Mass</td>
</tr>
<tr>
<td>Seasonal Sports</td>
<td>Bedford</td>
<td>In Mass</td>
</tr>
<tr>
<td>Tennis Supplies</td>
<td>Boston</td>
<td>In Mass</td>
</tr>
<tr>
<td>Boating Supplie</td>
<td>Jacksonville</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>Aerobic Supplie</td>
<td>Newport Beach</td>
<td>Not in Mass</td>
</tr>
<tr>
<td>Sports Unlimite</td>
<td>Irving</td>
<td>Not in Mass</td>
</tr>
</tbody>
</table>
Notes

- This function is not allowed in a GROUP BY clause.
- Arguments to this function cannot be query expressions.

Compatibility

SQL compatible

CAST

Converts an expression to another data type. The first argument is the expression to be converted. The second argument is the target data type.

The length option for the data_type argument specifies the length for conversions to CHAR and VARCHAR data types. If omitted, the default is 1 byte.

If the expression evaluates to NULL, the result of the function is null. Specifying NULL with the CAST function is useful for set operations, such as UNION, that require two tables to have the same structure. CAST NULL allows you to specify a column of the correct data type, so a table with a similar structure to another, but with fewer columns, can be in a union operation with the other table.

The CAST function provides a data-type-conversion mechanism compatible with the SQL standard.

Use the CONVERT function, enclosed in the ODBC escape clause { fn }, to specify ODBC-compliant syntax for data type conversion. See “CONVERT (ODBC compatible)” section on page 141 for more information.

Syntax

```
CAST ( { expression | NULL } AS data_type [ ( length ) ] )
```

Example

The following SQL example uses CAST to convert an integer field from a catalog table to a CHARACTER data type:

```
SELECT CAST(fld AS CHAR(25)), fld FROM sysprogress.syscalctable;
```

```
<table>
<thead>
<tr>
<th>CONVERT(CHARACTER(25), FLD)</th>
<th>FLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
```

1 record selected

Compatibility

SQL compatible
CEILING

Returns the smallest integer greater than or equal to expression.

Syntax

`CEILING ( expression )`

Example

This example illustrates the `CEILING` function:

```
SELECT CEILING (32.5) 'Ceiling'
FROM SYSPROGRESS.SYSCALCTABLE;
```

Note

The expression must evaluate to a numeric data type.

Compatibility

ODBC compatible

CHAR

Returns a character string with the first character having an ASCII value equal to the argument expression. `CHAR` is identical to `CHR` but provides ODBC-compatible syntax.

Syntax

`CHAR ( integer_expression )`

Example

This example illustrates the `CHAR` function:

```
SELECT *
FROM  customer
WHERE SUBSTR (zip, 1, 1) = CHAR (53) ;
```

Notes

- The argument to the function must be of type `INTEGER`, `TINYINT`, or `SMALLINT`.
- The result is of type `CHARACTER`.
- If the argument `integer_expression` evaluates to `NULL`, the result is `NULL`.
- The `CHAR` and `CHR` functions are character-set dependent and support single-byte and multi-byte characters. If `integer_expression` is a valid character encoding integer value in the current SQL server character set, the function returns the correct character. If it is not a valid character, the function returns a `NULL` value.

Compatibility

ODBC compatible
CHR

Returns a character string with the first character having an ASCII value equal to the argument expression.

Syntax

```
CHR ( integer_expression )
```

Example

This example illustrates the `CHR` function and the `SUBSTR (substring)` function:

```
SELECT *
FROM  customer
WHERE SUBSTR (zip, 1, 1) = CHR (53) ;
```

Notes

- The argument to the function must be of type `INTEGER`, `TINYINT`, or `SMALLINT`.
- The result is of type `CHARACTER`.
- If the argument `integer_expression` evaluates to `NULL`, the result is `NULL`.
- The `CHR` and `CHAR` functions are character-set dependent, and support multi-byte characters. If `integer_expression` is a valid character encoding integer value in the current SQL server character set, the function returns the correct character. If it is not a valid character the function returns a `NULL` value.

Compatibility

Progress extension
COALESCE

Specifies a series of expressions and returns the first expression whose value is not NULL. If all the expressions evaluate as null, COALESCE returns a NULL value.

Syntax

\[
\text{COALESCE}\ ( \text{expression1, expression2} \ [\ldots])
\]

The COALESCE syntax is shorthand notation for a common case that can also be represented in a CASE expression. The following two formulations are equivalent:

\[
\text{COALESCE}\ ( \text{expression1} \ , \text{expression2} \ , \text{expression3} \ )
\]

\[
\text{CASE} \\
\quad \text{WHEN expression1 IS NOT NULL THEN expression1} \\
\quad \text{WHEN expression2 IS NOT NULL THEN expression2} \\
\quad \text{ELSE expression3} \\
\text{END}
\]

Example

This example illustrates the COALESCE function:

\[
\text{SELECT COALESCE (end_date, start_date) from job_hist;}
\]

Notes

- This function is not allowed in a GROUP BY clause.
- Arguments to this function cannot be query expressions.

Compatibility

- SQL compatible

CONCAT

Returns a concatenated character string formed by concatenating two arguments.

Syntax

\[
\text{CONCAT}\ ( \text{char_expression} \ , \text{char_expression} \ )
\]

Example

This example illustrates the CONCAT function:

\[
\text{SELECT last_name, empno, salary} \\
\quad \text{FROM customer} \\
\quad \text{WHERE project = CONCAT('US',proj_nam);}
\]
Notes

- Both of the arguments must be of type CHARACTER or VARCHAR.
- The result is of type VARCHAR.
- If any of the argument expressions evaluate to NULL, the result is NULL.
- The two char_expression expressions and the result of the CONCAT function can contain multi-byte characters.

Compatibility

ODBC compatible

CONVERT (ODBC compatible)

Converts an expression to another data type. The first argument is the expression to be converted. The second argument is the target data type.

If the expression evaluates to NULL, the result of the function is NULL.

The ODBC CONVERT function provides ODBC-compliant syntax for data type conversions. You must enclose the function with the ODBC escape clause { fn } to use ODBC-compliant syntax.

Syntax

{ fn CONVERT ( expression , data_type ) }

Note

Braces are part of the actual syntax. The following data types are used:

| SQL_BINARY | SQL_BIT  | SQL_CHAR | SQL_DATE | SQL_DECIMAL |
| SQL_DOUBLE | SQL_FLOAT | SQL_INTEGER | SQL_REAL |
| SQL_SMALLINT | SQL_TIME  | SQL_TIMESTAMP | SQL_TINYINT |
| SQL_VARBINARY | SQL_VARCHAR |

Compatibility

ODBC compatible
CONVERT (Progress extension)

Converts an expression to another data type. The first argument is the target data type. The second argument is the expression to be converted to that type.

The length option for the data_type argument specifies the length for conversions to CHAR and VARCHAR data types. If omitted, the default is 30 bytes.

If the expression evaluates to NULL, the result of the function is NULL.

The CONVERT function syntax is similar to, but not compatible with, the ODBC CONVERT function. Enclose the function in the ODBC escape clause { fn } to specify ODBC-compliant syntax. See the ODBC compatible CONVERT function for more information.

Syntax

CONVERT ( 'data_type [ (length) ]', expression )

Example

The following SQL example uses the CONVERT function to convert an INTEGER field from a system table to a character string:

```sql
SELECT CONVERT('CHAR', fld), fld FROM sysprogress.syscalctable;
```

<table>
<thead>
<tr>
<th>CONVERT(CHAR,FLD)</th>
<th>FLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

1 record selected

```sql
SELECT CONVERT('CHAR(35)', fld), fld FROM sysprogress.syscalctable;
```

<table>
<thead>
<tr>
<th>CONVERT(CHAR(35),FLD)</th>
<th>FLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

1 record selected

Note

When data_type is CHARACTER( length ) or VARCHAR( length ), the length specification represents the number of characters. The converted result can contain multi-byte characters.

Compatibility

Progress extension
COS

Returns the cosine of expression.

Syntax

COS ( expression )

Example

This example illustrates the COS function:

```sql
select cos(45 * pi()/180) 'Cosine of 45 degrees'
  from sysprogress.syscalctable;
```

<table>
<thead>
<tr>
<th>COSINE OF 45 DEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.707106781186548</td>
</tr>
</tbody>
</table>

1 record selected

Notes

• COS takes an angle expression and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

• The expression specifies an angle in radians.

• The expression must evaluate to an approximate numeric data type.

• To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

Compatibility

ODBC compatible

COUNT

Computes either the number of rows in a group of rows or the number of non-NULL values in a group of values.

Syntax

COUNT ( { [ ALL ] expression } | { DISTINCT column_ref } | * )

Example

This example illustrates the COUNT function:

```sql
SELECT COUNT (*)
  FROM orders
  WHERE order_date = SYSDATE ;
```
**CURDATE**

**Notes**
- The keyword `DISTINCT` specifies that the duplicate values are to be eliminated before computing the count.
- If the argument to `COUNT` function is `'*'`, then the function computes the count of the number of rows in a group.
- If the argument to `COUNT` function is not `'*'`, then `NULL` values are eliminated before the number of rows is computed.
- The argument `column_ref` or `expression` can be of any type.
- The result of the function is of `BIGINT` data type. The result is never `NULL`.

**CURDATE**

Returns the current date as a `DATE` value. This function takes no arguments.

**Syntax**

```sql
CURDATE ( )
```

**Example**

The following example shows how to use the `CURDATE` function:

```sql
INSERT INTO objects (object_owner, object_id, create_date)
VALUES (USER, 1001, CURDATE()) ;
```

**Note**

SQL statements can refer to `CURDATE` anywhere they can refer to a `DATE` expression.

**Compatibility**

ODBC compatible

**CURTIME**

Returns the current time as a `TIME` value. This function takes no arguments.

**Syntax**

```sql
CURTIME ( )
```

**Example**

This example illustrates how to use the `CURTIME` function to `INSERT` the current time into the `create_time` column of the `objects` table:

```sql
INSERT INTO objects (object_owner, object_id, create_time)
VALUES (USER, 1001, CURTIME()) ;
```
**CURRVAL**

CURRVAL returns the current value of a sequence, and uses the following syntax to reference the current value of a sequence.

**Syntax**

\[ \text{schema.sequence.CURRVAL} \]

- **schema**
  - Specifies the schema that contains the sequence. To refer to the current value of a sequence in the schema of another user, you must have `SELECT` object privilege on the sequence.

- **sequence**
  - Specifies the name of the sequence whose current value you want.

**Use CURRVAL in:**

- The `SELECT` list of a `SELECT` statement not contained in a subquery or view
- The `SELECT` list of a subquery in an `INSERT` statement
- The `VALUES` clause of an `INSERT` statement
- The `SET` clause of an `UPDATE` statement

**CURRVAL cannot be used in:**

- A query of a view
- A `SELECT` statement with a `GROUP BY` clause that references a sequence
- A `SELECT` statement with an `ORDER BY` clause that references a sequence
- A `SELECT` statement that is combined with another `SELECT` statement with the `UNION`, `INTERSECT`, or `MINUS` set operator
- The `WHERE` clause of a `SELECT` or `UPDATE` statement
- The `DEFAULT` value of a column in a `CREATE TABLE` or `ALTER TABLE` statement
- The condition of a `CHECK` constraint

---

**Note**

SQL statements can refer to `CURTIME` anywhere they can refer to a `TIME` expression.

**Compatibility**

ODBC compatible
Example

In the following example, the OpenEdge SQL sequence generator returns the current value of the customer sequence:

```
SELECT customer_sequence.CURRVAL FROM pub.customer;
```

---

**DATABASE**

Returns the name of the database corresponding to the current connection name. This function takes no arguments, and the trailing parentheses are optional.

**Syntax**

```
DATABASE [ ( ) ]
```

**Example**

The following example shows how to use the `DATABASE` function:

```
select database() from t2;
DATABASE
--------
steel
1 record selected
```

**Compatibility**

ODBC compatible
DAYNAME

Returns a character string containing the name of the day (for example, Sunday through Saturday) for the day portion of `date_expression`. The argument `date_expression` can be the name of a column, the result of another scalar function, or a date or timestamp literal.

Syntax

```
DAYNAME ( date_expression )
```

Example

This example illustrates the `DAYNAME` function:

```
SELECT *
FROM orders
WHERE order_no = 342 and DAYNAME(order_date) = 'SATURDAY';
```

<table>
<thead>
<tr>
<th>ORDER_NO</th>
<th>ORDER_DATE</th>
<th>REFERENCE</th>
<th>CUST_NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>342</td>
<td>08/10</td>
<td>tdfg/101</td>
<td>10001</td>
</tr>
</tbody>
</table>

1 record selected

Compatibility

ODBC compatible

DAYOFMONTH

Returns the day of the month in the argument as a short integer value in the range of 1–31. The argument `date_expression` can be the name of a column, the result of another scalar function, or a date or timestamp literal.

Syntax

```
DAYOFMONTH ( date_expression )
```

Example

This example illustrates the `DAYOFMONTH` function:

```
SELECT *
FROM orders
WHERE DAYOFMONTH (order_date) = 14;
```

Notes

- The `date_expression` argument must be of type `DATE`.
- If `date_expression` is supplied as a date literal, it can be any of the valid `date_literal` formats where the day specification (DD) precedes the month specification (MM).
DAYOFWEEK

- The result is of type SHORT.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

ODBC compatible

DAYOFWEEK

Returns the day of the week in the argument as a short integer value in the range of 1–7.

The argument date_expression can be the name of a column, the result of another scalar function, or a date or timestamp literal.

Syntax

```
DAYOFWEEK ( date_expression )
```

Example

The following example shows how to use the DAYOFWEEK function:

```
SELECT *
  FROM orders
  WHERE DAYOFWEEK (order_date) = 2 ;
```

Notes

- The argument to the function must be of type DATE.
- If date_expression is supplied as a date literal, it can be any of the valid date_literal formats where the day specification (DD) precedes the month specification (MM).
- The result is of type SHORT.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

ODBC compatible
DAYOFYEAR

Returns the day of the year in the argument as a short integer value in the range of 1–366. The argument `date_expression` can be the name of a column, the result of another scalar function, or a date or timestamp literal.

**Syntax**

```
DAYOFYEAR ( date_expression )
```

**Example**

This example illustrates the DAYOFYEAR function:

```
SELECT *
FROM orders
WHERE DAYOFYEAR (order_date) = 300 ;
```

**Notes**

- The argument to the function must be of type `DATE`.
- The result is of type `SHORT`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.

**Compatibility**

ODBC compatible

---

DB_NAME

Returns the name of the database corresponding to the current connection name.

**Syntax**

```
DB_NAME ( )
```

**Example**

This example illustrates the DB_NAME function:

```
SELECT DB_NAME() FROM T2;
```

```
  DB_NAME
  -------
demo

1 record selected
```

**Compatibility**

Progress extension
DECODE

DECODE

Compares the value of the first argument expression with each search_expression and, if a match is found, returns the corresponding match_expression. If no match is found, then the function returns the default_expression. If a default_expression is not specified and no match is found, then the function returns a NULL value.

Syntax

```
DECODE ( expression, search_expression, match_expression
    [, search_expression, match_expression . . ]
    [, default_expression ] )
```

Example

This example illustrates one way to use the DECODE function:

```
SELECT ename, DECODE (deptno,
    10, 'ACCOUNTS',
    20, 'RESEARCH',
    30, 'SALES',
    40, 'SUPPORT',
    'NOT ASSIGNED'
) FROM employee ;
```

Notes

- Use a simple case expression when SQL-compatible syntax is a requirement.
- The first argument expression can be of any type. The types of all search_expressions must be compatible with the type of the first argument.
- The match_expressions can be of any type. The types of all match_expressions must be compatible with the type of the first match_expression.
- The type of the default_expression must be compatible with the type of the first match_expression.
- The type of the result is the same as that of the first match_expression.
- If the first argument expression is NULL, then the value of the default_expression is returned, if it is specified. Otherwise NULL is returned.

Compatibility

Progress extension
DEGREES

Returns the number of degrees in an angle specified in radians by *expression*.

**Syntax**

```
DEGREES ( expression )
```

**Example**

This example illustrates the DEGREES function:

```
SELECT DEGREES(3.14159265359) 'Degrees in pi Radians'
  FROM SYSPROGRESS.SYSCALCTABLE;
```

**Notes**

- The *expression* specifies an angle in radians.
- The *expression* must evaluate to a numeric data type.

**Compatibility**

ODBC compatible

EXP

Returns the exponential value of *expression* (*e* raised to the power of *expression*).

**Syntax**

```
EXP ( expression )
```

**Example**

This example illustrates the EXP function:

```
SELECT EXP( 4 ) 'e to the 4th power' from sysprogress.syscalctable;
```

**Note**

*expression* must evaluate to an approximate numeric data type.

**Compatibility**

ODBC compatible
FLOOR

Returns the largest integer less than or equal to expression.

Syntax

FLOOR ( expression )

Example

This example illustrates the FLOOR function:

```
SELECT FLOOR (32.5) 'Floor' from sysprogress.syscalctable;
```

Note

eexpression must evaluate to a numeric data type.

Compatibility

ODBC compatible

GREATEST

Returns the greatest value among the values of the given expressions.

Syntax

GREATEST ( expression , expression ... )

Example

This example illustrates the GREATEST function:

```
SELECT cust_no, last_name,
    GREATEST (ADD_MONTHS (start_date, 10), SYSDATE)
FROM customer;
```

Notes

- The first argument to the function can be of any type. However, the types of the subsequent arguments must be compatible with that of the first argument.
- The type of the result is the same as that of the first argument.
- If any of the argument expressions evaluate to NULL, the result is NULL.
- When the data type of an expression is either CHARACTER(length) or VARCHAR(length), the expression can contain multi-byte characters. The sort weight for each character is determined by the collation table in the database.

Compatibility

Progress extension
HOUR

Returns the hour in the argument as a short integer value in the range of 0–23.

Syntax

```
HOUR ( time_expression )
```

Example

This example illustrates the HOUR function:

```
SELECT *
FROM arrivals
WHERE HOUR (in_time) < 12 ;
```

Notes

- The argument to the function must be of type TIME.
- The argument must be specified in the format `hh:mm:ss`.
- The result is of type SHORT.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

ODBC compatible

IFNULL

Returns `value` if `expr` is NULL. If `expr` is not NULL, IFNULL returns `expr`.

Syntax

```
IFNULL( expr, value)
```

Example

In this example, which illustrates the IFNULL function, the SELECT statement returns three rows with a NULL value in column `C1`, and two non-NULL values:

```
SELECT C1, IFNULL(C1, 9999) FROM TEMP ORDER BY C1;
```

<table>
<thead>
<tr>
<th>C1</th>
<th>IFNULL(C1,9999)</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>9999</td>
</tr>
<tr>
<td>9999</td>
<td>9999</td>
</tr>
<tr>
<td>9999</td>
<td>9999</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
The data type of `value` must be compatible with the data type of `expr`.

### Compatibility

ODBC compatible

---

## INITCAP

Returns the result of the argument character expression after converting the first character to uppercase and the subsequent characters to lowercase.

### Syntax

```
INITCAP ( char_expression )
```

### Example

The following example shows how to use the `INITCAP` function:

```
SELECT INITCAP (last_name) 
    FROM customer;
```

### Notes

- The `char_expression` must be of type `CHARACTER`.
- The result is of type `CHARACTER`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.
- A `char_expression` and the result can contain multi-byte characters. The uppercase conversion for the first character and the lowercase conversion for the rest of the characters is based on the case table in the `convmap` file. The default case table is `BASIC`.

### Compatibility

Progress extension

---

## INSERT

Returns a character string where `length` number of characters have been deleted from `string_exp1` beginning at `start_pos`, and `string_exp2` has been inserted into `string_exp1`, beginning at `start_pos`.

### Syntax

```
INSERT( string_exp1 , start_pos , length , string_exp2 )
```
Example

This example illustrates the `INSERT` function:

```
SELECT INSERT(last_name,2,4,'xx')
FROM customer
WHERE last_name = 'Goldman';
```

```
INSERT LAST_NAME,2,4,XX)
------------------------
Gxxan
1 record selected
```

The two letters 'o' and 'l' are deleted from the name 'Goldman' in the `last_name` column, and the letters 'xx' are inserted into the `last_name` column, beginning at the fourth character, overlaying the letters 'd' and 'm'.

Notes

- The `string_exp` can be type fixed-length or variable-length `CHARACTER`.
- The `start_pos` and `length` can be of data type `INTEGER`, `SMALLINT`, or `TINYINT`.
- The result string is of the type `string_exp1`.
- If any of the argument expressions evaluate to `NULL`, the result is `NULL`.
- If `start_pos` is negative or zero, the result string evaluates to `NULL`.
- If `length` is negative, the result evaluates to `NULL`.
- `string_exp1` and `string_exp2` and the result can contain multi-byte characters. This is determined by the character set of the SQL server. The `length` argument specifies a number of characters.

Compatibility

ODBC compatible

INSTR

Searches character string `char_expression1` for the character string `char_expression2`. The search begins at `start_pos` of `char_expression1`. If `occurrence` is specified, then `INSTR` searches for the `n`th occurrence, where `n` is the value of the fourth argument.

The position (with respect to the start of `char_expression1`) is returned if a search is successful. Zero is returned if no match can be found.

Syntax

```
INSTR ( char_expression1 , char_expression2
             [, , start_pos [, , occurrence ] ])
```
**LAST_DAY**

Returns the date corresponding to the last day of the month containing the argument date.

### Syntax

```
LAST_DAY ( date_expression )
```

### Example

This example illustrates the `LAST_DAY` function:

```
SELECT * 
FROM orders 
WHERE LAST_DAY (order_date) + 1 = '08/01/2003' ;
```

---

**INSTR**

This example illustrates the `INSTR` function:

```
SELECT cust_no, last_name 
FROM customer 
WHERE INSTR (LOWER (addr), 'heritage') > 0 ;
```

### Notes

- The first and second arguments must be `CHARACTER` data type.
- The third and fourth arguments, if specified, must be `SMALLINT` or `TINYINT` data type.
- The value for start position in a character string is the ordinal number of the character in the string. The very first character in a string is at position 1, the second character is at position 2, the n-th character is at position n.
- If you do not specify `start_pos`, a default value of 1 is assumed.
- If you do not specify `occurrence`, a default value of 1 is assumed.
- The result is `INTEGER` data type.
- If any of the argument expressions evaluate to `NULL`, the result is `NULL`.
- A `char_expression` and the result can contain multi-byte characters.

### Compatibility

Progress extension
Notes

- The argument to the function must be of type `DATE`.
- The result is of type `DATE`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.

Compatibility

Progress extension

---

**LCASE**

Returns the result of the argument character expression after converting all the characters to lowercase. `LCASE` is the same as `LOWER` but provides ODBC-compatible syntax.

**Syntax**

```
LCASE ( char_expression )
```

**Example**

This example illustrates the `LCASE` function:

```
SELECT *
FROM customer
WHERE LCASE (last_name) = 'smith' ;
```

Notes

- The argument to the function must be of type `CHARACTER`.
- The result is of type `CHARACTER`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.
- A `char_expression` and the result can contain multi-byte characters. The lowercase conversion is determined by the case table in the `convmap` file. The default case table is `BASIC`.

Compatibility

ODBC compatible
LEAST

Returns the lowest value among the values of the given expressions.

Syntax

LEAST ( expression , expression, ... )

Example

This example illustrates the LEAST function:

```
SELECT cust_no, last_name,
       LEAST (ADD_MONTHS (start_date, 10), SYSDATE)
FROM customer ;
```

Notes

- The first argument to the function can be of any type. However, the types of the subsequent arguments must be compatible with that of the first argument.
- The type of the result is the same as that of the first argument.
- If any of the argument expressions evaluate to NULL, the result is NULL.
- When the data type of an expression is either CHARACTER(length) or VARCHAR(length), the expression can contain multi-byte characters. The sort weight for each character is determined by the collation table in the database.

Compatibility

Progress extension

LEFT

Returns the leftmost count of characters of string_exp.

Syntax

LEFT ( string_exp , count )

Example

The following example shows how to use the LEFT function:

```
SELECT LEFT(last_name,4) FROM  customer WHERE last_name = 'Goldman';

LEFT(LAST_NAME),4)----------------
Gold
1 record selected
```
Notes

- `string_exp` can be fixed-length or variable-length CHARACTER data types.
- `count` can be INTEGER, SMALLINT, or TINYINT data types.
- If any of the arguments of the expression evaluate to NULL, the result is NULL.
- If the `count` is negative, the result evaluates to NULL.
- The `string_exp` and the result can contain multi-byte characters. The function returns the number of characters.

Compatibility

ODBC compatible

---

**LENGTH**

Returns the string length of the value of the given character expression.

**Syntax**

```
LENGTH ( char_expression )
```

**Example**

This example illustrates the `LENGTH` function:

```
SELECT last_name 'LONG LAST_NAME'
FROM customer
WHERE LENGTH (last_name) > 5 ;
```

**Notes**

- The argument to the function must be of type CHARACTER or VARCHAR.
- The result is of type INTEGER.
- If the argument expression evaluates to NULL, the result is NULL.
- `char_expression` can contain multi-byte characters. The function returns a number of characters.

**Compatibility**

ODBC compatible
LOCATE

Returns the location of the first occurrence of `char_expr1` in `char_expr2`. If the function includes the optional integer argument `start_pos`, `LOCATE` begins searching `char_expr2` at that position. If the function omits the `start_pos` argument, `LOCATE` begins its search at the beginning of `char_expr2`.

`LOCATE` denotes the first character position of a character expression as 1. If the search fails, `LOCATE` returns 0. If either character expression is `NULL`, `LOCATE` returns a `NULL` value.

**Syntax**

```
LOCATE( char_expr1 , char_expr2 , [ start_pos ] )
```

**Example**

In the following example, which uses two string literals as character expressions, `LOCATE` returns a value of 6:

```
SELECT LOCATE('this', 'test this test', 1) FROM TEST;
```

```
LOCATE(THIS, ------------
6
1 record selected
```

**Note**

`char_expr1` and `char_expr2` can contain multi-byte characters. The `start_pos` argument specifies the position of a starting character, not a byte position. The search is case sensitive. Character comparisons use the collation table in the database.

**Compatibility**

ODBC compatible
LOG10

Returns the base 10 logarithm of expression.

Syntax

```
LOG10 ( expression )
```

Example

This example illustrates the LOG10 function:

```
SELECT LOG10 (100) 'Log base 10 of 100' FROM SYSPROGRESS.SYSCALCTABLE;
```

Note

The expression must evaluate to an approximate numeric data type.

Compatibility

ODBC compatible

LOWER

Returns the result of the argument char_expression after converting all the characters to lowercase.

Syntax

```
LOWER ( char_expression )
```

Example

This example illustrates the LOWER function:

```
SELECT *
FROM customer
WHERE LOWER (last_name) = 'smith' ;
```

Notes

- The argument to the function must be of type CHARACTER.
- The result is of type CHARACTER.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

SQL compatible
LPAD

Pads the character string corresponding to the first argument on the left with the character string corresponding to the third argument. After the padding, the length of the result is `length`.

**Syntax**

```sql
LPAD ( char_expression , length [ , pad_expression ] )
```

**Example**

This example illustrates two ways to use the LPAD function:

```sql
SELECT LPAD (last_name, 30) FROM customer ;
SELECT LPAD (last_name, 30, '.') FROM customer ;
```

**Notes**

- The first argument to the function must be of type CHARACTER. The second argument to the function must be of type INTEGER. The third argument, if specified, must be of type CHARACTER. If the third argument is not specified, the default value is a string of length 1 containing one blank.

- If $L1$ is the length of the first argument and $L2$ is the value of the second argument:
  - If $L1$ is less than $L2$, the number of characters padded is equal to $L2$ minus $L1$.
  - If $L1$ is equal to $L2$, no characters are padded and the result string is the same as the first argument.
  - If $L1$ is greater than $L2$, the result string is equal to the first argument truncated to the first $L2$ characters.

- The result is of type CHARACTER.

- If the argument expression evaluates to NULL, the result is NULL.

- The `char_expression` and `pad_expression` can contain multi-byte characters. The `length` specifies a number of characters.

**Compatibility**

Progress extension
LTRIM

Removes all the leading characters in `char_expression` that are present in `char_set` and returns the resulting string. The first character in the result is guaranteed not to be in `char_set`. If you do not specify the `char_set` argument, leading blanks are removed.

**Syntax**

```sql
LTRIM ( char_expression [, char_set ] )
```

**Example**

This example illustrates the `LTRIM` function:

```sql
SELECT last_name, LTRIM (addr, ' ') FROM customer ;
```

**Notes**

- The first argument to the function must be of type `CHARACTER`.
- The second argument to the function must be of type `CHARACTER`.
- The result is of type `CHARACTER`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.
- The `char_expression`, the character set specified by `char_set`, and the result can contain multi-byte characters.

**Compatibility**

- ODBC compatible

MAX

Returns the maximum value in a group of values.

**Syntax**

```sql
COUNT ( { [ ALL ] expression } | { DISTINCT column_ref } | * )
```

**Example**

This example illustrates the `MAX` function:

```sql
SELECT order_date, product, MAX (qty) FROM orders GROUP BY order_date, product ;
```
MIN

Notes

- Specifying `DISTINCT` has no effect on the result.
- The argument `column_ref` or `expression` can be of any type.
- The result of the function is of the same data type as that of the argument.
- The result is `NULL` if the result set is empty or contains only `NULL` values.

MIN

Returns the minimum value in a group of values.

Syntax

```
MIN ( { [ ALL ] expression } | { DISTINCT column_ref } )
```

Example

This example illustrates the `MIN` function:

```
SELECT MIN (salary)
  FROM employee
  WHERE deptno = 20 ;
```

MINUTE

Returns the minute value in the argument as a short integer in the range of 0–59.

Syntax

```
MINUTE ( time_expression )
```

Example

This example illustrates the `MINUTE` function:

```
SELECT *
  FROM arrivals
  WHERE MINUTE (in_time) > 10 ;
```

Notes

- The argument to the function must be of type `TIME`.
- The argument must be specified in the format `HH:MI:SS`. 
MOD

Returns the remainder of expression1 divided by expression2.

Syntax

MOD ( expression1 , expression2 )

Example

This example illustrates the MOD function:

SELECT MOD (11, 4) 'Modulus' FROM MYMATH;

Notes

• Both expression1 and expression2 must evaluate to exact numeric data types.
• If expression2 evaluates to zero, MOD returns zero.

Compatibility

ODBC compatible

MONTH

Returns the month in the year specified by the argument as a short integer value in the range of 1–12.

Syntax

MONTH ( date_expression )

Example

This example illustrates the MONTH function:

SELECT * FROM orders WHERE MONTH (order_date) = 6 ;

Notes

• The argument to the function must be of type DATE.
• If date_expression is supplied as a time literal, it can be any of the valid date_literal formats where the day specification (DD) precedes the month specification (MM).
MONTHNAME

- The result is of type SHORT.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

ODBC compatible

MONTHNAME

Returns a character string containing the name of the month (for example, January through December) for the month portion of date_expression. The argument date_expression can be the name of a column, the result of another scalar function, or a date or timestamp literal.

Syntax

MONTHNAME ( date_expression )

Example

In this example, which illustrates the MONTHNAME function, the query returns all rows where the name of the month in the order_date column is equal to 'June':

```
SELECT *
FROM orders
WHERE order_no = 346 and MONTHNAME(order_date)='JUNE';
```

```
ORDER_NO ORDER_DATE  REFERENCE  CUST_NO
------- ---------  --------       -------
346     06/01/2003 87/rd       10002
```

1 record selected

Compatibility

ODBC compatible

MONTHS_BETWEEN

Computes the number of months between two date values corresponding to the first and second arguments.

Syntax

MONTHS_BETWEEN ( date_expression, date_expression )

Example

This example illustrates the MONTHS_BETWEEN function:

```
SELECT MONTHS_BETWEEN (SYSDATE, order_date)
FROM orders
WHERE order_no = 1002 ;
```
Notes

• The first and second arguments to the function must be of type \texttt{DATE}.

• The result is of type \texttt{INTEGER}.

• The result is negative if the date corresponding to the second argument is greater than that corresponding to the first argument.

• If any of the argument expressions evaluates to \texttt{NULL}, the result is \texttt{NULL}.

Compatibility

Progress extension

\section*{NEXT\_DAY}

\textbf{NEXT\_DAY}

Returns the minimum date that is greater than the date corresponding to the first argument where the day of the week is the same as that specified by the second argument.

\textbf{Syntax}

\begin{verbatim}
NEXT\_DAY ( date_expression, day_of_week )
\end{verbatim}

\textbf{Example}

This example illustrates the \texttt{NEXT\_DAY} function:

\begin{verbatim}
SELECT NEXT\_DAY (order_date, 'MONDAY') FROM orders ;
\end{verbatim}

Notes

• The first argument to the function must be of type \texttt{DATE}.

• The second argument to the function must be of type \texttt{CHARACTER}. The result of the second argument must be a valid day of the week ('SUNDAY', 'MONDAY' etc.).

• The result is of type \texttt{DATE}.

• If any of the argument expressions evaluate to \texttt{NULL}, the result is \texttt{NULL}.

Compatibility

Progress extension
NEXTVAL

**NEXTVAL** returns a sequence’s next value. References to NEXTVAL increment the sequence value by the defined increment and return the new value.

Use the following syntax to reference the next value of a sequence:

**Syntax**

```
schema.sequence.NEXTVAL
```

**schema**

Specifies the schema that contains the sequence. To refer to the next value of a sequence in the schema of another user, you must have SELECT object privilege on the sequence.

**sequence**

Specifies the name of the sequence whose next value you want. A statement referencing **NEXTVAL** for a noncycling sequence returns an error after reaching the maximum value.

Use **NEXTVAL** in the:

- **SELECT** list of a **SELECT** statement not contained in a subquery or view
- **SELECT** list of a subquery in an **INSERT** statement
- **VALUES** clause of an **INSERT** statement
- **SET** clause of an **UPDATE** statement

**NEXTVAL** cannot be used in:

- A query of a view
- A **SELECT** statement with a **GROUP BY** clause that references a sequence
- A **SELECT** statement with an **ORDER BY** clause that references a sequence
- A **SELECT** statement that is combined with another **SELECT** statement with the **UNION**, **INTERSECT**, or **MINUS** set operator
- The **WHERE** clause of a **SELECT** or **UPDATE** statement
- The **DEFAULT** value of a column in a **CREATE TABLE** or **ALTER TABLE** statement
- The condition of a **CHECK** constraint
Example

In the following example, the sequence generator increments the customer sequence and uses its value for a new customer inserted into the table `pub.customer`:

```
INSERT INTO pub.customer VALUES (customer_sequence.NEXTVAL, 'USA', 'BackCountry Equipment', 'Sugar Hill Road', '12A', 'Franconia', 'NH', '03242', 'Dan Egan', '603-762-2121', 'Kirsten Ulmner', 10000.00, 500.00, 'net 10', 0, 'contact monthly');
```

NOW

Returns the current date and time as a `TIMESTAMP` value. This function takes no arguments.

Syntax

```
NOW ( )
```

Compatibility

ODBC compatible

NULLIF

Returns a NULL value for `expression1` if it is equal to `expression2`. It is useful for converting values to NULL from applications that use some other representation for missing or unknown data. The NULLIF scalar function is a type of conditional expression.

Syntax

```
NULLIF ( expression1, expression2 )
```

Example

This example uses the NULLIF scalar function to insert a NULL value into an address column if the host-language variable contains a single space character:

```
INSERT INTO employee (add1) VALUES (NULLIF (:address1, ' '));
```

Notes

- This function is not allowed in a GROUP BY clause.
- Arguments to this function cannot be query expressions.
- The NULLIF expression is shorthand notation for a common case that can also be represented in a CASE expression, as shown:

```
CASE
    WHEN expression1 = expression2 THEN NULL
    ELSE expression1
END
```
NVL

Compatibility

SQL compatible

NVL

Returns the value of the first expression if the first expression value is not **NULL**. If the first expression value is **NULL**, the value of the second expression is returned.

Syntax

```
NVL ( expression , expression )
```

Example

This example illustrates the **NVL** function:

```
SELECT salary + NVL (comm, 0) 'TOTAL SALARY' FROM employee;
```

Notes

- The **NVL** function is not ODBC compatible. Use the **IFNULL** function when ODBC-compatible syntax is required.
- The first argument to the function can be of any type.
- The type of the second argument must be compatible with that of the first argument.
- The type of the result is the same as the first argument.

Compatibility

Progress extension

PI

Returns the constant value of **PI** as a floating-point value.

Syntax

```
PI ( )
```

Example

This example illustrates the **PI** function:

```
SELECT PI () FROM SYSPROGRESS.SYSCALCTABLE;
```

Compatibility

ODBC compatible
POWER

Returns expression1 raised to the power of expression2.

Syntax

```
POWER ( expression1 , expression2 )
```

Example

This example illustrates the POWER function, raising '3' to the second power:

```
SELECT POWER ( 3 , 2 ) '3 raised to the 2nd power'
FROM SYSPROGRESS.SYSCALCTABLE;
```

Notes

- expression1 must evaluate to a numeric data type.
- expression2 must evaluate to an exact numeric data type.

PREFIX

Returns the substring of a character string, starting from the position specified by start_pos and ending before the specified character.

Syntax

```
PREFIX ( char_expression , start_pos , char_expression )
```

char_expression

Evaluates to a character string, typically a character-string literal or column name. If the expression evaluates to NULL, PREFIX returns NULL.

start_pos

Evaluates to an integer value. PREFIX searches the string specified in the first argument starting at that position. A value of 1 indicates the first character of the string.

char_expression

Evaluates to a single character. PREFIX returns the substring that ends before that character. If PREFIX does not find the character, it returns the substring beginning at start_pos, to the end of the string. If the expression evaluates to more than one character, PREFIX ignores all but the first character.
PRO_ARR_DESCAPE function

Example

The following example shows one way to use the `PREFIX` function:

```sql
create table prefix_table
    (colstring varchar(20),
     colchar char(1))
;

insert into prefix_table values ('string.with.dots', '.');
insert into prefix_table values ('string-with-dashes', '-');

select colstring, colchar, prefix(colstring, 1, '.') from prefix_table;

<table>
<thead>
<tr>
<th>COLSTRING</th>
<th>COLCHAR</th>
<th>prefix(COLSTRING, 1, .)</th>
</tr>
</thead>
<tbody>
<tr>
<td>string.with.dots</td>
<td>.</td>
<td>string</td>
</tr>
<tr>
<td>string-with-dashes</td>
<td>-</td>
<td>string-with-dashes</td>
</tr>
</tbody>
</table>

select colstring, colchar, prefix(colstring, 1, colchar) from prefix_table;

<table>
<thead>
<tr>
<th>COLSTRING</th>
<th>COLCHAR</th>
<th>prefix(COLSTRING, 1, COLCHAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>string.with.dots</td>
<td>.</td>
<td>string</td>
</tr>
<tr>
<td>string-with-dashes</td>
<td>-</td>
<td>string</td>
</tr>
</tbody>
</table>

select colstring, colchar, prefix(colstring, 1, 'X') from prefix_table;

<table>
<thead>
<tr>
<th>COLSTRING</th>
<th>COLCHAR</th>
<th>prefix(COLSTRING, 1, X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>string.with.dots</td>
<td>.</td>
<td>string.with.dots</td>
</tr>
<tr>
<td>string-with-dashes</td>
<td>-</td>
<td>string-with-dashes</td>
</tr>
</tbody>
</table>
```

Note

Each `char_expression` and the result can contain multi-byte characters. The `start_pos` argument specifies the character position, not a byte position. Character comparisons are case sensitive and are determined by sort weights in the collation table in the database.

Compatibility

Progress extension

PRO_ARR_DESCAPE function

Removes escape characters from a single element of a character array. `PRO_ARR_DESCAPE` scans the `char_element` looking for the separator character ( ; ) or an escape character ( ~ ). The function removes an escape character when it finds any of these constructs:

- Escape character followed by a separator character ( ~; )
- Escape character followed by another escape character ( ~~ )
- Escape character followed by a NULL terminator ( ~\0 )
PRO_ARR_ESCAPE function

Syntax

```sql
PRO_ARR_ESCAPE( 'char_element' ) ;
```

`char_element`

The character representation of an array element, without any leading or trailing separators. Must be data type `NVARCHAR`, `VARCHAR`, or `CHAR`.

Examples

The following example returns the string 'aa;aa':

```sql
PRO_ARR_ESCAPE('aa~;aa') ;
```

The following example returns the string 'aa~aa'. There is no change, since another special character does not follow the escape character:

```sql
PRO_ARR_ESCAPE('aa~aa') ;
```

This example returns the string 'aa~;aa':

```sql
PRO_ARR_ESCAPE('aa~;aa') ;
```

Note

`char_element` should not be the name of an array column, since the column contains true separators that would be destroyed by this function.

PRO_ARR_ESCAPE function

Adds required escape characters to a single element of a character array.

`PRO_ARR_ESCAPE` scans the `char_element` looking for the separator character ( ; ) or an escape character ( ~ ). The function inserts an additional escape character when it finds any of these constructs:

- Escape character followed by a separator character ( ~; )
- Escape character followed by another escape character ( ~~ )
- Escape character followed by a NULL terminator ( ~\0 )

Syntax

```sql
PRO_ARR_ESCAPE( 'char_element' ) ;
```

`char_element`

The character representation of an array element, without any leading or trailing separators. Must be data type `NVARCHAR`, `VARCHAR`, or `CHAR`. 
Examples
The following example returns the string 'aa~;aa':

```
PRO_ARR_ESCAPE('aa;aa') ;
```

The following example returns the string 'aa-aa'. There is no change, since another special character does not follow the escape character:

```
PRO_ARR_ESCAPE('aa-aa') ;
```

This example returns the string 'aa~;aa':

```
PRO_ARR_ESCAPE('aa~;aa') ;
```

Notes
• char_element must be data type NVARCHAR, VARCHAR, or CHAR.
• char_element must not be the name of an array column, since the column contains true separators that would be destroyed by this function.

PRO_ELEMENT function
Extracts one or more elements from an array column and returns the NVARCHAR or VARCHAR string between the specified positions, including any internal separator characters and any internal escape characters.

Syntax
```
PRO_ELEMENT ( 'array_style_expression', start_position, end_position ) ;
```

array_style_expression
A string of data type VARCHAR or CHAR, with a semicolon (;) separating each element of the array.

start_position
The position in the string marking the beginning of the element PRO_ELEMENT is to extract.

end_position
The position in the string marking the end of the element to be extracted.
Examples

The following example returns the string 'bb':

```sql
PRO_ELEMENT('aa;bb;cc', 2, 2) ;
```

The next example returns the string 'aa;bb':

```sql
PRO_ELEMENT('aa;bb;cc', 1, 2) ;
```

This example returns the string 'aa~;aa':

```sql
PRO_ELEMENT('aa~;aa;bb;cc', 1, 1) ;
```

Notes

- The `array_style_expression` must be data type NVARCHAR, VARCHAR, or CHAR.
- The returned string does not include the leading separator of the first element, or the trailing separator ( ; ) of the last element.
- Even if you are extracting only one element, the escape characters are included in the result.
- You must invoke `PRO_ARR_DESCAPE` to remove any escape characters.
- If the expression evaluates to NULL, the result is NULL.

QUARTER

Returns the quarter in the year specified by the argument as a short integer value in the range of 1–4.

Syntax

```sql
QUARTER ( date_expression )
```

Example

In this example, which illustrates the QUARTER function, the query requests all rows in the orders table where the `order_date` is in the third quarter of the year:

```sql
SELECT *
FROM orders
WHERE QUARTER (order_date) = 3 ;
```

Notes

- The argument to the function must be of type DATE.
- If `date_expression` is supplied as a date literal, it can be any of the valid `date_literal` formats where the day specification (DD) precedes the month specification (MM).
RADIANS

- The result is of type SHORT.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

ODBC compatible

RADIANS

Returns the number of radians in an angle specified in degrees by expression.

Syntax

RADIANS ( expression )

Example

This example illustrates the RADIANS function:

```
SELECT RADIANS(180) 'Radians in 180 degrees' FROM SYSPROGRESS.SYSCALCTABLE;
```

Notes

- expression specifies an angle in degrees.
- expression must evaluate to a numeric data type.

Compatibility

ODBC compatible

RAND

Returns a randomly generated number, using expression as an optional seed value.

Syntax

RAND ( [ expression ] )

Example

This example illustrates the RAND function, supplying an optional seed value of ’3’:

```
SELECT RAND(3) 'Random number using 3 as seed value'
               FROM MYMATH;
```

Note

expression must be an INT (32-bit) data type.

Compatibility

ODBC compatible
Returns a character string composed of string_exp repeated count times.

Syntax

```sql
REPEAT ( string_exp , count )
```

Example

The following example shows how to use the REPEAT function:

```sql
SELECT REPEAT(fld1,3) FROM test100WHERE fld1 = 'Afghanistan';
```

```
REPEAT(FLD1,3)
-------------
AfghanistanAfghanistanAfghanistan
1 record selected
```

Notes

- The string_exp can be of the type fixed-length or variable-length CHARACTER.
- The count can be of type INTEGER, SMALLINT, or TINYINT.
- If any of the arguments of the expression evaluates to a NULL, the result is NULL.
- If the count is negative or zero, the result evaluates to NULL.
- string_exp and the result can contain multi-byte characters.

Compatibility

ODBC compatible

Replaces all occurrences of string_exp2 in string_exp1 with string_exp3.

Syntax

```sql
REPLACE ( string_exp1 , string_exp2 , string_exp3 )
```

Example

This example illustrates the REPLACE function, replacing the letters 'mi' in the last_name 'Smith' with the letters 'moo':

```sql
SELECT REPLACE ( last_name,'mi','moo' ) FROM customer WHERE last_name = 'Smith';
```

```
REPLACE(LAST_NAME,MI,MOO)
--------------
Smooth
1 record selected
```
Notes

- `string_exp` can be fixed-length or variable-length CHARACTER data types.
- If any of the arguments of the expression evaluates to NULL, the result is NULL.
- If the replacement string is not found in the search string, it returns the original string.
- Each occurrence of `string_exp` and the result can contain multi-byte characters. Character comparisons are case sensitive and are determined by sort weights in the collation table in the database.

Compatibility

ODBC compatible
ROUND

Returns the rounded value of a numeric expression.

Syntax

\[
\text{ROUND ( num_expression [ , rounding_factor ] ) ;}
\]

Example

This example illustrates four calls to the \texttt{ROUND} function:

\[
\begin{align*}
\text{-- rounding_factor 2 returns 2953861.83} \\
\text{ROUND ( 2953861.8320, 2 )} \\
\text{-- rounding_factor -2 returns 2953900.00} \\
\text{ROUND ( 2953861.8320, -2 )} \\
\text{-- rounding_factor 0 returns 2953862.00} \\
\text{ROUND ( 2953861.8320, 0 )} \\
\text{-- No rounding_factor argument also returns 2953862.00} \\
\text{ROUND ( 2953861.8320 )}
\end{align*}
\]

In each case the \texttt{num_expression} is 2953861.8320. In the first call the \texttt{rounding_factor} is 2, in the second call the \texttt{rounding_factor} is -2, in the third call the \texttt{rounding_factor} is 0, and in the fourth call no \texttt{rounding_factor} is specified.

Notes

- \texttt{num_expression} must be numeric or must be convertible to numeric.
- \texttt{num_expression} must be one of these supported data types:
  - \texttt{INTEGER}
  - \texttt{TINYINT}
  - \texttt{SMALLINT}
  - \texttt{NUMBER}
  - \texttt{FLOAT}
  - \texttt{DOUBLE PRECISION}
- If the data type of \texttt{num_expression} is not a supported type, \texttt{ROUND} returns an error message.
- The \texttt{num_expression} is rounded to the next higher digit when:
  - The digit before a negative \texttt{rounding_factor} is 5 or greater
  - The digit after a positive \texttt{rounding_factor} is 5 or greater
• The num_expression is rounded to the next lower digit when:
  – The digit before a negative rounding_factor is 4 or less
  – The digit after a positive rounding_factor is 4 or less

• rounding_factor is an integer between –32 and +32 inclusive, and indicates the digit position to which you want to round num_expression. Figure 1 illustrates how the digit positions are numbered. In the figure, the num_expression is 2953861.8320.

```
2953861.8320

Position -2
Position 0
Position 2
```

Figure 1: ROUND digit positions

• If you do not specify a rounding_factor, the function rounds num_expression to digit 0 (the ones place).

• To round to the right of the decimal point, specify a positive rounding_factor.

• To round to the left of the decimal, specify a negative rounding_factor.

Compatibility

Progress extension

---

**ROWID**

Returns the row identifier of the current row in a table. This function takes no arguments. The ROWID of a row is determined when the row is inserted into the table. Once assigned, the ROWID remains the same for the row until the row is deleted. At any given time, each row in a table is uniquely identified by its ROWID. Using its ROWID is the most efficient way of selecting the row.

**Syntax**

```sql
ROWID
```
Example

This example illustrates the **ROWID** function, returning all columns from the row in the *customers* table where the **ROWID** = '10':

```sql
SELECT *
FROM customers
WHERE ROWID = '10';
```

Note

The **ROWID** function returns a string of up to 19 characters in length.

Compatibility

Progress extension

---

**RPAD**

Pads the character string corresponding to the first argument on the right with the character string corresponding to the third argument. After the padding, the length of the result is equal to the value of the second argument *length*.

**Syntax**

```sql
RPAD ( char_expression, length [ , pad_expression ] )
```

**Example**

This example illustrates two ways to use the **RPAD** function:

```sql
SELECT RPAD (last_name, 30)
FROM customer ;

SELECT RPAD (last_name, 30, '.')
FROM customer ;
```

**Notes**

- The first argument to the function must be of type **CHARACTER**. The second argument to the function must be of type **INTEGER**. The third argument, if specified, must be of type **CHARACTER**. If the third argument is not specified, the default value is a string of length 1 containing one blank.

- If $L_1$ is the length of the first argument and $L_2$ is the value of the second argument:
  - If $L_1$ is less than $L_2$, the number of characters padded is equal to $L_2$ minus $L_1$.
  - If $L_1$ is equal to $L_2$, no characters are padded and the result string is the same as the first argument.
  - If $L_1$ is greater than $L_2$, the result string is equal to the first argument truncated to the first $L_2$ characters.
RTRIM

• The result is of type CHARACTER.

• If the argument expression evaluates to NULL, the result is NULL.

• char_expression and pad_expression can contain multi-byte characters. length represents the number of characters in the result.

Compatibility

Progress extension

RTRIM

Removes all the trailing characters in char_expression that are present in char_set and returns the resultant string. The last character in the result is guaranteed not to be in char_set. If you do not specify a char_set, trailing blanks are removed.

Syntax

RTRIM ( char_expression [ , char_set ] )

Example

This example illustrates the RTRIM function:

```sql
SELECT RPAD ( RTRIM (addr, ' '), 30, '.')
FROM customer ;
```

Notes

• The first argument to the function must be of type CHARACTER.

• The second argument to the function must be of type CHARACTER.

• The result is of type CHARACTER.

• If the argument expression evaluates to NULL, the result is NULL.

• The char_expression, the character set specified by char_set, and the result can contain multi-byte characters. Character comparisons are case sensitive and are determined by the collation table in the database.

Compatibility

ODBC compatible
SECOND

Returns the seconds in the argument as a short integer value in the range of 0–59.

Syntax

```
SECOND ( time_expression )
```

Example

This example illustrates the SECOND function, requesting all columns from rows in the arrivals table where the in_time column is less than or equal to '40':

```
SELECT * FROM arrivals WHERE SECOND (in_time) <= 40 ;
```

Notes

- The argument to the function must be of type TIME.
- The argument must be specified in the format HH:MI:SS.
- The result is of type SHORT.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

ODBC compatible

SIGN

Returns 1 if expression is positive, -1 if expression is negative, or zero if expression is zero.

Syntax

```
SIGN ( expression )
```

Example

This example illustrates the SIGN function:

```
SELECT SIGN(-14) 'Sign' FROM MYMATH;
```

Note

expression must evaluate to a NUMERIC data type.

Compatibility

ODBC compatible
SIN

Returns the sine of expression.

Syntax

```
SIN ( expression )
```

Example

This example illustrates the SIN trigonometric function:

```
select sin(45 * pi()/180) 'Sine of 45 degrees' from MYMATH;
```

Notes

- SIN takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.
-expression specifies an angle in radians.
- expression must evaluate to an approximate numeric data type.
- To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

Compatibility

ODBC compatible

SQRT

Returns the square root of expression.

Syntax

```
SQRT ( expression )
```

Example

This example illustrates the SQRT function, requesting the square root of the value ‘28’:

```
SELECT SQRT(28) 'square root of 28' FROM MYMATH;
```
Notes

- The value of expression must be positive.
- expression must evaluate to an approximate numeric data type.

Compatibility

ODBC compatible

SUBSTR

Returns the substring of the character string corresponding to the first argument starting at start_pos and length characters long. If the third argument length is not specified, the substring starting at start_pos up to the end of char_expression is returned.

Syntax

```
SUBSTR ( char_expression, start_pos [ , length ] )
```

Example

This example illustrates the SUBSTR function:

```
SELECT last_name, '(', SUBSTR (phone, 1, 3) , ')
        , SUBSTR (phone, 4, 3), '-',
        SUBSTR (phone, 7, 4)
FROM customer ;
```

Notes

- The first argument must be of type CHARACTER. It can be any meaningful character value (for example, a literal expression, database column, or parameter).
- The second argument must be of type INTEGER. It indicates the starting position from which the substring result is extracted.
- The third argument, if specified, must be of type INTEGER. It indicates the number of characters the substring function will extract.
- The values for specifying position in the character string start from 1. The first character in a string is at position 1, the second character is at position 2, and so on.
- The result is of type CHARACTER.
- If any of the argument expressions evaluate to NULL, the result is NULL.
- char_expression and the result can contain multi-byte characters.
- If the value of start_pos is:
  - Smaller than 0, the function returns a “Bad argument” error
  - Bigger than the actual length of the string value, the function returns an empty zero length substring
• If the value of \textit{length} is:
  – Smaller than 0, the function returns a “Bad argument” error.
  – Bigger than the actual length of the substring (from \textit{start_pos} to the end of the literal), the function returns the substring from \textit{start_pos} to the end of the literal.
  – Bigger than the actual length of the substring (from \textit{start_pos} up to the end of the column’s row data), the function returns the substring from the \textit{start_pos} to the end of the column’s row data. The function returns this, \textbf{even when the substring result exceeds the column’s SQL width}.
  – Bigger than 0 and the column’s row data exceeds the column’s SQL width, the function returns the substring.

• If \textit{length} is not specified, the function returns the substring from \textit{start_pos} to the end of the literal.

\textbf{Note} The function returns the end of the column’s row data \textbf{if the length of the substring is not bigger than the column’s SQL width}. Otherwise, the function returns the substring truncated to the column’s SQL width.

\textbf{Compatibility} Progress extension

\textbf{SUBSTRING (ODBC compatible)}

Returns the substring of the character string corresponding to the first argument starting at \textit{start_pos} and \textit{length} characters long. If the third argument \textit{length} is not specified, the substring starting at \textit{start_pos} up to the end of \textit{char_expression} is returned.

\textbf{Syntax}

\begin{verbatim}
SUBSTRING ( char_expression, start_pos [ , length ] )
\end{verbatim}

\textbf{Example} This example illustrates the \texttt{SUBSTRING} function:

\begin{verbatim}
SELECT last_name, '{', SUBSTRING (phone, 1, 3) , '}', SUBSTRING (phone, 4, 3), '-', SUBSTRING (phone, 7, 4)
FROM customer ;
\end{verbatim}

\textbf{Notes}

• The first argument must be of type \texttt{CHARACTER}. It can be any meaningful character value (for example, a literal expression, database column, or parameter).

• The second argument must be of type \texttt{INTEGER}. It indicates the starting position from which the substring result is extracted.
• The third argument, if specified, must be of type INTEGER. It indicates the number of characters the substring function will extract.

• The values for specifying position in the character string start from 1. The first character in a string is at position 1, the second character is at position 2, and so on.

• The result is of type CHARACTER.

• If any of the argument expressions evaluate to NULL, the result is NULL.

• char_expression and the result can contain multi-byte characters.

• If the value of start_pos is:
  – Smaller than 0, the function returns a “Bad argument” error
  – Bigger than the actual length of the string value, the function returns an empty zero length substring

• If the value of length is:
  – Smaller than 0, the function returns a “Bad argument” error.
  – Bigger than the actual length of the substring (from start_pos to the end of the literal), the function returns the substring from start_pos to the end of the literal.
  – Bigger than the actual length of the substring (from start_pos up to the end of the column’s row data), the function returns the substring from the start_pos to the end of the column’s row data. The function returns this, even when the substring result exceeds the column’s SQL width.
  – Bigger than 0 and the column’s row data exceeds the column’s SQL width, the function returns the substring.

• If length is not specified, the function returns the substring from start_pos to the end of the literal.

• The function returns the end of the column’s row data if the length of the substring is not bigger than the column’s SQL width. Otherwise, the function returns the substring truncated to the column’s SQL width.

Compatibility

ODBC compatible
SUFFIX

Returns the substring of a character string starting after the position specified by `start_pos` and the second `char_expression`, to the end of the string.

Syntax

```
SUFFIX (char_expression , start_pos , char_expression )
```

**char_expression**

Evaluates to a character string, typically a character-string literal or column name. If the expression evaluates to `NULL`, `SUFFIX` returns `NULL`.

**start_pos**

Evaluates to an integer value. `SUFFIX` searches the string specified in the first argument starting at that position. A value of 1 indicates the first character of the string.

**char_expression**

Evaluates to a single character. `SUFFIX` returns the substring that begins with that character. If `SUFFIX` does not find the character after `start_pos`, it returns `NULL`. If the expression evaluates to more than one character, `SUFFIX` ignores all but the first character.

**Example**

This example illustrates two ways to use the `SUFFIX` function:

```
SELECT C1, C2, SUFFIX(C1, 6, '.') FROM T1;
C1     C2 SUFFIX(C1,6,.
--     -- -------------
test.pref  .
pref.test  s
2 records selected

SELECT C1, C2, SUFFIX(C1, 1, C2) FROM T1;
C1     C2 SUFFIX(C1,1,C
--     -- -------------
test.pref  . pref
pref.test  s t
2 records selected
```

**Note**

Each `char_expression` and the result can contain multi-byte characters. The `start_pos` argument specifies the character position, not a byte position. Character comparisons are case sensitive and are determined by sort weights in the collation table in the database.

**Compatibility**

Progress extension
SUM

Returns the sum of the values in a group. The keyword DISTINCT specifies that the duplicate values are to be eliminated before computing the sum.

Syntax

```
SUM ( { [ALL] expression } | { DISTINCT column_ref } )
```

Example

This example illustrates the SUM function:

```
SELECT SUM (amount)
    FROM orders
    WHERE order_date = SYSDATE;
```

Notes

- The argument column_ref or expression can be of any type.
- The result of the function is of the same data type as that of the argument except that the result is of type INTEGER when the argument is of type SMALLINT or TINYINT.
- The result can have a NULL value.

SYSDATE

Returns the current date as a DATE value. This function takes no arguments, and the trailing parentheses are optional.

Syntax

```
SYSDATE [ ( ) ]
```

Example

This example illustrates the SYSDATE function, inserting a new row into the objects table, setting the create_date column to the value of the current date:

```
INSERT INTO objects (object_owner, object_id, create_date)
    VALUES (USER, 1001, SYSDATE);
```

Compatibility

Progress extension

SYSTIME

Returns the current time as a TIME value to the nearest second. This function takes no arguments, and the trailing parentheses are optional. SQL statements can refer to SYSTIME anywhere they can refer to a TIME expression.
SYSTIMESTAMP

Syntax

```
SYSTIME [ ( ) ]
```

Example

This example illustrates the SYSTIME function, inserting a new row into the objects table, setting the create_time column to the value of the current time:

```
INSERT INTO objects (object_owner, object_id, create_time)
VALUES (USER, 1001, SYSTIME) ;
```

Compatibility

Progress extension

SYSTIMESTAMP

Returns the current date and time as a TIMESTAMP value. This function takes no arguments, and the trailing parentheses are optional.

Syntax

```
SYSTIMESTAMP [ ( ) ]
```

Example

This example illustrates different formats for SYSDATE, SYSTIME, and SYSTIMESTAMP:

```
SELECT SYSDATE FROM test;
SYSDATE
-------
09/13/2003
1 record selected

SELECT SYSTIME FROM test;
SYSTIME
-------
14:44:07:000
1 record selected

SELECT SYSTIMESTAMP FROM test;
SYSTIMESTAMP
----------
2003-09-13 14:44:15:000
1 record selected
```

Compatibility

Progress extension
TAN

Returns the tangent of expression.

Syntax

TAN ( expression )

Example

The following example shows how to use the TAN function:

```
select tan(45 * pi()/180) 'Tangent of 45 degrees' from MYMATH;
TANGENT OF 45 DEGREES
---------------
1.000000000000000
1 record selected
```

Notes

- TAN takes an angle (expression) and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.
- expression specifies an angle in radians.
- expression must evaluate to an approximate numeric data type.
- To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

Compatibility

ODBC compatible

TO_CHAR

Converts the given expression to character form and returns the result. The primary use for TO_CHAR is to format the output of date-time expressions through the format_string argument.

Syntax

```
TO_CHAR ( expression [, format_string ] )
```

expression

Converts to character form. It must evaluate to a value of the date or time data type to use the format_string.
TO_DATE

format_string

Specifies the format of the output. SQL ignores the format string if the expression argument does not evaluate to a date or time.

Notes

• The first argument to the function can be of any type.
• The second argument, if specified, must be of type CHARACTER.
• The result is of type CHARACTER.
• The format argument can be used only when the type of the first argument is DATE.
• If any of the argument expressions evaluates to NULL, the result is NULL.

Compatibility

Progress extension

TO_DATE

Converts the given date literal to a date value.

Syntax

TO_DATE ( date_literal )

Example

This example illustrates the TO_DATE function, returning all columns from rows in the orders table where the order_date column is earlier or equal to the date '12/31/2003':

```sql
SELECT * 
FROM orders 
WHERE order_date <= TO_DATE ('12/31/2003') ;
```

Notes

• The result is of type DATE.
• Supply the date literal in any valid format.

Compatibility

Progress extension
TO_NUMBER

Converts the given character expression to a number value.

**Syntax**

\[ \text{TO\_NUMBER (char\_expression)} \]

**Example**

This example illustrates the `TO_NUMBER` function and the `SUBSTR` function:

```sql
SELECT * 
FROM customer 
WHERE TO_NUMBER (SUBSTR(phone, 1, 3)) = 603 ;
```

**Notes**

- The argument to the function must be of type `CHARACTER`.
- The result is of type `NUMERIC`.
- If any of the argument expressions evaluates to `NULL`, the result is `NULL`.

**Compatibility**

Progress extension

TO_TIME

Converts the given time literal to a time value.

**Syntax**

\[ \text{TO\_TIME (time\_literal)} \]

**Example**

The following example shows how to use the `TO_DATE` and the `TO_TIME` functions:

```sql
SELECT * FROM orders 
WHERE order_date < TO_DATE ('05/15/2003') 
AND order_time < TO_TIME ('12:00:00') ;
```

**Notes**

- The result is of type `TIME`.
- Supply the time literal in any valid format.

**Compatibility**

Progress extension
**TO_TIMESTAMP**

Converts the given timestamp literal to a timestamp value.

**Syntax**

```sql
TO_TIMESTAMP ( timestamp_lit )
```

**Example**

The following example shows how to use the `TO_TIMESTAMP` function:

```sql
SELECT * FROM DTEST WHERE C3 = TO_TIMESTAMP('4/18/03 10:41:19')
```

**Notes**

- The result is of type `TIME`.
- Supply the timestamp literal in any valid format.

**Compatibility**

Progress Extension

---

**TRANSLATE**

Translates each character in `char_expression` that is in `from_set` to the corresponding character in `to_set`. The translated character string is returned as the result.

**Syntax**

```sql
TRANSLATE ( char_expression , from_set , to_set )
```

**Example**

This example substitutes underscores for spaces in customer names:

```sql
SELECT TRANSLATE (customer_name, ' ', '_')
  "TRANSLATE Example" from customers;
```

```
TRANSLATE EXAMPLE
-----------------
Sports_Cars_Inc.__________________________________
Mighty_Bulldozer_Inc._____________________________
Ship_Shapers_Inc._________________________________
Tower_Construction_Inc.___________________________
Chemical_Construction_Inc.________________________
Aerospace_Enterprises_Inc.________________________
Medical_Enterprises_Inc.__________________________
Rail_Builders_Inc.________________________________
Luxury_Cars_Inc.__________________________________
Office_Furniture_Inc._____________________________
10 records selected
```
Notes

- `char_expression`, `from_set`, and `to_set` can be any character expression.
- For each character in `char_expression`, `TRANSLATE` checks for the same character in `from_set`.
- If it is in `from_set`, `TRANSLATE` translates it to the corresponding character in `to_set` (if the character is the \(n\)th character in `from_set`, the \(n\)th character in `to_set`).
- If the character is not in `from_set`, `TRANSLATE` does not change it.
- If `from_set` is longer than `to_set`, `TRANSLATE` does not change trailing characters in `from_set` that do not have a corresponding character in `to_set`.
- If either `from_set` or `to_set` is `NULL`, `TRANSLATE` does nothing.

Compatibility

Progress extension

UCASE

Returns the result of the argument character expression after converting all the characters to uppercase. `UCASE` is identical to `UPPER`, but provides ODBC-compatible syntax.

Syntax

```
UCASE ( char_expression )
```

Example

This example illustrates the `UCASE` function, returning columns from rows in the customer table where the `last_name` column, after being converted to uppercase, is equal to 'SMITH':

```
SELECT *
FROM customer
WHERE UCASE ( last_name ) = 'SMITH' ;
```

Notes

- The argument to the function must be of type `CHARACTER`.
- The result is of type `CHARACTER`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.
- A `char_expression` and the result can contain multi-byte characters. The uppercase conversion is determined by the case table in the `convmap` file. The default case table is `BASIC`.

Compatibility

ODBC compatible
**UPPER**

Returns the result of the argument character expression after converting all the characters to uppercase.

**Syntax**

```
UPPER ( char_expression )
```

**Example**

This example illustrates the `UPPER` function, returning columns from rows in the `customer` table where the `last_name` column, after being converted to uppercase, is equal to 'SMITH':

```
SELECT *
FROM customer
WHERE UPPER (last_name) = 'SMITH';
```

**Notes**

- The argument to the function must be of type `CHARACTER`.
- The result is of type `CHARACTER`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.
- A `char_expression` and the result can contain multi-byte characters. The uppercase conversion is determined by the case table in the `convmap` file. The default case table is `BASIC`.

**Compatibility**

SQL compatible

---

**USER**

Returns a character-string identifier for the user of the current transaction, as determined by the host operating system. This function takes no arguments, and the trailing parentheses are optional.

**Syntax**

```
USER [ ( ) ]
```

**Note**

SQL statements can refer to `USER` anywhere they can refer to a character string expression.

**Compatibility**

ODBC compatible
WEEK

Returns the week of the year as a short integer value in the range of 1–53.

Syntax

```
WEEK ( time_expression )
```

Example

The query returns all columns from rows in the orders table where the `order_date` is in the fifth week of the year. This example illustrates the `WEEK` function:

```
SELECT *
FROM orders
WHERE WEEK (order_date) = 5;
```

Notes

- The argument to the function must be of type `DATE`.
- If `date_expression` is supplied as a date literal, it can be any of the valid `date_literal` formats where the day specification (DD) precedes the month specification (MM).
- The result is of type `SHORT`.
- If the argument expression evaluates to `NULL`, the result is `NULL`.

Compatibility

ODBC compatible

YEAR

Returns the year as a short integer value in the range of 0–9999.

Syntax

```
YEAR ( date_expression )
```

Example

The query returns all columns in rows in the `orders` table where the year in the `order_date` column is equal to ‘2003’. This example illustrates the `YEAR` function:

```
SELECT *
FROM orders
WHERE YEAR (order_date) = 2003;
```
Notes

- The argument to the function must be of type DATE.
- If date_expression is supplied as a date literal, it can be any of the valid date_literal formats where the day specification (DD) precedes the month specification (MM).
- The result is of type SHORT.
- If the argument expression evaluates to NULL, the result is NULL.

Compatibility

ODBC compatible
OpenEdge SQL Reserved Words

This section provides a list of words that have special syntactic meaning to SQL and cannot be used as identifiers for constants, variables, cursors, types, tables, records, subprograms, or packages.

OpenEdge SQL reserved words

Reserved words are keywords. You can use keywords as identifiers in SQL statements only if you delimit them with double quotation marks. If you use keywords without delimiting them, the statement generates one of the following errors:

- error(-20003): Syntax error
- error(-20049): Keyword used for a name

Table 1 provides a list of OpenEdge SQL reserved words.

<table>
<thead>
<tr>
<th>Table 1: OpenEdge SQL reserved words (1 of 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
</tr>
<tr>
<td>AFTER</td>
</tr>
<tr>
<td>ANY</td>
</tr>
<tr>
<td>ARRAY</td>
</tr>
<tr>
<td>ASIN</td>
</tr>
<tr>
<td>AUDIT_ADMIN</td>
</tr>
<tr>
<td>AVG</td>
</tr>
<tr>
<td>BINARY</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>CALL</td>
</tr>
<tr>
<td>CEILING</td>
</tr>
<tr>
<td>CHARTORONID</td>
</tr>
<tr>
<td>CLOB</td>
</tr>
<tr>
<td>COLGROUP</td>
</tr>
<tr>
<td>COMpress</td>
</tr>
<tr>
<td>CONTAINS</td>
</tr>
<tr>
<td>CREATE</td>
</tr>
<tr>
<td>CURRENT_USER</td>
</tr>
<tr>
<td>CYCLE</td>
</tr>
<tr>
<td>DAYNAME</td>
</tr>
<tr>
<td>DBA</td>
</tr>
<tr>
<td>DECLARE</td>
</tr>
<tr>
<td>DEGREES</td>
</tr>
<tr>
<td>DISCONNECT</td>
</tr>
<tr>
<td>EACH</td>
</tr>
<tr>
<td>EVENT_GROUP</td>
</tr>
<tr>
<td>EXP</td>
</tr>
<tr>
<td>FLOAT</td>
</tr>
<tr>
<td>FROM</td>
</tr>
<tr>
<td>HASH</td>
</tr>
<tr>
<td>IFNULL</td>
</tr>
<tr>
<td>INDICATOR</td>
</tr>
<tr>
<td>INSERT</td>
</tr>
<tr>
<td>INTERSECT</td>
</tr>
<tr>
<td>JOIN</td>
</tr>
<tr>
<td>LEAST</td>
</tr>
<tr>
<td>LIKE</td>
</tr>
<tr>
<td>LOG</td>
</tr>
<tr>
<td>LPAD</td>
</tr>
<tr>
<td>MAX</td>
</tr>
<tr>
<td>MINUS</td>
</tr>
<tr>
<td>OpenEdge SQL reserved words (3 of 4)</td>
</tr>
<tr>
<td>-------------------------------------</td>
</tr>
<tr>
<td><strong>MODE</strong></td>
</tr>
<tr>
<td><strong>MONTHS_BETWEEN</strong></td>
</tr>
<tr>
<td><strong>NEWROW</strong></td>
</tr>
<tr>
<td><strong>NOCYCLE</strong></td>
</tr>
<tr>
<td><strong>NOMINVALUE</strong></td>
</tr>
<tr>
<td><strong>NULL</strong></td>
</tr>
<tr>
<td><strong>NUMERIC</strong></td>
</tr>
<tr>
<td><strong>ODBC_CONVERT</strong></td>
</tr>
<tr>
<td><strong>ON</strong></td>
</tr>
<tr>
<td><strong>OUT</strong></td>
</tr>
<tr>
<td><strong>POWER</strong></td>
</tr>
<tr>
<td><strong>PRIVILEGES</strong></td>
</tr>
<tr>
<td><strong>PRO_ARRAY_ELEMENT</strong></td>
</tr>
<tr>
<td><strong>PRO_CAN_CREATE</strong></td>
</tr>
<tr>
<td><strong>PRO_CAN_READ</strong></td>
</tr>
<tr>
<td><strong>PRO_CLIENT_FILE_TRIGGER</strong></td>
</tr>
<tr>
<td><strong>PRO_CREATE</strong></td>
</tr>
<tr>
<td><strong>PRO_DESCRIPTION</strong></td>
</tr>
<tr>
<td><strong>PRO_ENABLE_LARGE_KEYS</strong></td>
</tr>
<tr>
<td><strong>PRO_HELP</strong></td>
</tr>
<tr>
<td><strong>PRO_NAME</strong></td>
</tr>
<tr>
<td><strong>PRO_PROCNAME</strong></td>
</tr>
<tr>
<td><strong>PRO_SA_COL_LABEL</strong></td>
</tr>
<tr>
<td><strong>PRO_SA_LABEL</strong></td>
</tr>
<tr>
<td><strong>PRO_STATUS</strong></td>
</tr>
<tr>
<td><strong>PRO_VALMSG</strong></td>
</tr>
<tr>
<td><strong>PUBLIC</strong></td>
</tr>
<tr>
<td><strong>RAND</strong></td>
</tr>
<tr>
<td><strong>RECID</strong></td>
</tr>
<tr>
<td><strong>REPEAT</strong></td>
</tr>
<tr>
<td><strong>RESULT</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>ROLLBACK</td>
</tr>
<tr>
<td>ROWIDTOCHAR</td>
</tr>
<tr>
<td>SCHEMA</td>
</tr>
<tr>
<td>SEQUENCE</td>
</tr>
<tr>
<td>SHARE</td>
</tr>
<tr>
<td>SIN</td>
</tr>
<tr>
<td>SOUNDEX</td>
</tr>
<tr>
<td>SQL_BIT</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
</tr>
<tr>
<td>SQL_TIME</td>
</tr>
<tr>
<td>SQL_TSI_FRACTSECOND</td>
</tr>
<tr>
<td>SQL_TSI_QUARTER</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
</tr>
<tr>
<td>STATISTICS</td>
</tr>
<tr>
<td>SUBSTR</td>
</tr>
<tr>
<td>SUSER_NAME</td>
</tr>
<tr>
<td>SYSTIMESTAMP</td>
</tr>
<tr>
<td>TAN</td>
</tr>
<tr>
<td>TIMESTAMPADD</td>
</tr>
<tr>
<td>TOP</td>
</tr>
<tr>
<td>TO_TIME</td>
</tr>
<tr>
<td>TRANSLATE</td>
</tr>
<tr>
<td>UID</td>
</tr>
<tr>
<td>UPPER</td>
</tr>
<tr>
<td>USING</td>
</tr>
<tr>
<td>VARCHAR</td>
</tr>
<tr>
<td>WEEK</td>
</tr>
<tr>
<td>WORK</td>
</tr>
</tbody>
</table>
OpenEdge SQL Error Messages

This section provides information on error messages generated by the various components of OpenEdge® SQL. Error message information includes:

- Error code
- SQLSTATE value
- Class condition
- Subclass message

Overview

In addition to the OpenEdge-specific error codes, error conditions have an associated SQLSTATE value. SQLSTATE is a five-character status parameter whose value indicates the condition status returned by the most recent SQL statement. The first two characters of the SQLSTATE value specify the class code and the last three characters specify the subclass code:

- Class codes of a–h and 0–4 are reserved by the SQL standard. For those class codes only, subclass codes of a–h and 0–4 are also reserved by the standard.
- Subclasses S and T and class IM are reserved by the ODBC standard.
- Class codes of i–z and 5–9 are specific to database implementations such as OpenEdge SQL. All subclass codes in those classes are implementation defined except as noted for ODBC.
# Error codes, SQLSTATE values, and messages

Table 1 is a list of OpenEdge SQL error messages, ordered by error code number. The table shows the corresponding SQLSTATE value for each message.

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQLSTATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>00000</td>
<td>Successful completion</td>
<td>***status okay.</td>
</tr>
<tr>
<td>100L</td>
<td>02000</td>
<td>No data</td>
<td>**sql not found.</td>
</tr>
<tr>
<td>10002</td>
<td>22503</td>
<td>Data exception</td>
<td>Tuple not found for the Specified TID.</td>
</tr>
<tr>
<td>10012</td>
<td>N0N12</td>
<td>Flag</td>
<td>ETPL_SCAN_EOP.</td>
</tr>
<tr>
<td>10013</td>
<td>22914</td>
<td>Data Exception</td>
<td>No more records to be fetched.</td>
</tr>
<tr>
<td>10100</td>
<td>2150b</td>
<td>Cardinality violation</td>
<td>Too many fields exist.</td>
</tr>
<tr>
<td>10101</td>
<td>70701</td>
<td>OpenEdge/SQL MM error</td>
<td>No more records exist.</td>
</tr>
<tr>
<td>10102</td>
<td>2350i</td>
<td>Integrity constraint</td>
<td>Duplicate primary/index key value.</td>
</tr>
<tr>
<td>10104</td>
<td>M0M06</td>
<td>OpenEdge/SQL rss error</td>
<td>Specified index method is not supported.</td>
</tr>
<tr>
<td>10107</td>
<td>N0N07</td>
<td>Flag</td>
<td>EIX_SCAN_EOP flag is set.</td>
</tr>
<tr>
<td>10108</td>
<td>50903</td>
<td>OpenEdge/SQL rds error</td>
<td>Duplicate record specified.</td>
</tr>
<tr>
<td>10301</td>
<td>M0901</td>
<td>OpenEdge/SQL rss error</td>
<td>Table is locked and LCK_NOWAIT.</td>
</tr>
<tr>
<td>10400</td>
<td>22501</td>
<td>Data exception</td>
<td>Invalid file size for alter log statement.</td>
</tr>
<tr>
<td>10920</td>
<td>22521</td>
<td>Data exception</td>
<td>Already existing value specified.</td>
</tr>
<tr>
<td>11100</td>
<td>50901</td>
<td>OpenEdge/SQL rds error</td>
<td>Invalid transaction id.</td>
</tr>
<tr>
<td>11102</td>
<td>50903</td>
<td>OpenEdge/SQL rds error</td>
<td>TDS area specified is not found.</td>
</tr>
<tr>
<td>11103</td>
<td>50504</td>
<td>OpenEdge/SQL rds error</td>
<td>TDS not found for binding.</td>
</tr>
<tr>
<td>11104</td>
<td>50505</td>
<td>OpenEdge/SQL rds error</td>
<td>Transaction aborted.</td>
</tr>
<tr>
<td>11105</td>
<td>50506</td>
<td>OpenEdge/SQL rds error</td>
<td>Transaction error.</td>
</tr>
<tr>
<td>11109</td>
<td>50510</td>
<td>OpenEdge/SQL rds error</td>
<td>Invalid transaction handle.</td>
</tr>
<tr>
<td>11111</td>
<td>50912</td>
<td>OpenEdge/SQL rds error</td>
<td>Invalid isolation level.</td>
</tr>
</tbody>
</table>
Table 1: OpenEdge SQL error codes and messages

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>11300</td>
<td>M0M00</td>
<td>OpenEdge/SQL rss error</td>
<td>Specified INFO type is not supported.</td>
</tr>
<tr>
<td>11301</td>
<td>M0M01</td>
<td>OpenEdge/SQL rss error</td>
<td>Specified index type is not supported.</td>
</tr>
<tr>
<td>16001</td>
<td>22701</td>
<td>Data exception</td>
<td>MM– No data block.</td>
</tr>
<tr>
<td>16002</td>
<td>70702</td>
<td>OpenEdge SQLMM error</td>
<td>MM– Bad swap block.</td>
</tr>
<tr>
<td>16003</td>
<td>70703</td>
<td>OpenEdge SQLMM error</td>
<td>MM– No cache block.</td>
</tr>
<tr>
<td>16004</td>
<td>22704</td>
<td>Data exception</td>
<td>MM– Invalid row number.</td>
</tr>
<tr>
<td>16005</td>
<td>70705</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Invalid cache block.</td>
</tr>
<tr>
<td>16006</td>
<td>70706</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Bad swap file.</td>
</tr>
<tr>
<td>16007</td>
<td>70707</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Row too big.</td>
</tr>
<tr>
<td>16008</td>
<td>70708</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Array initialized.</td>
</tr>
<tr>
<td>16009</td>
<td>70709</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Invalid chunk number.</td>
</tr>
<tr>
<td>16010</td>
<td>70710</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Cannot create table.</td>
</tr>
<tr>
<td>16011</td>
<td>70711</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Cannot alter table.</td>
</tr>
<tr>
<td>16012</td>
<td>70712</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Cannot drop table.</td>
</tr>
<tr>
<td>16020</td>
<td>70713</td>
<td>OpenEdge SQL MM error</td>
<td>MM– TPL ctor error.</td>
</tr>
<tr>
<td>16021</td>
<td>70714</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Insertion error.</td>
</tr>
<tr>
<td>16022</td>
<td>70715</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Deletion error.</td>
</tr>
<tr>
<td>16023</td>
<td>70716</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Updation error.</td>
</tr>
<tr>
<td>16024</td>
<td>70717</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Fetching error.</td>
</tr>
<tr>
<td>16025</td>
<td>70718</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Sorting error.</td>
</tr>
<tr>
<td>16026</td>
<td>70719</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Printing error.</td>
</tr>
<tr>
<td>16027</td>
<td>70720</td>
<td>OpenEdge SQL MM error</td>
<td>MM– TPLSCAN ctor error.</td>
</tr>
<tr>
<td>16028</td>
<td>70721</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Scan fetching error.</td>
</tr>
<tr>
<td>16030</td>
<td>70722</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Can't create index.</td>
</tr>
<tr>
<td>16031</td>
<td>70723</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Can't drop index.</td>
</tr>
<tr>
<td>16032</td>
<td>70724</td>
<td>OpenEdge SQL MM error</td>
<td>MM– IXSCAN ctor error.</td>
</tr>
<tr>
<td>16033</td>
<td>70725</td>
<td>OpenEdge SQL MM error</td>
<td>MM– IX ctor error.</td>
</tr>
</tbody>
</table>
Table 1: OpenEdge SQL error codes and messages (3 of 17)

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQLSTATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>16034</td>
<td>70726</td>
<td>OpenEdge SQL MM error</td>
<td>MM– IX deletion error.</td>
</tr>
<tr>
<td>16035</td>
<td>70727</td>
<td>OpenEdge SQL MM error</td>
<td>MM– IX appending error.</td>
</tr>
<tr>
<td>16036</td>
<td>70728</td>
<td>OpenEdge SQL MM error</td>
<td>MM– IX insertion error.</td>
</tr>
<tr>
<td>16037</td>
<td>70729</td>
<td>OpenEdge SQL MM error</td>
<td>MM– IX scan fetching error.</td>
</tr>
<tr>
<td>16040</td>
<td>70730</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Begin transaction.</td>
</tr>
<tr>
<td>16041</td>
<td>70731</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Commit transaction.</td>
</tr>
<tr>
<td>16042</td>
<td>40000</td>
<td>Transaction rollback</td>
<td>***MM– Rollback transaction.</td>
</tr>
<tr>
<td>16043</td>
<td>70732</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Mark point.</td>
</tr>
<tr>
<td>16044</td>
<td>70733</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Rollback savepoint.</td>
</tr>
<tr>
<td>16045</td>
<td>70734</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Set &amp; Get isolation.</td>
</tr>
<tr>
<td>16050</td>
<td>70735</td>
<td>OpenEdge SQL MM error</td>
<td>MM– TID to char.</td>
</tr>
<tr>
<td>16051</td>
<td>70736</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Char to TID.</td>
</tr>
<tr>
<td>16054</td>
<td>70737</td>
<td>OpenEdge SQL MM error</td>
<td>MM– Bad value list size to indirect sort.</td>
</tr>
<tr>
<td>20000</td>
<td>50501</td>
<td>OpenEdge SQL rds error</td>
<td>SQL internal error.</td>
</tr>
<tr>
<td>20001</td>
<td>50502</td>
<td>OpenEdge SQL rds error</td>
<td>Memory allocation failure.</td>
</tr>
<tr>
<td>20002</td>
<td>50503</td>
<td>OpenEdge SQL rds error</td>
<td>Open database failed.</td>
</tr>
<tr>
<td>20003</td>
<td>2a504</td>
<td>Syntax error</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>20004</td>
<td>28505</td>
<td>Invalid auth specs</td>
<td>User not found.</td>
</tr>
<tr>
<td>20005</td>
<td>22506</td>
<td>Data exception</td>
<td>Table/View/Synonym not found.</td>
</tr>
<tr>
<td>20006</td>
<td>22507</td>
<td>Data exception</td>
<td>Column not found/specified.</td>
</tr>
<tr>
<td>20007</td>
<td>22508</td>
<td>Data exception</td>
<td>No columns in table.</td>
</tr>
<tr>
<td>20008</td>
<td>22509</td>
<td>Data exception</td>
<td>Inconsistent types.</td>
</tr>
<tr>
<td>20009</td>
<td>22510</td>
<td>Data exception</td>
<td>Column ambiguously specified.</td>
</tr>
<tr>
<td>20010</td>
<td>22511</td>
<td>Data exception</td>
<td>Duplicate column specification.</td>
</tr>
<tr>
<td>20011</td>
<td>22512</td>
<td>Data exception</td>
<td>Invalid length.</td>
</tr>
<tr>
<td>20012</td>
<td>22513</td>
<td>Data exception</td>
<td>Invalid precision.</td>
</tr>
<tr>
<td>20013</td>
<td>22514</td>
<td>Data exception</td>
<td>Invalid scale.</td>
</tr>
<tr>
<td>Error code</td>
<td>SQL STATE value</td>
<td>Class condition</td>
<td>Subclass message</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>20014</td>
<td>22515</td>
<td>Data exception</td>
<td>Missing input parameters.</td>
</tr>
<tr>
<td>20015</td>
<td>22516</td>
<td>Data exception</td>
<td>Subquery returns multiple rows.</td>
</tr>
<tr>
<td>20016</td>
<td>22517</td>
<td>Data exception</td>
<td>Null value supplied for a mandatory (not null) column.</td>
</tr>
<tr>
<td>20017</td>
<td>22518</td>
<td>Data exception</td>
<td>Too many values specified.</td>
</tr>
<tr>
<td>20018</td>
<td>22519</td>
<td>Data exception</td>
<td>Too few values specified.</td>
</tr>
<tr>
<td>20019</td>
<td>50520</td>
<td>OpenEdge SQL rds error</td>
<td>Cannot modify table referred to in subquery.</td>
</tr>
<tr>
<td>20020</td>
<td>42521</td>
<td>Access rule violation</td>
<td>Bad column specification for group by clause.</td>
</tr>
<tr>
<td>20021</td>
<td>42522</td>
<td>Access rule violation</td>
<td>Non-group-by expression in having clause.</td>
</tr>
<tr>
<td>20022</td>
<td>42523</td>
<td>Access rule violation</td>
<td>Non-group-by expression in select clause.</td>
</tr>
<tr>
<td>20023</td>
<td>42524</td>
<td>Access rule violation</td>
<td>Aggregate function not allowed here.</td>
</tr>
<tr>
<td>20024</td>
<td>0a000</td>
<td>Feature not supported</td>
<td>Sorry, operation not yet implemented.</td>
</tr>
<tr>
<td>20025</td>
<td>42526</td>
<td>Access rule violation</td>
<td>Aggregate functions nested.</td>
</tr>
<tr>
<td>20026</td>
<td>50527</td>
<td>OpenEdge SQL rds error</td>
<td>Too many table references.</td>
</tr>
<tr>
<td>20027</td>
<td>42528</td>
<td>Access rule violation</td>
<td>Bad field specification in order by clause.</td>
</tr>
<tr>
<td>20028</td>
<td>50529</td>
<td>OpenEdge SQL rds error</td>
<td>An index with the same name already exists.</td>
</tr>
<tr>
<td>20029</td>
<td>50530</td>
<td>OpenEdge SQL rds error</td>
<td>Index referenced not found.</td>
</tr>
<tr>
<td>20030</td>
<td>22531</td>
<td>Data exception</td>
<td>Table space with same name already exists.</td>
</tr>
<tr>
<td>20031</td>
<td>50532</td>
<td>OpenEdge SQL rds error</td>
<td>Cluster with same name already exists.</td>
</tr>
<tr>
<td>20032</td>
<td>50533</td>
<td>OpenEdge SQL rds error</td>
<td>No cluster with this name.</td>
</tr>
<tr>
<td>20033</td>
<td>22534</td>
<td>Data exception</td>
<td>Table space not found.</td>
</tr>
<tr>
<td>20034</td>
<td>50535</td>
<td>OpenEdge SQL rds error</td>
<td>Bad free (&lt;specification_name&gt;) specification.</td>
</tr>
</tbody>
</table>
Table 1: OpenEdge SQL error codes and messages (of 17)

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20035</td>
<td>50536</td>
<td>OpenEdge SQL rds error</td>
<td>At least column spec or null clause should be specified.</td>
</tr>
<tr>
<td>20036</td>
<td>07537</td>
<td>Dynamic sql error</td>
<td>Not prepared.</td>
</tr>
<tr>
<td>20037</td>
<td>24538</td>
<td>Invalid cursor state</td>
<td>Executing select statement.</td>
</tr>
<tr>
<td>20038</td>
<td>24539</td>
<td>Invalid cursor state</td>
<td>Cursor not closed.</td>
</tr>
<tr>
<td>20039</td>
<td>24540</td>
<td>Invalid cursor state</td>
<td>Open for nonselect statement.</td>
</tr>
<tr>
<td>20040</td>
<td>24541</td>
<td>Invalid cursor state</td>
<td>Cursor not opened.</td>
</tr>
<tr>
<td>20041</td>
<td>22542</td>
<td>Data exception</td>
<td>Table/View/Synonym already exists.</td>
</tr>
<tr>
<td>20042</td>
<td>2a543</td>
<td>Syntax error</td>
<td>Distinct specified more than once in query.</td>
</tr>
<tr>
<td>20043</td>
<td>50544</td>
<td>OpenEdge SQL rds error</td>
<td>Tuple size too high.</td>
</tr>
<tr>
<td>20044</td>
<td>50545</td>
<td>OpenEdge SQL rds error</td>
<td>Array size too high.</td>
</tr>
<tr>
<td>20045</td>
<td>08546</td>
<td>Connection exception</td>
<td>File does not exist or not accessible.</td>
</tr>
<tr>
<td>20046</td>
<td>50547</td>
<td>OpenEdge SQL rds error</td>
<td>Field value not null for some tuples.</td>
</tr>
<tr>
<td>20047</td>
<td>42548</td>
<td>Access rule violation</td>
<td>Granting to self not allowed.</td>
</tr>
<tr>
<td>20048</td>
<td>42549</td>
<td>Access rule violation</td>
<td>Revoking for self not allowed.</td>
</tr>
<tr>
<td>20049</td>
<td>22550</td>
<td>Data exception</td>
<td>Keyword used for a name.</td>
</tr>
<tr>
<td>20050</td>
<td>21551</td>
<td>Cardinality violation</td>
<td>Too many fields specified.</td>
</tr>
<tr>
<td>20051</td>
<td>21552</td>
<td>Cardinality violation</td>
<td>Too many indexes on this table.</td>
</tr>
<tr>
<td>20052</td>
<td>22553</td>
<td>Data exception</td>
<td>Overflow error.</td>
</tr>
<tr>
<td>20053</td>
<td>08554</td>
<td>Connection exception</td>
<td>Database not opened.</td>
</tr>
<tr>
<td>20054</td>
<td>08555</td>
<td>Connection exception</td>
<td>Database not specified or improperly specified.</td>
</tr>
<tr>
<td>20055</td>
<td>08556</td>
<td>Connection exception</td>
<td>Database not specified or database not started.</td>
</tr>
<tr>
<td>20056</td>
<td>28557</td>
<td>Invalid auth specs</td>
<td>No DBA access rights.</td>
</tr>
<tr>
<td>20057</td>
<td>28558</td>
<td>Invalid auth specs</td>
<td>No RESOURCE privileges.</td>
</tr>
<tr>
<td>20058</td>
<td>40559</td>
<td>Transaction rollback</td>
<td>Executing SQL statement for an aborted transaction.</td>
</tr>
</tbody>
</table>
### Table 1: OpenEdge SQL error codes and messages (6 of 17)

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20059</td>
<td>22560</td>
<td>Data exception</td>
<td>No files in the table space.</td>
</tr>
<tr>
<td>20060</td>
<td>22561</td>
<td>Data exception</td>
<td>Table not empty.</td>
</tr>
<tr>
<td>20061</td>
<td>22562</td>
<td>Data exception</td>
<td>Input parameter size too high.</td>
</tr>
<tr>
<td>20062</td>
<td>42563</td>
<td>Syntax error</td>
<td>Full pathname not specified.</td>
</tr>
<tr>
<td>20063</td>
<td>50564</td>
<td>OpenEdge SQL rds error</td>
<td>Duplicate file specification.</td>
</tr>
<tr>
<td>20064</td>
<td>08565</td>
<td>Connection exception</td>
<td>Invalid attach type.</td>
</tr>
<tr>
<td>20065</td>
<td>26000</td>
<td>Invalid SQL statement name</td>
<td>Invalid statement type.</td>
</tr>
<tr>
<td>20066</td>
<td>33567</td>
<td>Invalid SQL descriptor name</td>
<td>Invalid sqlda.</td>
</tr>
<tr>
<td>20067</td>
<td>08568</td>
<td>Connection exception</td>
<td>More than one database cannot be attached locally.</td>
</tr>
<tr>
<td>20068</td>
<td>42569</td>
<td>Syntax error</td>
<td>Bad arguments.</td>
</tr>
<tr>
<td>20069</td>
<td>33570</td>
<td>Invalid SQL descriptor name</td>
<td>SQLDA size not enough.</td>
</tr>
<tr>
<td>20070</td>
<td>33571</td>
<td>Invalid SQL descriptor name</td>
<td>SQLDA buffer length too high.</td>
</tr>
<tr>
<td>20071</td>
<td>42572</td>
<td>Access rule violation</td>
<td>Specified operation not allowed on the view.</td>
</tr>
<tr>
<td>20072</td>
<td>50573</td>
<td>OpenEdge SQL rds error</td>
<td>Server is not allocated.</td>
</tr>
<tr>
<td>20073</td>
<td>2a574</td>
<td>Access rule violation</td>
<td>View query specification for view too long.</td>
</tr>
<tr>
<td>20074</td>
<td>2a575</td>
<td>Access rule violation</td>
<td>View column list must be specified as expressions are given.</td>
</tr>
<tr>
<td>20075</td>
<td>21576</td>
<td>Cardinality violation</td>
<td>Number of columns in column list is less than in select list.</td>
</tr>
<tr>
<td>20076</td>
<td>21577</td>
<td>Cardinality violation</td>
<td>Number of columns in column list is more than in select list.</td>
</tr>
<tr>
<td>20077</td>
<td>42578</td>
<td>Access rule violation</td>
<td>Check option specified for noninsertable view.</td>
</tr>
<tr>
<td>20078</td>
<td>42579</td>
<td>Access rule violation</td>
<td>Given SQL statement is not allowed on the view.</td>
</tr>
<tr>
<td>20079</td>
<td>50580</td>
<td>OpenEdge SQL rds error</td>
<td>More tables cannot be created.</td>
</tr>
</tbody>
</table>
## Table 1: OpenEdge SQL error codes and messages

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20080</td>
<td>44581</td>
<td>Check option violation</td>
<td>View check option violation.</td>
</tr>
<tr>
<td>20081</td>
<td>22582</td>
<td>Data exception</td>
<td>Number of expressions projected on either side of set-op do not match.</td>
</tr>
<tr>
<td>20082</td>
<td>42583</td>
<td>Access rule violation</td>
<td>Column names not allowed in order by clause for this statement.</td>
</tr>
<tr>
<td>20083</td>
<td>42584</td>
<td>Access rule violation</td>
<td>Outerjoin specified on a complex predicate.</td>
</tr>
<tr>
<td>20084</td>
<td>42585</td>
<td>Access rule violation</td>
<td>Outerjoin specified on a sub-query.</td>
</tr>
<tr>
<td>20085</td>
<td>42586</td>
<td>Access rule violation</td>
<td>Invalid Outerjoin specification.</td>
</tr>
<tr>
<td>20086</td>
<td>42587</td>
<td>Access rule violation</td>
<td>Duplicate table constraint specification.</td>
</tr>
<tr>
<td>20087</td>
<td>21588</td>
<td>Cardinality violation</td>
<td>Column count mismatch.</td>
</tr>
<tr>
<td>20088</td>
<td>28589</td>
<td>Invalid auth specs</td>
<td>Invalid user name.</td>
</tr>
<tr>
<td>20089</td>
<td>22590</td>
<td>Data exception</td>
<td>System date retrieval failed.</td>
</tr>
<tr>
<td>20090</td>
<td>42591</td>
<td>Access rule violation</td>
<td>Table column list must be specified as expressions are given.</td>
</tr>
<tr>
<td>20091</td>
<td>2a592</td>
<td>Access rule violation</td>
<td>Query statement too long.</td>
</tr>
<tr>
<td>20092</td>
<td>2d593</td>
<td>Invalid transaction termination</td>
<td>No tuples selected by the subquery for update.</td>
</tr>
<tr>
<td>20093</td>
<td>22594</td>
<td>Data exception</td>
<td>Synonym already exists.</td>
</tr>
<tr>
<td>20094</td>
<td>hz595</td>
<td>Remote database access</td>
<td>Database link with same name already exists.</td>
</tr>
<tr>
<td>20095</td>
<td>hz596</td>
<td>Remote database access</td>
<td>Database link not found.</td>
</tr>
<tr>
<td>20096</td>
<td>08597</td>
<td>Connection exception</td>
<td>Connect String not specified/incorrect.</td>
</tr>
<tr>
<td>20097</td>
<td>hz598</td>
<td>Remote database access</td>
<td>Specified operation not allowed on a remote table.</td>
</tr>
<tr>
<td>20098</td>
<td>22599</td>
<td>Data exception</td>
<td>More than one row selected by the query.</td>
</tr>
<tr>
<td>20099</td>
<td>24000</td>
<td>Invalid cursor state</td>
<td>Cursor not positioned on a valid row.</td>
</tr>
<tr>
<td>Error code</td>
<td>SQL STATE value</td>
<td>Class condition</td>
<td>Subclass message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>20100</td>
<td>4250a</td>
<td>Access rule violation</td>
<td>Subquery not allowed here.</td>
</tr>
<tr>
<td>20101</td>
<td>2350b</td>
<td>Integrity constraint</td>
<td>No references for the table.</td>
</tr>
<tr>
<td>20102</td>
<td>2350c</td>
<td>Integrity constraint</td>
<td>Primary/Candidate key column defined null.</td>
</tr>
<tr>
<td>20103</td>
<td>2350d</td>
<td>Integrity constraint</td>
<td>No matching key defined for the referenced table.</td>
</tr>
<tr>
<td>20104</td>
<td>2350e</td>
<td>Integrity constraint</td>
<td>Keys in reference constraint incompatible.</td>
</tr>
<tr>
<td>20105</td>
<td>5050f</td>
<td>OpenEdge SQL rds error</td>
<td>Not allowed in read only isolation level.</td>
</tr>
<tr>
<td>20106</td>
<td>2150g</td>
<td>Cardinality violation</td>
<td>Invalid ROWID.</td>
</tr>
<tr>
<td>20107</td>
<td>hz50h</td>
<td>Remote database access</td>
<td>Remote database not started.</td>
</tr>
<tr>
<td>20108</td>
<td>0850i</td>
<td>Connection exception</td>
<td>Remote Network Server not started.</td>
</tr>
<tr>
<td>20109</td>
<td>hz50j</td>
<td>Remote database access</td>
<td>Remote database name not valid.</td>
</tr>
<tr>
<td>20110</td>
<td>0850k</td>
<td>Connection exception</td>
<td>TCP/IP Remote HostName is unknown.</td>
</tr>
<tr>
<td>20114</td>
<td>33002</td>
<td>Invalid SQL descriptor name</td>
<td>Fetched Value NULL &amp; indicator var not defined.</td>
</tr>
<tr>
<td>20115</td>
<td>5050l</td>
<td>OpenEdge SQL rds error</td>
<td>References to the table/record present.</td>
</tr>
<tr>
<td>20116</td>
<td>2350m</td>
<td>Integrity constraint</td>
<td>Constraint violation.</td>
</tr>
<tr>
<td>20117</td>
<td>2350n</td>
<td>Integrity constraint</td>
<td>Table definition not complete.</td>
</tr>
<tr>
<td>20118</td>
<td>4250o</td>
<td>Access rule violation</td>
<td>Duplicate constraint name.</td>
</tr>
<tr>
<td>20119</td>
<td>2350p</td>
<td>Integrity constraint</td>
<td>Constraint name not found.</td>
</tr>
<tr>
<td>20120</td>
<td>22000</td>
<td>Data exception</td>
<td>**Use of reserved word.</td>
</tr>
<tr>
<td>20121</td>
<td>5050q</td>
<td>OpenEdge SQL rds error</td>
<td>Permission denied.</td>
</tr>
<tr>
<td>20122</td>
<td>5050r</td>
<td>OpenEdge SQL rds error</td>
<td>Procedure not found.</td>
</tr>
<tr>
<td>20123</td>
<td>5050s</td>
<td>OpenEdge SQL rds error</td>
<td>Invalid arguments to procedure.</td>
</tr>
<tr>
<td>20124</td>
<td>5050t</td>
<td>OpenEdge SQL rds error</td>
<td>Query conditionally terminated.</td>
</tr>
<tr>
<td>20125</td>
<td>0750u</td>
<td>Dynamic sql-error</td>
<td>Number of open cursors exceeds limit.</td>
</tr>
</tbody>
</table>
### Table 1: OpenEdge SQL error codes and messages

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20126</td>
<td>34000</td>
<td>Invalid cursor name</td>
<td>***Invalid cursor name.</td>
</tr>
<tr>
<td>20127</td>
<td>07001</td>
<td>Dynamic sql-error</td>
<td>Bad parameter specification for the statement.</td>
</tr>
<tr>
<td>20128</td>
<td>2250x</td>
<td>Data Exception</td>
<td>Numeric value out of range.</td>
</tr>
<tr>
<td>20129</td>
<td>2250y</td>
<td>Data Exception</td>
<td>Data truncated.</td>
</tr>
<tr>
<td>20132</td>
<td>5050u</td>
<td>OpenEdge SQL rds error</td>
<td>Revoke failed because of restrict.</td>
</tr>
<tr>
<td>20134</td>
<td>5050v</td>
<td>OpenEdge SQL rds error</td>
<td>Invalid long data type column references.</td>
</tr>
<tr>
<td>20135</td>
<td>5050x</td>
<td>OpenEdge SQL rds error</td>
<td>Contains operator is not supported in this context.</td>
</tr>
<tr>
<td>20135</td>
<td>m0m01</td>
<td>OpenEdge SQL diagnostics error</td>
<td>Diagnostics statement failed.</td>
</tr>
<tr>
<td>20136</td>
<td>5050z</td>
<td>OpenEdge SQL rds error</td>
<td>Contains operator is not supported for this datatype.</td>
</tr>
<tr>
<td>20137</td>
<td>50514</td>
<td>OpenEdge SQL rds error</td>
<td>Index is not defined or does not support CONTAINS.</td>
</tr>
<tr>
<td>20138</td>
<td>50513</td>
<td>OpenEdge SQL rds error</td>
<td>Index on long fields requires that it can push down only CONTAINS.</td>
</tr>
<tr>
<td>20140</td>
<td>50512</td>
<td>OpenEdge SQL rds error</td>
<td>Procedure already exists.</td>
</tr>
<tr>
<td>20141</td>
<td>85001</td>
<td>OpenEdge SQL Stored procedure Compilation</td>
<td>Error in stored procedure compilation.</td>
</tr>
<tr>
<td>20142</td>
<td>86001</td>
<td>OpenEdge SQL Stored procedure Execution</td>
<td>Error in Stored Procedure Execution.</td>
</tr>
<tr>
<td>20143</td>
<td>86002</td>
<td>OpenEdge SQL Stored procedure Execution</td>
<td>Too many recursions in call procedure.</td>
</tr>
<tr>
<td>20144</td>
<td>86003</td>
<td>OpenEdge SQL Stored procedure Execution</td>
<td>Null value fetched.</td>
</tr>
<tr>
<td>20145</td>
<td>86004</td>
<td>OpenEdge SQL Stored procedure Execution</td>
<td>Invalid field reference.</td>
</tr>
<tr>
<td>20146</td>
<td>86005</td>
<td>OpenEdge SQL Triggers</td>
<td>Trigger with this name already exists.</td>
</tr>
<tr>
<td>20147</td>
<td>86006</td>
<td>OpenEdge SQL Triggers</td>
<td>Trigger with this name does not exist.</td>
</tr>
<tr>
<td>20148</td>
<td>86007</td>
<td>OpenEdge SQL Triggers</td>
<td>Trigger Execution Failed.</td>
</tr>
</tbody>
</table>
Table 1: OpenEdge SQL error codes and messages

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20152</td>
<td>22001</td>
<td>Data exception</td>
<td>Character string is too long.</td>
</tr>
<tr>
<td>20170</td>
<td>22P0</td>
<td>Data exception</td>
<td>An invalid reference to a sequence was used.</td>
</tr>
<tr>
<td>20172</td>
<td>22565</td>
<td>Data exception</td>
<td>Sequence already exists in current schema.</td>
</tr>
<tr>
<td>20173</td>
<td>0ALLV</td>
<td>Feature not supported</td>
<td>LIKE predicate for long data type uses unsupported feature.</td>
</tr>
<tr>
<td>20174</td>
<td>86010</td>
<td>OpenEdge SQL Triggers</td>
<td>Invalid colnum number specified for Trigger OLDROW/NEWROW getValue/setValue method.</td>
</tr>
<tr>
<td>20175</td>
<td>86011</td>
<td>OpenEdge SQL Triggers</td>
<td>Incompatible data type specified for Trigger OLDROW/NEWROW getValue/setValue method.</td>
</tr>
<tr>
<td>20176</td>
<td>85001</td>
<td>OpenEdge SQL Stored proc/Trigger</td>
<td>IO error while compiling stored procedure/trigger.</td>
</tr>
<tr>
<td>20178</td>
<td>OaC01</td>
<td>Feature not supported</td>
<td>Cannot rename table/column with check constraint.</td>
</tr>
<tr>
<td>20180</td>
<td>22915</td>
<td>Data exception</td>
<td>Long data exceeds column width.</td>
</tr>
<tr>
<td>20181</td>
<td>22916</td>
<td>Data exception</td>
<td>Long data exceeds maximum size of data that can be selected.</td>
</tr>
<tr>
<td>20211</td>
<td>22800</td>
<td>Data exception</td>
<td>Remote procedure call error.</td>
</tr>
<tr>
<td>20212</td>
<td>08801</td>
<td>Connection exception</td>
<td>SQL client bind to daemon failed.</td>
</tr>
<tr>
<td>20213</td>
<td>08802</td>
<td>Connection exception</td>
<td>SQL client bind to SQL server failed.</td>
</tr>
<tr>
<td>20214</td>
<td>08803</td>
<td>Connection exception</td>
<td>SQL NETWORK service entry is not available.</td>
</tr>
<tr>
<td>20215</td>
<td>08804</td>
<td>Connection exception</td>
<td>Invalid TCP/IP hostname.</td>
</tr>
<tr>
<td>20216</td>
<td>hz805</td>
<td>Remote database access</td>
<td>Invalid remote database name.</td>
</tr>
<tr>
<td>20217</td>
<td>08806</td>
<td>Connection exception</td>
<td>Network error on server.</td>
</tr>
<tr>
<td>20218</td>
<td>08807</td>
<td>Connection exception</td>
<td>Invalid protocol.</td>
</tr>
<tr>
<td>20219</td>
<td>2e000</td>
<td>Invalid connection name</td>
<td>***Invalid connection name.</td>
</tr>
</tbody>
</table>
Table 1: OpenEdge SQL error codes and messages

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20220</td>
<td>08809</td>
<td>Connection exception</td>
<td>Duplicate connection name.</td>
</tr>
<tr>
<td>20221</td>
<td>08810</td>
<td>Connection exception</td>
<td>No active connection.</td>
</tr>
<tr>
<td>20222</td>
<td>08811</td>
<td>Connection exception</td>
<td>No environment defined database.</td>
</tr>
<tr>
<td>20223</td>
<td>08812</td>
<td>Connection exception</td>
<td>Multiple local connections.</td>
</tr>
<tr>
<td>20224</td>
<td>08813</td>
<td>Connection exception</td>
<td>Invalid protocol in connect_string.</td>
</tr>
<tr>
<td>20225</td>
<td>08814</td>
<td>Connection exception</td>
<td>Exceeding permissible number of connections.</td>
</tr>
<tr>
<td>20226</td>
<td>80815</td>
<td>OpenEdge SQL snw error</td>
<td>Bad database handle.</td>
</tr>
<tr>
<td>20227</td>
<td>08816</td>
<td>Connection exception</td>
<td>Invalid host name in connect_string.</td>
</tr>
<tr>
<td>20228</td>
<td>28817</td>
<td>Invalid auth specs</td>
<td>Access denied (Authorization failed).</td>
</tr>
<tr>
<td>20229</td>
<td>22818</td>
<td>Data exception</td>
<td>Invalid date value.</td>
</tr>
<tr>
<td>20230</td>
<td>22819</td>
<td>Data exception</td>
<td>Invalid date string.</td>
</tr>
<tr>
<td>20231</td>
<td>22820</td>
<td>Data exception</td>
<td>Invalid number strings.</td>
</tr>
<tr>
<td>20232</td>
<td>22821</td>
<td>Data exception</td>
<td>Invalid number string.</td>
</tr>
<tr>
<td>20233</td>
<td>22822</td>
<td>Data exception</td>
<td>Invalid time value.</td>
</tr>
<tr>
<td>20234</td>
<td>22523</td>
<td>Data exception</td>
<td>Invalid time string.</td>
</tr>
<tr>
<td>20235</td>
<td>22007</td>
<td>Data exception</td>
<td>Invalid time stamp string.</td>
</tr>
<tr>
<td>20236</td>
<td>22012</td>
<td>Data exception</td>
<td>Division by zero attempted.</td>
</tr>
<tr>
<td>20238</td>
<td>22615</td>
<td>Data exception</td>
<td>Error in format type.</td>
</tr>
<tr>
<td>20239</td>
<td>2c000</td>
<td>Invalid character set name</td>
<td>Invalid character set name specified.</td>
</tr>
<tr>
<td>20240</td>
<td>5050y</td>
<td>OpenEdge SQL rds error</td>
<td>Invalid collation name specified.</td>
</tr>
<tr>
<td>20241</td>
<td>08815</td>
<td>Connection exception</td>
<td>Service in use.</td>
</tr>
<tr>
<td>20300</td>
<td>90901</td>
<td>DBS error</td>
<td>Column group column does not exist.</td>
</tr>
<tr>
<td>20301</td>
<td>90902</td>
<td>DBS error</td>
<td>Column group column already specified.</td>
</tr>
</tbody>
</table>
Table 1: OpenEdge SQL error codes and messages  

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20302</td>
<td>90903</td>
<td>DBS error</td>
<td>Column group name already specified.</td>
</tr>
<tr>
<td>20303</td>
<td>90904</td>
<td>DBS error</td>
<td>Column groups have not covered all columns.</td>
</tr>
<tr>
<td>20304</td>
<td>90905</td>
<td>DBS error</td>
<td>Column groups are not implemented in Progress storage.</td>
</tr>
<tr>
<td>23000</td>
<td>22563</td>
<td>OpenEdge SQL Data exception</td>
<td>Table create returned invalid table id.</td>
</tr>
<tr>
<td>23001</td>
<td>22564</td>
<td>OpenEdge SQL Data exception</td>
<td>Index create returned invalid index id.</td>
</tr>
<tr>
<td>25128</td>
<td>j0j28</td>
<td>OpenEdge SQL odbc trans layer</td>
<td>Query terminated as max row limit exceeded for a remote table.</td>
</tr>
<tr>
<td>25131</td>
<td>j0j29</td>
<td>OpenEdge SQL odbc trans layer</td>
<td>Unable to read column info from remote table.</td>
</tr>
<tr>
<td>30001</td>
<td>5050w</td>
<td>OpenEdge SQL rds error</td>
<td>Query aborted on user request.</td>
</tr>
<tr>
<td>30002</td>
<td>k0k02</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid network handle.</td>
</tr>
<tr>
<td>30003</td>
<td>k0k03</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid sqlnetwork INTERFACE.</td>
</tr>
<tr>
<td>30004</td>
<td>k0k04</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid sqlnetwork INTERFACE procedure.</td>
</tr>
<tr>
<td>30005</td>
<td>k0k05</td>
<td>OpenEdge SQL network interface</td>
<td>INTERFACE is already attached.</td>
</tr>
<tr>
<td>30006</td>
<td>k0k06</td>
<td>OpenEdge SQL network interface</td>
<td>INTERFACE entry not found.</td>
</tr>
<tr>
<td>30007</td>
<td>k0k07</td>
<td>OpenEdge SQL network interface</td>
<td>INTERFACE is already registered.</td>
</tr>
<tr>
<td>30008</td>
<td>k0k08</td>
<td>OpenEdge SQL network interface</td>
<td>Mismatch in pkt header size and total argument size.</td>
</tr>
<tr>
<td>30009</td>
<td>k0k09</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid server id.</td>
</tr>
<tr>
<td>30010</td>
<td>k0k10</td>
<td>OpenEdge SQL network interface</td>
<td>Reply does not match the request.</td>
</tr>
<tr>
<td>30011</td>
<td>k0k02</td>
<td>OpenEdge SQL network interface</td>
<td>Memory allocation failure.</td>
</tr>
<tr>
<td>Error code</td>
<td>SQLSTATE value</td>
<td>Class condition</td>
<td>Subclass message</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>30031</td>
<td>k0k11</td>
<td>OpenEdge SQL network interface</td>
<td>Error in transmission of packet.</td>
</tr>
<tr>
<td>30032</td>
<td>k0k12</td>
<td>OpenEdge SQL network interface</td>
<td>Error in reception of packet.</td>
</tr>
<tr>
<td>30033</td>
<td>k0k13</td>
<td>OpenEdge SQL network interface</td>
<td>No packet received.</td>
</tr>
<tr>
<td>30034</td>
<td>k0k14</td>
<td>OpenEdge SQL network interface</td>
<td>Connection reset.</td>
</tr>
<tr>
<td>30051</td>
<td>k0k15</td>
<td>OpenEdge SQL network interface</td>
<td>Network handle is inprocess handle.</td>
</tr>
<tr>
<td>30061</td>
<td>k0k16</td>
<td>OpenEdge SQL network interface</td>
<td>Could not connect to sql network daemon.</td>
</tr>
<tr>
<td>30062</td>
<td>k0k17</td>
<td>OpenEdge SQL network interface</td>
<td>Error in number of arguments.</td>
</tr>
<tr>
<td>30063</td>
<td>k0k18</td>
<td>OpenEdge SQL network interface</td>
<td>Requested INTERFACE not registered.</td>
</tr>
<tr>
<td>30064</td>
<td>k0k19</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid INTERFACE procedure id.</td>
</tr>
<tr>
<td>30065</td>
<td>k0k20</td>
<td>OpenEdge SQL network interface</td>
<td>Requested server executable not found.</td>
</tr>
<tr>
<td>30066</td>
<td>k0k21</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid configuration information.</td>
</tr>
<tr>
<td>30067</td>
<td>k0k22</td>
<td>OpenEdge SQL network interface</td>
<td>INTERFACE not supported.</td>
</tr>
<tr>
<td>30091</td>
<td>k0k23</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid service name.</td>
</tr>
<tr>
<td>30092</td>
<td>k0k24</td>
<td>OpenEdge SQL network interface</td>
<td>Invalid host.</td>
</tr>
<tr>
<td>30093</td>
<td>k0k25</td>
<td>OpenEdge SQL network interface</td>
<td>Error in tcp/ip accept call.</td>
</tr>
<tr>
<td>30094</td>
<td>k0k26</td>
<td>OpenEdge SQL network interface</td>
<td>Error in tcp/ip connect call.</td>
</tr>
<tr>
<td>30095</td>
<td>k0k27</td>
<td>OpenEdge SQL network interface</td>
<td>Error in tcp/ip bind call.</td>
</tr>
<tr>
<td>30096</td>
<td>k0k28</td>
<td>OpenEdge SQL network interface</td>
<td>Error in creating socket.</td>
</tr>
</tbody>
</table>
### Table 1: OpenEdge SQL error codes and messages

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>30097</td>
<td>k0k29</td>
<td>OpenEdge SQL network interface</td>
<td>Error in setting socket option.</td>
</tr>
<tr>
<td>30101</td>
<td>k0k30</td>
<td>OpenEdge SQL network interface</td>
<td>Interrupt occurred.</td>
</tr>
<tr>
<td>40001</td>
<td>L0L01</td>
<td>OpenEdge SQL env error</td>
<td>Error in reading configuration.</td>
</tr>
<tr>
<td>210001</td>
<td>08P00</td>
<td>Connection exception</td>
<td>Failure to acquire share schema lock during connect.</td>
</tr>
<tr>
<td>210002</td>
<td>08004</td>
<td>Connection exception</td>
<td>Failure in finding DLC environment variable.</td>
</tr>
<tr>
<td>210003</td>
<td>08004</td>
<td>Connection exception</td>
<td>DLC environment variable exceeds maximum size (&lt;\text{max_size}&gt; -&gt; &lt;\text{DLC_path}&gt;).</td>
</tr>
<tr>
<td>210004</td>
<td>08004</td>
<td>Connection exception</td>
<td>Error opening convmap.cp file (&lt;\text{filename}&gt; &lt;\text{path}&gt;).</td>
</tr>
<tr>
<td>210005</td>
<td>P1000</td>
<td>Unavailable resource</td>
<td>Failure getting lock table on table (&lt;\text{table_name}&gt;).</td>
</tr>
<tr>
<td>210011</td>
<td>08004</td>
<td>Internal error</td>
<td>Fatal error identifying database log in SQL.</td>
</tr>
<tr>
<td>210012</td>
<td>22P00</td>
<td>Data exception</td>
<td>Column (&lt;\text{column_name}&gt;) in table (&lt;\text{table_name}&gt;) has value exceeding its max length or precision.</td>
</tr>
<tr>
<td>210013</td>
<td>08004</td>
<td>Connection exception</td>
<td>Unable to complete server connection. (&lt;\text{function_name}&gt;; \text{reason} &lt;\text{summary_of_reason}&gt;).</td>
</tr>
<tr>
<td>210014</td>
<td>22P01</td>
<td>Data exception</td>
<td>Column values too big to make key. Table (&lt;\text{table_name}&gt;); index (&lt;\text{index_name}&gt;).</td>
</tr>
<tr>
<td>210015</td>
<td>P1000</td>
<td>Unavailable resource</td>
<td>Failure getting record lock on a record table (&lt;\text{table_name}&gt;).</td>
</tr>
<tr>
<td>210016</td>
<td>P1001</td>
<td>Unavailable resource</td>
<td>Lock table is full.</td>
</tr>
<tr>
<td>210017</td>
<td>P1002</td>
<td>Unavailable resource</td>
<td>Failure to acquire exclusive schema lock for DDL operation.</td>
</tr>
<tr>
<td>210018</td>
<td>0AP01</td>
<td>Unsupported feature</td>
<td>Update of word indexes not yet supported. Table (&lt;\text{table_name}&gt;), index (&lt;\text{index_name}&gt;).</td>
</tr>
</tbody>
</table>
Table 1: OpenEdge SQL error codes and messages (15 of 17)

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
</table>
| 210019     | 0A000           | Unsupported feature                                   | Scan of word indexes not yet supported. Table `<table_name>`, index `<index_name>`.
| 210020     | 0AP03           | Unsupported feature                                   | The first index created for a table may not be dropped.                          |
| 210021     | 85001           | Progress/SQL stored procedure compilation            | Location of the Java compiler was not specified.                                |
| 210044     | 86008           | OpenEdge stored procedure execution                  | Need to recompile stored procedures (run scriptSQLConvertSPTP - refer to release notes). |
| 210045     | 86009           | OpenEdge SQL triggers                                 | Need to recompile triggers (run scriptSQLConvertSPTP - refer to release notes).  |
| 210047     | 22P00           | OpenEdge SQL Update Statistics                       | Table %s.%s at Rowid %s has column %s whose value exceeding its max length or precision. |
| 210048     | 70101           | Data exception                                        | Cache overflowed.                                                                |
| 210049     | 22566           | Data exception                                        | Unable to read sequence record.                                                 |
| 210050     | 22564           | Data exception                                        | The sequence was unable to cycle to another value.                              |
| 210051     | 22563           | Data exception                                        | Sequence not found.                                                             |
| 210052     | 22P00           | Data exception                                        | Maximum number of sequences already defined.                                    |
| 210054     | 2250z           | Data exception                                        | A sequence value was referenced outside of the defined range of values.          |
| 210055     | 42807           | Access rule violation                                 | Operation not allowed on the read-only database.                                |
| 210056     | 42700           | Syntax error                                          | Syntax error at or about %s.                                                    |
| 210057     | 85001           | OpenEdge SQL Stored proc/Trigger                      | OpenEdge/SQL Java Native Interface(JNI) version not supported.                  |
| 210058     | 85001           | OpenEdge SQL Stored proc/Trigger                      | Error from Java compiler. Compiler messages follow.                             |
### Table 1: OpenEdge SQL error codes and messages

<table>
<thead>
<tr>
<th>Error code</th>
<th>SQL STATE value</th>
<th>Class condition</th>
<th>Subclass message</th>
</tr>
</thead>
<tbody>
<tr>
<td>210059</td>
<td>22P00</td>
<td>OpenEdge SQL statistics</td>
<td>Database statistics (%s) updated for all tables.</td>
</tr>
<tr>
<td>210060</td>
<td>22P00</td>
<td>OpenEdge SQL statistics</td>
<td>Database statistics (%s) updated for table %s.</td>
</tr>
<tr>
<td>210061</td>
<td>22P00</td>
<td>OpenEdge SQL statistics</td>
<td>Database statistics (%s) updated by direct user SQL statement (%s).</td>
</tr>
<tr>
<td>210062</td>
<td>70101</td>
<td>OpenEdge SQL statement mgr</td>
<td>mgr removed a prepared, never executed statement from statement cache. %s statements currently in use (%s cache).</td>
</tr>
<tr>
<td>211013</td>
<td>3F001</td>
<td>Bad schema reference</td>
<td>SQL cannot alter or drop a table or index created by ABL or SQL 89.</td>
</tr>
<tr>
<td>211014</td>
<td>3F002</td>
<td>Bad schema reference</td>
<td>Incorrect view owner name on CREATE VIEW—cannot be PUB or _FOREIGN.</td>
</tr>
<tr>
<td>211015</td>
<td>3F003</td>
<td>Bad schema reference</td>
<td>Database object (table, view, index, trigger, procedure, or synonym) owned by “sysprogress” cannot be created, dropped, or altered.</td>
</tr>
<tr>
<td>211016</td>
<td>3F004</td>
<td>Bad schema reference</td>
<td>Database schema table cannot be created, dropped, or altered.</td>
</tr>
<tr>
<td>211017</td>
<td>3F004</td>
<td>Bad schema reference</td>
<td>Attempt to insert, update, or delete a row in a schema table.</td>
</tr>
<tr>
<td>211018</td>
<td>0A000</td>
<td>Array reference error</td>
<td>Array reference/update incorrect.</td>
</tr>
<tr>
<td>218001</td>
<td>P8P18</td>
<td>OpenEdge I18N NLS error</td>
<td>Failure to create a NLS character set conversion handler.</td>
</tr>
<tr>
<td>219901</td>
<td>P0000</td>
<td>Internal error</td>
<td>Internal error &lt;error_num1&gt; &lt;error_meaning&gt; in SQL from subsystem &lt;subsystem_name&gt; function &lt;function_name&gt; called from &lt;calling_function&gt; on &lt;object_2&gt; for &lt;object_1&gt;. Save log for Progress technical support.</td>
</tr>
<tr>
<td>219902</td>
<td>P0001</td>
<td>Internal error</td>
<td>Failure reading schema during DDL operation.</td>
</tr>
<tr>
<td>Error code</td>
<td>SQL STATE value</td>
<td>Class condition</td>
<td>Subclass message</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>----------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>219903</td>
<td>P0002</td>
<td>Internal error</td>
<td>Inconsistent metadata - contact Progress technical support.</td>
</tr>
<tr>
<td>219951</td>
<td>40P00</td>
<td>Transaction rollback</td>
<td>Fatal error &lt;error_num&gt; &lt;error_meaning&gt; in SQL from subsystem &lt;subsystem_name&gt; function &lt;function_name&gt; called from &lt;calling_function&gt; on &lt;object_2&gt; for &lt;object_1&gt;. Save log for Progress technical support.</td>
</tr>
</tbody>
</table>
OpenEdge SQL System Limits

This section provides a list of the maximum sizes for various attributes of the OpenEdge® SQL database environment, and for elements of SQL queries addressed to this environment.

Table 1: OpenEdge SQL system limits

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of open cursors</td>
<td>OPEN_CURSORS</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of procedure arguments in an SQL CALL statement</td>
<td>TPE_MAX_PROC_ARGS</td>
<td>50</td>
</tr>
<tr>
<td>Maximum length of an SQL statement</td>
<td>TPE_MAX_SQLSTMTLEN</td>
<td>131000</td>
</tr>
<tr>
<td>Maximum length of a column in a table</td>
<td>TPE_MAX_FLDLEN</td>
<td>31983</td>
</tr>
<tr>
<td>Maximum length of default value specification</td>
<td>TPE_MAX_DFLT_LEN</td>
<td>250</td>
</tr>
<tr>
<td>Maximum length of a connect string</td>
<td>TPE_MAX_CONNLEN</td>
<td>100</td>
</tr>
<tr>
<td>Attribute</td>
<td>Name</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Maximum length for a table name</td>
<td>TPE_MAX_IDLEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum length for an area name</td>
<td>TPE_MAX_AREA_NAME</td>
<td>32</td>
</tr>
<tr>
<td>Maximum length for a username in a connect string</td>
<td>TPE_UNAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum length of an error message</td>
<td>TPE_MAX_ERRLEN</td>
<td>256</td>
</tr>
<tr>
<td>Maximum number of columns in a table</td>
<td>TPE_MAX_FIELDS</td>
<td>5000</td>
</tr>
<tr>
<td>Maximum number of bytes that can be inserted in a large key entry</td>
<td>MAX_KEY_DATA_SIZE</td>
<td>1980</td>
</tr>
<tr>
<td>Maximum number of bytes that can be inserted in a small key entry</td>
<td>SMALL_KEY_DATA_SIZE</td>
<td>193</td>
</tr>
<tr>
<td>Maximum length of a CHECK constraint clause</td>
<td>SQL_MAXCHKCL_SZ</td>
<td>240</td>
</tr>
<tr>
<td>Maximum number of nesting levels in an SQL statement</td>
<td>SQL_MAXLEVELS</td>
<td>25</td>
</tr>
<tr>
<td>Maximum number of table references in an SQL statement: other platforms</td>
<td>SQL_MAXTBLREF</td>
<td>250</td>
</tr>
<tr>
<td>Maximum size of input parameters for an SQL statement</td>
<td>SQL_MAXIPARAMS_SZ</td>
<td>512</td>
</tr>
<tr>
<td>Maximum number of outer references in an SQL statement</td>
<td>SQL_MAX_OUTER_REF</td>
<td>25</td>
</tr>
<tr>
<td>Maximum nesting level for view references</td>
<td>MAX_VIEW_LEVEL</td>
<td>25</td>
</tr>
<tr>
<td>Maximum number of check constraints in a table</td>
<td>SQL_MAXCHKCNSTRS</td>
<td>1000 total constraints per table</td>
</tr>
<tr>
<td>Maximum number of foreign constraints in a table</td>
<td>SQL_MAXFRNCNSTRS</td>
<td>1000 total constraints per table</td>
</tr>
<tr>
<td>Maximum LOB length</td>
<td>SQL_MAXLOB</td>
<td>1 GB</td>
</tr>
</tbody>
</table>
OpenEdge® SQL maintains a set of system tables for storing information about tables, columns, indexes, constraints, and privileges. This section describes those system catalog tables.

Overview of system catalog tables

OpenEdge SQL maintains a set of system tables for storing information about tables, columns, indexes, constraints, and privileges.

All users have read access to the system catalog tables. SQL Data Definition Language (DDL) statements and GRANT and REVOKE statements modify system catalog tables. The system tables are modified in response to these statements, as the database evolves and changes.

The owner of the system tables is sysprogress. If you connect to a OpenEdge SQL environment with a username other than sysprogress, you must use the owner qualifier when you reference a system table in a SQL query. Alternatively, you can issue a SET SCHEMA sysprogress statement to set the default username for unqualified table names to sysprogress.

Core tables store information on the tables, columns, and indexes that make up the database. The remaining tables contain detailed information on database objects and statistical information.

Table 1 lists the system catalog tables in the same order that they are presented in following sections.
<table>
<thead>
<tr>
<th>System table</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTABLES</td>
<td>Core system table; one row for each TABLE in the database</td>
</tr>
<tr>
<td>SYSCOLUMNS</td>
<td>Core system table; one row for each COLUMN of each table in the database</td>
</tr>
<tr>
<td>SYSINDEXES</td>
<td>Core system table. One row for each component of each INDEX in the database</td>
</tr>
<tr>
<td>SYSCALCTABLE</td>
<td>A single row with a single column set to the value 100</td>
</tr>
<tr>
<td>SYSNCHARSTAT</td>
<td>One row for each CHARACTER column in the database</td>
</tr>
<tr>
<td>SYSCOLAUTH</td>
<td>One row for each column for each user holding privileges on the column</td>
</tr>
<tr>
<td>SYSCOLSTAT</td>
<td>Provides statistical information on data distribution</td>
</tr>
<tr>
<td>SYSCOLUMNS_FULL</td>
<td>Superset of information in core system table SYSCOLUMNS</td>
</tr>
<tr>
<td>SYSDATATYPES</td>
<td>Information on supported data types</td>
</tr>
<tr>
<td>SYSDATESTAT</td>
<td>One set of rows for each DATE column in the database</td>
</tr>
<tr>
<td>SYSDBAUTH</td>
<td>One row for each user with database-wide privileges</td>
</tr>
<tr>
<td>SYSFLOATSTAT</td>
<td>One set of rows for each FLOAT column in the database</td>
</tr>
<tr>
<td>SYSIDXSTAT</td>
<td>Information on indexes in the database</td>
</tr>
<tr>
<td>SYSINTSTAT</td>
<td>One set of rows for each INTEGER column in the database</td>
</tr>
<tr>
<td>SYSNUMSTAT</td>
<td>One set of rows for each NUMERIC column in the database</td>
</tr>
<tr>
<td>SYSREALSTAT</td>
<td>One set of rows for each REAL column in the database</td>
</tr>
<tr>
<td>SYSSEQAUTH</td>
<td>One row for each unique user/sequence combination, holding sequence privileges on a sequence of the database</td>
</tr>
<tr>
<td>SYSSEQUENCES</td>
<td>View of OpenEdge schema table_sequence</td>
</tr>
</tbody>
</table>
## System tables and descriptions (2 of 2)

<table>
<thead>
<tr>
<th>System table</th>
<th>Summary description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSMINTSTAT</td>
<td>One set of rows for each SMALLINT column in the database</td>
</tr>
<tr>
<td>SYSSYNONYMS</td>
<td>One row for each SYNONYM in the database</td>
</tr>
<tr>
<td>SYSTABAUTH</td>
<td>One row for each unique user/table combination holding table privileges on a table in the database</td>
</tr>
<tr>
<td>SYSTABLES_FULL</td>
<td>Superset of information in core system table SYSTABLES</td>
</tr>
<tr>
<td>SYSTBLSTAT</td>
<td>Contains statistics for user tables in the database</td>
</tr>
<tr>
<td>SYSTIMESTAT</td>
<td>One set of rows for each TIME column in the database</td>
</tr>
<tr>
<td>SYSTINYINTSTAT</td>
<td>One set of rows for each TINYINT column in the database</td>
</tr>
<tr>
<td>SYSTRIGCOLS</td>
<td>One row for each column specified in each trigger in the database</td>
</tr>
<tr>
<td>SYSTRIGGER</td>
<td>One row for each trigger in the database</td>
</tr>
<tr>
<td>SYSTSSTAT</td>
<td>One set of rows for each TIMESTAMP column in the database</td>
</tr>
<tr>
<td>SYSTSTZSTAT</td>
<td>One set of rows for each TIMESTAMP WITH TIME ZONE column in the database</td>
</tr>
<tr>
<td>SYSNVARCHARSTAT</td>
<td>One set of rows for each VARCHAR column in the database</td>
</tr>
<tr>
<td>SYSVIEWS</td>
<td>One row for each VIEW in the database</td>
</tr>
<tr>
<td>SYS_CHKCOL_USAGE</td>
<td>One row for each CHECK CONSTRAINT defined on a column in the database</td>
</tr>
<tr>
<td>SYS_CHK_CONSTRS</td>
<td>One row for each CHECK CONSTRAINT defined on a user table in the database</td>
</tr>
<tr>
<td>SYS_KEYCOL_USAGE</td>
<td>One row for each column in the database defined with a PRIMARY KEY or FOREIGN KEY</td>
</tr>
<tr>
<td>SYS_REF_CONSTRS</td>
<td>One row for each table in the database defined with a REFERENTIAL INTEGRITY CONSTRAINT</td>
</tr>
<tr>
<td>SYS_TBL_CONSTRS</td>
<td>One row for each CONSTRAINT defined on a table in the database</td>
</tr>
</tbody>
</table>
SYSTABLES

Contains one row for each table in the database.

Table 2 provides details of the SYSTABLES table.

Table 2: SYSTABLES core system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>creator</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>has_ccnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>has_fcnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>has_pcnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>has_ucnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>segid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tbl_status</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>tbltype</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
</tbody>
</table>

SYSCOLUMNS

Contains one row for each column of every table in the database.

Table 3 provides details of the SYSCOLUMNS table.

Table 3: SYSCOLUMNS core system table

(1 of 2)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>col</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>collation</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>coltype</td>
<td>VARCHAR</td>
<td>10</td>
</tr>
<tr>
<td>dflt_value</td>
<td>VARCHAR</td>
<td>250</td>
</tr>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
SYSINDEXES

Contains one row for each component of an index in the database. For an index with \( n \) components, there will be \( n \) rows in this table.

Table 4 provides details of the SYSINDEXES table.

Table 4:  SYSCOLUMNS core system table  

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbreviate</td>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>active</td>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>creator</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>colname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>desc</td>
<td>VARCHAR</td>
<td>144</td>
</tr>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>idxcompress</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>idxmethod</td>
<td>VARCHAR</td>
<td>2</td>
</tr>
<tr>
<td>idxname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>idxorder</td>
<td>CHARACTER</td>
<td>1</td>
</tr>
<tr>
<td>idxowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>idxsegid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>idxseq</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>ixcol_user_misc</td>
<td>VARCHAR</td>
<td>20</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
Table 4:  SYINDEXES core system table (2 of 2)

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 5:  SYSCALCTABLE system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>fld</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6:  SYSNCHARSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>VARCHAR</td>
<td>2000</td>
</tr>
</tbody>
</table>
SYSCOLAUTH

Contains one row for the update privileges held by users on individual columns of tables in the database.

Table 7 provides details of the SYSCOLAUTH table.

Table 7: SYSCOLAUTH system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>grantor</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>ref</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>sel</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>tbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>upd</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
</tbody>
</table>

SYSCOLSTAT

Provides statistical information on data distribution for columns in tables.

Table 8 provides details of the SYSCOLSTAT table.

Table 8: SYSCOLSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>val_ts</td>
<td>TIMESTAMP</td>
<td>8</td>
</tr>
</tbody>
</table>
A superset of information in the SYSCOLUMNS core system table.

Table 9 provides details of the SYSCOLUMNS_FULL table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>array_extent</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>charset</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>col</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>collation</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>coltype</td>
<td>VARCHAR</td>
<td>20</td>
</tr>
<tr>
<td>col_label</td>
<td>VARCHAR</td>
<td>60</td>
</tr>
<tr>
<td>col_label_sa</td>
<td>VARCHAR</td>
<td>12</td>
</tr>
<tr>
<td>col_subtype</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR</td>
<td>144</td>
</tr>
<tr>
<td>df1t_value</td>
<td>VARCHAR</td>
<td>250</td>
</tr>
<tr>
<td>df1t_value_sa</td>
<td>VARCHAR</td>
<td>12</td>
</tr>
<tr>
<td>display_order</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>field_rpos</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>format</td>
<td>VARCHAR</td>
<td>60</td>
</tr>
<tr>
<td>format_sa</td>
<td>VARCHAR</td>
<td>12</td>
</tr>
<tr>
<td>help</td>
<td>VARCHAR</td>
<td>126</td>
</tr>
<tr>
<td>help_sa</td>
<td>VARCHAR</td>
<td>12</td>
</tr>
<tr>
<td>id</td>
<td>VARCHAR</td>
<td>4</td>
</tr>
<tr>
<td>label</td>
<td>VARCHAR</td>
<td>60</td>
</tr>
<tr>
<td>label_sa</td>
<td>VARCHAR</td>
<td>12</td>
</tr>
<tr>
<td>nullflag</td>
<td>CHARACTER</td>
<td>2</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>scale</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>user_misc</td>
<td>VARCHAR</td>
<td>20</td>
</tr>
</tbody>
</table>
### SYSDATATYPES

Contains information on each data type supported by the database.

Table 10 provides details of the SYSDATATYPES table.

#### Table 10: SYSDATATYPES system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoincr</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>casesensitive</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>createparams</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>datatype</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>dhtypename</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>literalprefix</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>literalsuffix</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>localtypename</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>nullable</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>odbcmoney</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>searchable</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>typeprecision</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>unsignedattr</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
</tbody>
</table>
## SYSDATESTAT

Contains a set of rows for each column of data type DATE. Used by the optimizer, each row contains a sample of values in the column.

Table 11 provides details of the SYSDATESTAT table.

### Table 11: SYSDATESTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>DATE</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

## SYSDBAUTH

Contains the database-wide privileges held by users.

Table 12 provides details of the SYSDBAUTH table.

### Table 12: SYSDBAUTH system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba_acc</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>res_acc</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
</tbody>
</table>
SYSFLOATSTAT

Contains one row for each column of data type FLOAT. Used by the optimizer, each row contains a sampling of values in the column.

Table 13 provides details of the SYSFLOATSTAT table.

Table 13: SYSFLOATSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>FLOAT</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

SYSIDXSTAT

Contains statistics for indexes in the database.

Table 14 provides details of the SYSIDXSTAT table.

Table 14: SYSIDXSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>idxid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>val_ts</td>
<td>TIMESTAMP</td>
<td>8</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
SYSINTSTAT

Contains one row for each column of data type INTEGER. Used by the optimizer, each row contains a sampling of values in the column.

Table 15 provides details of the SYSINTSTAT table.

Table 15: SYSINTSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

SYSNUMSTAT

Contains one row for each column of data type NUMERIC. Used by the optimizer, each row contains a sampling of values in the column.

Table 16 provides details of the SYSNUMSTAT table.

Table 16: SYSNUMSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>NUMERIC</td>
<td>32</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
SYSPROCBIN

Contains one or more rows for each stored procedure and trigger in the database. Each row contains compiled Java bytecode for its procedure or trigger.

Table 17 provides details of the SYSPROCBIN table.

Table 17: SYSPROCBIN system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>proc_bin</td>
<td>VARBINARY</td>
<td>2000</td>
</tr>
<tr>
<td>proc_type</td>
<td>CHARACTER</td>
<td>2</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>seq</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

SYSPROCCOLUMNS

Contains one row for each column in the result set of a stored procedure.

Table 18 provides details of the SYSPROCCOLUMNS table.

Table 18: SYSPROCCOLUMNS system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>argtype</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>col</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>datatype</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>dflt_value</td>
<td>VARCHAR</td>
<td>250</td>
</tr>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>nullflag</td>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td>proc_id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>scale</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>width</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
Contains one row for each stored procedure in the database.

Table 19 provides details of the SYSPROCEDURES table.

Table 19: SYSPROCEDURES system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>creator</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>has_resultset</td>
<td>CHARACTER</td>
<td>1</td>
</tr>
<tr>
<td>has_return_val</td>
<td>CHARACTER</td>
<td>1</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>proc_id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>proc_name</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>proc_type</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

Contains one or more rows for each stored procedure and trigger in the database. The row contains the Java source code for a procedure or trigger.

Table 20 provides details of the SYSPROCTEXT table.

Table 20: SYSPROCTEXT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>proc_text</td>
<td>VARCHAR</td>
<td>2000</td>
</tr>
<tr>
<td>proc_type</td>
<td>CHARACTER</td>
<td>2</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>seq</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
SYSREALSTAT

Contains one row for each column of data type REAL. Used by the optimizer, each row contains a sampling of values in the column.

Table 21 provides details of the SYSREALSTAT table.

Table 21: SYSREALSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>REAL</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

SYSSEQAUTH

Contains information about sequence privileges for database users.

Table 22 provides details of the SYSSEQAUTH table.

Table 22: SYSSEQAUTH system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>grantor</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>ref</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>sel</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>seq-owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>seq-name</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>upd</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
</tbody>
</table>
SYSSEQUENCES

A view of the OpenEdge schema table_sequences.

Table 23 provides details of the SYSSEQUENCES table.

**Table 23: SYSSEQUENCES system table**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-name</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>seq-num</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>seq-init</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>seq-incr</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>seq-min</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>seq-max</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>cycle-ok</td>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>seq-misc</td>
<td>VARCHAR</td>
<td>208</td>
</tr>
<tr>
<td>db-recod</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>user-misc</td>
<td>VARCHAR</td>
<td>20</td>
</tr>
<tr>
<td>seq-owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>

SYSSYNONYMS

Contains one row for each synonym in the database.

Table 24 provides details of the SYSSYNONYMS table.

**Table 24: SYSSYNONYMS system table**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ispublic</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>screator</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>sname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>sowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>sremdb</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>stbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>stbowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>
SYSTABAUTH

Contains information about table privileges for each user in the database.

Table 25 provides details of the SYSTABAUTH table.

Table 25: SYSTABAUTH system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>del</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>exe</td>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>grantor</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>ins</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>ndx</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>ref</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>sel</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>tbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>upd</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
</tbody>
</table>

SYSTABLES_FULL

A superset of information in the SYSTABLES core system table.

Table 26 provides details of the SYSTABLES_FULL table.

Table 26: SYSTABLES_FULL system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_dump</td>
<td>VARCHAR</td>
<td>126</td>
</tr>
<tr>
<td>can_load</td>
<td>VARCHAR</td>
<td>126</td>
</tr>
<tr>
<td>creator</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR</td>
<td>144</td>
</tr>
<tr>
<td>dump_name</td>
<td>VARCHAR</td>
<td>16</td>
</tr>
<tr>
<td>file_label</td>
<td>VARCHAR</td>
<td>60</td>
</tr>
<tr>
<td>Column name</td>
<td>Column data type</td>
<td>Column size</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>file_label_sa</td>
<td>VARCHAR</td>
<td>12</td>
</tr>
<tr>
<td>frozen</td>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>has_ccnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>hasFcnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>has_pcnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>has_ucnstrs</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>hidden</td>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>id</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>last_change</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>prime_index</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>segid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tbltype</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>tbl_status</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>user_misc</td>
<td>VARCHAR</td>
<td>20</td>
</tr>
<tr>
<td>valexp</td>
<td>VARCHAR</td>
<td>144</td>
</tr>
<tr>
<td>valmsg</td>
<td>VARCHAR</td>
<td>144</td>
</tr>
<tr>
<td>valmsg_sa</td>
<td>VARCHAR</td>
<td>12</td>
</tr>
</tbody>
</table>
SYSTBLSTAT

Contains statistics for tables.

Table 27 provides details of the SYSTBLSTAT table.

Table 27: SYSTBLSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>val-ts</td>
<td>TIMESTAMP</td>
<td>8</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

SYSTIMESTAT

Contains one row for each column of data type TIME. Used by the optimizer, each row contains a sampling of values in the column.

Table 28 provides details of the SYSTIMESTAT table.

Table 28: SYSTIMESTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>TIME</td>
<td>4</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
SYSTINYINTSTAT

Contains one row for each column of data type TINYINT. Used by the optimizer, each row contains a sampling of values in the column.

Table 29 provides details of the SYSTINYINTSTAT table.

Table 29: SYSTINYINTSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>TINYINT</td>
<td>1</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

SYSTRIGCOLS

Contains one row for each column specified in each UPDATE trigger in the database.

Table 30 provides details of the SYSTRIGCOLS table.

Table 30: SYSTRIGCOLS system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>triggername</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>
SYSTRIGGER

Contains one row for each trigger in the database.

Table 31 provides details of the SYSTRIGGER table.

Table 31: SYSTRIGGER system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>fire_4gl</td>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>refers_to_new</td>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td>refers_to_old</td>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td>rssid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>statement_or_row</td>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td>tbl</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>triggerid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>triggername</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>trigger_event</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>trigger_time</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
</tbody>
</table>

SYSTSSTAT

Contains one row for each column of data type TIMESTAMP. Used by the optimizer, each row contains a sampling of values in the column.

Table 32 provides details of the SYSTSSTAT table.

Table 32: SYSTSSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>TIMESTAMP</td>
<td>8</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
SYSTSTZSTAT

Contains one row for each column of data type TIMESTAMP WITH TIME ZONE. Used by the optimizer, each row contains a sampling of values in the column.

Table 33 provides details of the SYSTSTZSTAT table.

Table 33: SYSTSTZSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>12</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>

SYSNVARCHARSTAT

Contains one row for each column of data type VARCHAR. Used by the optimizer, each row contains a sampling of values in the column.

Table 34 provides details of the SYSNVARCHARSTAT table.

Table 34: SYSNVARCHARSTAT system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>colid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>tblid</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>value</td>
<td>VARCHAR</td>
<td>2000</td>
</tr>
<tr>
<td>property</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>attribute</td>
<td>INTEGER</td>
<td>4</td>
</tr>
</tbody>
</table>
SYSVIEWS

Contains one row for each VIEW in the database.

Table 35 provides details of the SYSVIEWS table.

Table 35: SYSVIEWS system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>creator</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>seq</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>viewname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>viewtext</td>
<td>VARCHAR</td>
<td>2000</td>
</tr>
</tbody>
</table>

SYS_CHKCOL_USAGE

Contains one row for each column on which a check constraint is specified.

Table 36 provides details of the SYS_CHKCOL_USAGE table.

Table 36: SYS_CHKCOL_USAGE system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnstrname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>colname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>
SYS_CHK_CONSTRS

Contains one row for each CHECK CONSTRAINT specified on a user table. The chkclause column contains the content of the CHECK clause.

Table 37 provides details of the SYS_CHK_CONSTRS table.

Table 37: SYS_CHK_CONSTRS system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>chkclause</td>
<td>VARCHAR</td>
<td>2000</td>
</tr>
<tr>
<td>chkseq</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>cnstrname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>

SYS_KEYCOL_USAGE

Contains one row for each column on which a PRIMARY KEY or FOREIGN KEY is specified.

Table 38 provides details of the SYS_KEYCOL_USAGE table.

Table 38: SYS_KEYCOL_USAGE system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnstrname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>colname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>colposition</td>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>
SYS_REF_CONSTRS

Contains one row for each REFERENTIAL INTEGRITY CONSTRAINT specified on a user table.

Table 39 provides details of the SYS_REF_CONSTRS table.

Table 39: SYS_REF_CONSTRS system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnstrname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>deleterule</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>refcnstrname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>refowner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>reftblname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>

SYS_TBL_CONSTRS

Contains one row for each table constraint in the database.

Table 40 provides details of the SYS_TBL_CONSTRS table.

Table 40: SYS_TBL_CONSTRS system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column data type</th>
<th>Column size</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnstrname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>cnstrtype</td>
<td>VARCHAR</td>
<td>1</td>
</tr>
<tr>
<td>idxname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
<tr>
<td>tblname</td>
<td>VARCHAR</td>
<td>32</td>
</tr>
</tbody>
</table>
Data Type Compatibility

This section addresses compatibility issues when using the OpenEdge® SQL environment and earlier versions of the Progress® database. Specifically, it discusses mapping between Advanced Business Language (ABL) supported data types and the corresponding OpenEdge SQL data types.

Supported ABL data types and corresponding OpenEdge SQL data types

OpenEdge SQL supports many data types that do not correspond to ABL data types. Table 1 lists the ABL data types that do correspond to OpenEdge SQL data types.

<table>
<thead>
<tr>
<th>ABL data type</th>
<th>OpenEdge SQL data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td>VARARRAY</td>
</tr>
<tr>
<td>BLOB</td>
<td>BLOB</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>CLOB</td>
<td>CLOB</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL or NUMERIC</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>INT64</td>
<td>BIGINT</td>
</tr>
</tbody>
</table>

Table 1: ABL and corresponding OpenEdge SQL data types
Table 1: ABL and corresponding OpenEdge SQL data types

<table>
<thead>
<tr>
<th>ABL data type</th>
<th>OpenEdge SQL data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGICAL</td>
<td>BIT</td>
</tr>
<tr>
<td>RAW</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>RECID</td>
<td>RECID</td>
</tr>
</tbody>
</table>

Notes

- All other SQL types are not compatible with ABL. In particular, OpenEdge SQL CHARACTER data is not compatible with the ABL. Use OpenEdge SQL type VARCHAR to map ABL CHARACTER data.

- Data columns created using OpenEdge SQL that have a data type not supported by ABL are not accessible through ABL applications and utilities.

- For more information about OpenEdge SQL data types, see the “OpenEdge SQL Language Elements” section on page 251.
OpenEdge SQL Language Elements

This section describes Standard SQL language elements that are common to OpenEdge® SQL. The language elements described in this section include:

- OpenEdge SQL identifiers
- Number formats
- Date-time formats
- Date formats
- Time formats
- Data types
- Literals
- Relational operators

OpenEdge SQL identifiers

Identifiers are user-specified names for elements such as tables, views, and columns. The maximum length for SQL identifiers is 32 characters.

The two types of SQL identifiers are:

- Conventional identifiers
- Delimited identifiers enclosed in double quotation marks
**Conventional identifiers**

Conventional SQL identifiers must:

- Begin with an uppercase or lowercase letter
- Contain only letters (A–Z), digits (0–9), or the underscore character (_)
- Not be reserved words, such as `CREATE` or `DROP`
- Use ASCII characters only

SQL does not distinguish between uppercase and lowercase letters in SQL identifiers. It converts all names specified as conventional identifiers to uppercase, but statements can refer to the names in mixed case.

**Example**

The following example illustrates the use of identifiers in a simple query statement where `CustNum`, `Order`, and `OrderDate` are the user-specified names of columns:

```sql
SELECT CustNum, COUNT(*)
FROM Order
WHERE OrderDate < TO_DATE ('3/31/2004')
GROUP BY CustNum
HAVING COUNT(*) > 10 ;
```

**Delimited identifiers**

Delimited identifiers are strings of no more than 32 ASCII characters enclosed in quotation marks (" "). Delimited identifiers allow you to create identifiers that are identical to keywords or that use special characters (such as #, &, or *) or a space.

Enclosing a name in quotation marks preserves the case of the name and allows it to be a reserved word or to contain special characters. Special characters are any characters other than letters, digits, or the underscore character. Subsequent references to a delimited identifier must also use quotation marks. To include a quotation mark character in a delimited identifier, precede it with another quotation mark.

The following code example uses a delimited identifier to create a table named "Dealer Table", where the space character is part of the name:

```sql
CREATE TABLE "Dealer Table" (name, address, city, state)
AS
SELECT name, address, city, state
FROM customer
WHERE state IN ('CA', 'NY', 'TX') ;
```
Number formats

Numeric data has cultural characteristics that international applications must address. For example, numeric separators (decimal and thousands separators) and currency symbols differ across locales and regions. Therefore, OpenEdge applications provide the capability to store, manage and display data in formats that meet the needs of the international market. Table 1 defines the number formats that are supported by OpenEdge SQL.

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$999</td>
<td>Returns a value with a leading dollar sign.</td>
</tr>
<tr>
<td>,</td>
<td>9,999</td>
<td>Returns a comma in the specified position (not a thousands separator).</td>
</tr>
<tr>
<td>.</td>
<td>99.99</td>
<td>Returns a decimal in the specified position (not a fractional indicator).</td>
</tr>
<tr>
<td>0</td>
<td>0999</td>
<td>Displays and positions a leading or trailing zero.</td>
</tr>
<tr>
<td>9</td>
<td>9999</td>
<td>Sets the number of significant digits to be displayed. Displays the leading space if positive, leading minus if negative. Leading zeros are blank except for a zero value returning a zero for the integer part of the number.</td>
</tr>
<tr>
<td>D</td>
<td>99D9</td>
<td>Returns NLS_NUMERIC_CHARACTER in the specified position. The default D character is (.).</td>
</tr>
<tr>
<td>G</td>
<td>9G99</td>
<td>Returns NLS_NUMERIC_CHARACTER in the specified position. The default G character is (,).</td>
</tr>
<tr>
<td>L</td>
<td>L999</td>
<td>Return the local currency symbol NLS_CURRENCY in the specified position.</td>
</tr>
</tbody>
</table>

Date-time formats

The TO_CHAR function supports the date-format and the time-format strings to control the output of date and time values. The format strings consist of keywords that SQL interprets and replaces with formatted values.

Syntax

TO_CHAR ( expression [ , format_string ] )
**Date formats**

*format_string*

Specifies the format of the output. SQL ignores the format string if the `expression` argument does not evaluate to a date or time.

Supply the format strings, enclosed in single quotation marks, as the second argument to the function. The format strings are case sensitive. For instance, SQL replaces `DAY` with all uppercase letters, but follows the case of `Day`.

The following example illustrates the difference between how a date value displays with and without the `TO_CHAR` function:

**Example**

```sql
SELECT C1 FROM T2;
C1
--
09/29/1952
1 record selected

SELECT TO_CHAR(C1, 'Day, Month ddth'),
    TO_CHAR(C2, 'HH12 a.m.') FROM T2;

TO_CHAR(C1,DAY, MONTH DDTH) TO_CHAR(C2,HH12 A.M.)
-------------------------------------------
Monday , September 29th  02 p.m.
1 record selected
```

**Date formats**

A date-format string can contain any of the following format keywords along with other characters. The format keywords in the format string are replaced by corresponding values to get the result. The other characters are displayed as literals. Table 2 lists the date formats and their corresponding descriptions.

**Table 2:** Date formats and descriptions

<table>
<thead>
<tr>
<th>Date format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>The century as a two-digit number.</td>
</tr>
<tr>
<td>YYYY</td>
<td>The year as a four-digit number.</td>
</tr>
<tr>
<td>YYY</td>
<td>The last three digits of the year.</td>
</tr>
<tr>
<td>YY</td>
<td>The last two digits of the year.</td>
</tr>
<tr>
<td>Y</td>
<td>The last digit of the year.</td>
</tr>
<tr>
<td>Y,YYY</td>
<td>The year as a four-digit number with a comma after the first digit.</td>
</tr>
<tr>
<td>Q</td>
<td>The quarter of the year as a one-digit number (with values 1, 2, 3, or 4).</td>
</tr>
<tr>
<td>MM</td>
<td>The month value as a two-digit number (in the range 01-12).</td>
</tr>
</tbody>
</table>
The following example illustrates the use of the `DAY`, `MONTH`, `DD`, and `TH` format strings:

```sql
SELECT C1 FROM T2;

C1
--
09/29/1952
1 record selected

SELECT TO_CHAR (C1, 'Day, Month ddth'),
       TO_CHAR (C2, 'HH12 a.m.') FROM T2;

TO_CHAR (C1, 'DAY, MONTH DDTH') TO_CHAR (C2, 'HH12 A.M.')
---------------------------- ----------------------
Monday, September 29th 02 p.m. 1 record selected
```
Time formats

A time format string can contain any of the following format keywords along with other characters. The format keywords in the format string are replaced by corresponding values to get the result. The other characters are displayed as literals.

**Table 3** lists the time formats and their corresponding descriptions.

**Table 3: Time formats and descriptions**

<table>
<thead>
<tr>
<th>Time format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>The string <strong>AM</strong> or <strong>PM</strong> depending on whether time corresponds to morning or afternoon</td>
</tr>
<tr>
<td>PM</td>
<td>The string <strong>AM</strong> or <strong>PM</strong> depending on whether time corresponds to afternoon</td>
</tr>
<tr>
<td>A.M.</td>
<td>The string <strong>A.M.</strong> or <strong>P.M.</strong> depending on whether time corresponds to morning or afternoon</td>
</tr>
<tr>
<td>P.M.</td>
<td>The string <strong>A.M.</strong> or <strong>P.M.</strong> depending on whether time corresponds to afternoon</td>
</tr>
<tr>
<td>HH12</td>
<td>The hour value as a two-digit number (in the range 00 to 11)</td>
</tr>
<tr>
<td>HH</td>
<td>The hour value as a two-digit number (in the range 00 to 23)</td>
</tr>
<tr>
<td>HH24</td>
<td>The hour value as a two-digit number (in the range 00 to 23)</td>
</tr>
<tr>
<td>MI</td>
<td>The minute value as a two-digit number (in the range 00 to 59)</td>
</tr>
<tr>
<td>SS</td>
<td>The seconds value as a two-digit number (in the range 00 to 59)</td>
</tr>
<tr>
<td>SSSSS</td>
<td>The seconds from midnight as a five-digit number (in the range 00000 to 86399)</td>
</tr>
<tr>
<td>MLS</td>
<td>The milliseconds value as a three-digit number (in the range 000 to 999)</td>
</tr>
</tbody>
</table>

**Example**

The following example illustrates the **TO_CHAR** function, and the **Day**, **Month**, **dd**, and **HH12** format strings:

```sql
SELECT C1 FROM T2;
C1
--
09/29/1952
1 record selected

SELECT TO_CHAR (C1, 'Day, Month ddth'),
     TO_CHAR (C2, 'HH12 a.m.') FROM T2;

TO_CHAR (C1,DAY, MONTH DDTH) TO_CHAR (C2,HH12 A.M.)
-----------------------------------------------
Monday , September 29th 02 p.m.
1 record selected
```

*OpenEdge® Data Management: SQL Reference*
Data types

CREATE TABLE statements specify the data type for each column in the table they define. This section describes the data types SQL supports for table columns. All the data types can store null values. A null value indicates that the value is not known and is distinct from all non-null values.

Syntax

<table>
<thead>
<tr>
<th>char_data_type</th>
<th>exact_numeric_data_type</th>
<th>approx_numeric_data_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_time_data_type</td>
<td>bit_string_data_type</td>
<td>array_data_type</td>
</tr>
</tbody>
</table>

Example

The following example illustrates the use of data types in a CREATE TABLE statement:

```
CREATE TABLE CUSTOMERS
    (CUST_NUM INTEGER NOT NULL,
     COMPANY VARCHAR (20) NOT NULL,
     CUST_REP INTEGER,
     CREDIT_LIMIT INTEGER,
     PRIMARY KEY (CUST_NUM))
;
```

The OpenEdge SQL data types are:

- CHARACTER
- EXACT NUMERIC
- APPROXIMATE NUMERIC
- DATE-TIME
- BIT STRING
- ARRAY

Each data type is described in the following sections.

Character data types

Character data strings consist of a sequence of character from a defined character set, such as ASCII. A character string may have a fixed or varying length.

This is the syntax for character data types:

Syntax

```
{ CHARACTER | CHAR } [ ( length ) ]
| { CHARACTER VARYING | CHAR VARYING | VARCHAR | LVARCHAR } [ ( length ) ]
```
CHARACTER [ ( length ) ]

CHARACTER (alias CHAR) corresponds to a null-terminated character string with the length specified. Values are padded with blanks to the specified length. The default length is 1. The maximum length is 2,000 characters.

The OpenEdge SQL representation is a variable-length string. The host language representation is equivalent to a C language character string.

{ CHARACTER VARYING | CHAR VARYING | VARCHAR | LVARCHAR } [ ( length ) ]

CHARACTER VARYING, CHAR VARYING, and VARCHAR corresponds to a variable-length character string with the maximum length specified. The default length is 1 character. The maximum length is 31,995 characters. LVARCHAR has a maximum length of 1,073,741,823. A CLOB is an object of data type LVARCHAR.

**Notes**

- For data types CHARACTER( length ) and VARCHAR( length ) the value of length specifies the number of characters.
- The maximum length can be as large as 31,995. The sum of all the column lengths of a table row must not exceed 31,960.
- Due to index size limitations, only the narrower VARCHAR columns can be indexed.

**Maximum length for VARCHAR**

The maximum length of the VARCHAR data type depends on:

- **The number of columns in a table** — More columns in a table further limits the length of VARCHAR data.
- **When a table was created** — Tables created earlier can support longer VARCHAR data than tables created later.

**National Language Support (NLS)**

The VARCHAR data type has NLS. The choice of character set affects the available character count or maximum length of the data column. The limits established above assume a single-byte character set. Using a multiple-byte character set lowers the maximum character count proportionally. For example, if all the characters in a character set take 3 bytes per character, the practical maximum is 10,660 (31,982 divided by 3). If, however, you are using a variable-width character set, you will be able to hold between 10,660 and 31,982 characters, depending on the actual mix of characters you use.
Concatenation operator

Use the concatenation operator (||) to join two text strings together.

The following example provides an example of a concatenation operator used in a query:

**Example**

```
SELECT firstname  || ' ' || lastname from Employee;
```

Exact numeric data types

Exact numeric data types are used to represent the exact value of a number. This is the syntax for exact numeric data types:

**Syntax**

```
<table>
<thead>
<tr>
<th>TINYINT</th>
<th>SMALLINT</th>
<th>INTEGER</th>
<th>BIGINT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NUMERIC</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(precison [, scale])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DECIMAL</td>
<td>(precison, scale)</td>
<td></td>
</tr>
</tbody>
</table>
```

TINYINT

Corresponds to an integer value in the range –128 to +127 inclusive.

SMALLINT

Corresponds to an integer value in the range of –32768 to 32767 inclusive.

INTEGER

Corresponds to an integer value in the range of –2147483648 to 2147483647 inclusive.

BIGINT

Corresponds to an integer value in the range of -9223372036854775808 to 9223372036854775807 inclusive.
NUMERIC | NUMBER [ ( precision [, scale ] ) ]

Corresponds to a number with the given precision (maximum number of digits) and scale (the number of digits to the right of the decimal point). By default, NUMERIC columns have a precision of 32 and a scale of 0. If NUMERIC columns omit the scale, the default scale is 0.

The range of values for a NUMERIC type column is \(-n\) to \(+n\) where \(n\) is the largest number that can be represented with the specified precision and scale. If a value exceeds the precision of a NUMERIC column, SQL generates an overflow error. If a value exceeds the scale of a NUMERIC column, SQL rounds the value.

NUMERIC type columns cannot specify a negative scale or specify a scale larger than the precision.

DECIMAL [ ( precision , scale ) ]

Equivalent to type NUMERIC.

Approximate numeric data types

Approximate numeric data types are used to define data with a wide range of values and whose precision does not have to be exact. This is the syntax for an approximate data type:

Syntax

```
{ REAL | DOUBLE PRECISION | FLOAT [ ( precision ) ] }
```

REAL

Corresponds to a single precision floating-point number equivalent to the C language float type.

DOUBLE PRECISION

Corresponds to a double precision floating-point number equivalent to the C language double type.

FLOAT [ ( precision ) ]

Corresponds to a double precision floating-point number of the given precision, in bytes. By default, FLOAT columns have a precision of 8. The REAL data type is same as a FLOAT(4), and double-precision is the same as a FLOAT(8).
Date-time data types

Date-time data types are used to define points in time. This is the syntax for the date-time data types:

Syntax

<table>
<thead>
<tr>
<th>DATE</th>
<th>TIME</th>
<th>TIMESTAMP</th>
<th>TIMESTAMP WITH TIME ZONE</th>
</tr>
</thead>
</table>

**DATE**

Stores a date value as three parts: year, month, and day. The ranges for the parts are:

- Year: 1 to 9999
- Month: 1 to 12
- Day: Lower limit is 1; the upper limit depends on the month and the year

**TIME**

Stores a time value as four parts: hours, minutes, seconds, and milliseconds. The ranges for the parts are:

- Hours: 0 to 23
- Minutes: 0 to 59
- Seconds: 0 to 59
- Milliseconds: 0 to 999

**TIMESTAMP**

Combines the parts of **DATE** and **TIME**

**TIMESTAMP WITH TIME ZONE**

Combines the elements of **TIMESTAMP** with a time zone offset

Bit string data types

Bit string data types are used to define bit strings, which are sequences of bits having the value of either 0 or 1. This is the syntax for a bit string data type:

Syntax

<table>
<thead>
<tr>
<th>BIT</th>
<th>BINARY</th>
<th>VARBINARY</th>
<th>LVARBINARY [ ( length ) ]</th>
</tr>
</thead>
</table>

**BIT**

Corresponds to a single bit value of 0 or 1.
SQL statements can assign and compare values in BIT columns to and from columns of types CHAR, VARCHAR, BINARY, VARBINARY, TINYINT, SMALLINT, and INTEGER. However, in assignments from BINARY and VARBINARY, the value of the first four bits must be 0001 or 0000.

No arithmetic operations are allowed on BIT columns.

**BINARY [(length)]**

Corresponds to a bit field of the specified length of bytes. The default length is 1 byte. The maximum length is 2000 bytes.

When inserting literals into binary data types, INSERT statements must use a special format to store values in BINARY columns. They can specify the binary values as a bit string, hexadecimal string, or character string. INSERT statements must enclose binary values in single-quote marks, preceded by b for a bit string and x for a hexadecimal string. Table 4 lists the specification formats for binary values.

### Table 4: Specification formats for binary values

<table>
<thead>
<tr>
<th>Specification</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit string</td>
<td>b ' '</td>
<td>b '10101010010000'</td>
</tr>
<tr>
<td>Hexadecimal string</td>
<td>x ' '</td>
<td>x 'ad10'</td>
</tr>
<tr>
<td>Character string</td>
<td>' '</td>
<td>'ad10'</td>
</tr>
</tbody>
</table>

SQL interprets a character string as the character representation of a hexadecimal string.

If the data inserted into a BINARY column is less than the length specified, SQL pads it with zeros.

**BINARY** data can be assigned and compared to and from columns of type BIT, CHAR, and VARBINARY. Arithmetic operations are not allowed.

**VARBINARY (length)**

Corresponds to a variable-length bit field of the specified length in bytes. The default length is 1 byte. The maximum length is 31,995 bytes. The default length is 1. Due to index limitations, only the narrower VARBINARY columns can be indexed.

**LVARBINARY (length)**

Corresponds to an arbitrarily long byte array with the maximum length defined by the amount of available disk storage up to 1,073,741,823. A BLOB is an object of data type LVARBINARY.
Maximum length for VARBINARY

The maximum length of the VARBINARY data type depends on:

- **The number of columns in a table** — More columns in a table further limits the length of VARBINARY data.
- **When a table was created** — Tables created earlier can support longer VARBINARY data than tables created later.

LVARBINARY limitations

Current limitations for LVARBINARY support are as follows:

- LVARBINARY data type will only be accessible from the SQL Engine. LVARBINARY data columns added to tables created by the ABL (Advanced Business Language) are not visible to the ABL.
- LVARBINARY data columns cannot be part of an index.
- LVARBINARY data columns cannot be used for variables or as parameters in stored procedures.
- Comparison operations are not supported on LVARBINARY columns. Comparison operations between LVARBINARY columns are not supported. Comparison operations between LVARBINARY columns and columns of other data types are not supported.
- Conversion, aggregate, and scalar functions are disallowed on this data type.
- LVARBINARY does not have National Language Support (NLS).

Language support for LVARBINARY

This data type has normal column functionality except for the following exceptions:

- A column of data type LVARBINARY is not a valid column name in a CREATE INDEX statement.
- When issuing a CREATE TABLE statement, a valid data type for the column definitions is LVARBINARY. However, LVARBINARY does not allow the column constraints of PRIMARY KEY, FOREIGN KEY, UNIQUE, REFERENCES, and CHECK.
- When creating a table with a column of data type LVARBINARY, place the table in a new AREA.
- The VALUES option on the INSERT statement is not valid for the LVARBINARY data type.
- In a SELECT statement, a WHERE, GROUP BY, HAVING, or ORDER BY clause cannot use a column of data type LVARBINARY.
- There is no support for an UPDATE of an LVARBINARY column on a table that contains a column of data type LVARBINARY. Obtain the functionality of an UPDATE on an LVARBINARY column by using the DELETE and INSERT statements for the record.
Utility support for LVARBINARY

Use BINARY DUMP/LOAD to dump and load data that contains the LVARBINARY data type. SQLDUMP and SQLLOAD do not support tables with LVARBINARY column data.

Utility support for LVARCHAR (CLOBs)

Use BINARY DUMP/LOAD to dump and load data that contains the LVARCHAR (CLOBs) data type. SQLDUMP and SQLLOAD do not support tables with LVARCHAR (CLOBs) column data.

Array data types

The ARRAY data type is a composite data value that consists of zero or more elements of a specified data type (known as the element type). VARARRAY data type allows the size of an individual element value to exceed its declared size as long as the total size of the array is smaller than the array's SQL width.

The VARARRAY type is most compatible with the ABL array data definitions. For best compatibility with the ABL, use the VARARRAY type. The ARRAY type is less ABL compatible and more SQL standard compliant.

This is the syntax for the array data type:

Syntax

```
data_type ARRAY[int] | VARARRAY[int]
```

- **data_type**
  - The data type of the array. This is also known as the element type.

  **Supported data types are:** BINARY, BIT, CHAR, VARCHAR, DATE, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, TIME, TIMESTAMP, TIMESTAMP_TZ, TINYINT, and VARBINARY.

- **[int]**
  - An unsigned integer, indicating the array’s maximum element size.

**Example**

In this example, table TBL is created. TBL has two columns: column C1 is an array of up to 3 elements, all of them type int and column C2 is a variable-sized array of up to 4 elements, all of them type varchar:

```
CREATE TABLE TBL (C1 int ARRAY[3], C2 varchar(5) VARARRAY[4]);
```

The size of any element in C2 can be up to 20 characters (5*4) with a total size of 20 characters.
Notes

- OpenEdge SQL limits an array’s size. The array’s size must be an integer between 1 and 9999.

- Array columns and element references cannot be indexed because:
  - You cannot define a UNIQUE key with columns of type ARRAY.
  - You cannot define a PRIMARY key with columns of type ARRAY.
  - You cannot define a FOREIGN key with columns of type ARRAY.

- Array columns and element references cannot be used in GROUP BY clauses.

**ARRAY element reference**

An element reference allows you to access a specific element of an array. It operates on two arguments: the first must evaluate to an array and the second must evaluate to an integer. The integer refers to the ordinal position of the element in the array (the first element in the array is element number one, the second is element number two, and so on).

It is possible to select the array as a whole value, rather than selecting individual array elements. When the array as a whole is selected, SQL returns a VARCHAR datatype value. That value comprises all the elements, converted to character form, with elements separated from each other by a “;” delimiter.

**Example**

In this example, the fourth element of the array column named `array_column` is returned:

```
SELECT array_column[4] FROM TBL;
```

**Default value for ARRAY columns**

When creating array columns, you can specify a default value.

**Example**

Since no value is specified for array column `C2` when inserting values, the default value is used. The result returned from this example would be `10;10;10`:

```
CREATE TABLE tbl (C1 int, C2 int ARRAY[3] default '10');
INSERT INTO tbl (C1) VALUES (1);
SELECT C2 FROM tbl WHERE C1 = 1;
```

**Note**

The default value is applicable only at the column level. This means that if fewer values are specified when executing an insert statement, the default will not be used to fill up the rest of the array elements. Instead, NULL is used.
Assignment

When an array is assigned to an array target, the assignment is done one element at a time. Two arrays are assignable if their element’s data types are mutually assignable. This means:

- When an array is taken from SQL data to be assigned to an array target, the number of elements in the source array equals the maximum number of elements in the target array. The value of each element of the source is assigned to the corresponding element of the target.

- If the maximum number of elements in the target array is less than the number of elements in the source array, then an error is returned.

- If the maximum number of elements in the target array is greater than the number of elements in the source array, the assignment of each of the source element values to the target elements occurs and the rest of the target elements will be assigned values of NULL.

Example

```sql
CREATE TABLE TBL (C1 int, C2 int ARRAY[3]);
INSERT into TBL values (1, '111;222;333');
UPDATE TBL SET C2 = '777;888;999';
```

Comparison

OpenEdge SQL provides two scalar comparison operators: = and <>. Two arrays are comparable if their element data types are mutually comparable. During comparison, the elements are compared pair-wise in element order. Two arrays are equal if:

- They both have the same number of elements
- Each pair of elements is equal

Two arrays are not equal if:

- They do not have the same number or elements
- At least one pair of elements is not equal

Literals

A literal, also called a constant, is a type of expression that specifies a constant value. Generally, you can specify a literal wherever SQL syntax allows an expression. Some SQL constructs allow literals but disallow other forms of expressions.

There are three types of literals:

- NUMERIC
- CHARACTER-STRING
- DATE-TIME
The following sections discuss each type of literal.

**Numeric literals**

A numeric literal is a string of digits that SQL interprets as a decimal number. SQL allows the string to be in a variety of formats, including scientific notation.

This is the syntax for numeric literals:

**Syntax**

```
[ + | - ]{ [0-9] [0-9] ... }
[ . [0-9] [0-9] ... ]
[{ E | e }[ + | - ] [0-9] {[0-9]]}
```

**Example**

The numeric strings in the following example are all valid:

```
123
123.456
-123.456
12.34E-04
```

**Character-string literals**

A character-string literal is a string of characters enclosed in single quotation marks (`'`). To include a single quotation mark in a character-string literal, precede it with an additional single quotation mark.

The `INSERT` statements in the following example show embedding quotation marks in character-string literals:

```
insert into quote values('unquoted literal');
insert into quote values('''single-quoted literal''');
insert into quote values("double-quoted literal");
insert into quote values('O''Hare'); select * from quote;
```

```
c1
--
unquoted literal
'single-quoted literal'
"double-quoted literal"
O'Hare
```

A character string literal can contain multi-byte characters in the character set used by the SQL client. Only single-byte ASCII-encoded quote marks are valid in the syntax.
Date-time literals

SQL supports special formats for literals to be used in conjunction with date-time data types. Basic predicates and the `VALUES` clause of `INSERT` statements can specify date literals directly for comparison and insertion into tables. In other cases, you need to convert date literals to the appropriate date-time data type with the `CAST`, `CONVERT`, or `TO_DATE` scalar functions.

Enclose date-time literals in single quotation marks ( ' ' ).

Notes

- All text (names of days, months, ordinal number endings) in all date-format literals must be in the English language. The default date format is American. You can explicitly request another date format by using a format string.

- Time literals are in the English language only.

Date literals

A date literal specifies a day, month, and year using any of the following formats, enclosed in single quotation marks ( ' ' ). This is the syntax for date literals:

Syntax

```
{ d 'yyyy-mm-dd' }
```

Notes

Date literals must be enclosed in single quotations, such as the case with column values in an `INSERT` statement.

Examples

The following example illustrates how to use the date literal format with an `INSERT` statement:

```
INSERT INTO dtest VALUES ( { d '2004-05-07' } )
```
The **INSERT** and **SELECT** statements in the following example show some of the supported formats for date literals:

```sql
CREATE TABLE T2 (C1 DATE, C2 TIME);
INSERT INTO T2 (C1) VALUES('5/7/56');
INSERT INTO T2 (C1) VALUES('7/MAY/1956');
INSERT INTO T2 (C1) VALUES('1956/05/07');
INSERT INTO T2 (C1) VALUES(('1956-05-07'));
INSERT INTO T2 (C1) VALUES('29-SEP-1952');
SELECT C1 FROM T2;
```

<table>
<thead>
<tr>
<th>c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1956-05-07</td>
</tr>
<tr>
<td>1956-05-07</td>
</tr>
<tr>
<td>1956-05-07</td>
</tr>
<tr>
<td>1956-05-07</td>
</tr>
<tr>
<td>1952-09-29</td>
</tr>
</tbody>
</table>

**Time literals**

Time literals specify an hour, minute, second, and millisecond, using the following format, enclosed in single quotation marks ('). This is the syntax for time literals:

**Syntax**

```
{ t 'hh:mi:ss' }
```

A time literal enclosed in an escape clause is compatible with ODBC. Precede the literal string with an open brace ( { ) and a lowercase t. End the literal with a close brace ( } ).

**Note**

If you use the ODBC escape clause, you must specify the time using the format `hh:mi:ss`.

- **hh**
  - Specifies the hour value as a two-digit number in the range 00 to 23.

- **mi**
  - Specifies the minute value as a two-digit number in the range 00 to 59.

- **ss**
  - Specifies the seconds value as a two-digit number in the range 00 to 59.

- **mls**
  - Specifies the milliseconds value as a three-digit number in the range 000 to 999.
Examples

The following example illustrates how to use the time literal format with an `INSERT` statement:

```
INSERT INTO ttest VALUES ( { t '23:22:12' } ) ;
```

The `INSERT` statements in the following example show some of the formats SQL will and will not accept for time literals:

```
INSERT INTO T2 (C2) VALUES('3');
error(-20234): Invalid time string

INSERT INTO T2 (C2) VALUES('8:30');
error(-20234): Invalid time string

INSERT INTO T2 (C2) VALUES('8:30:1');

INSERT INTO T2 (C2) VALUES('8:30:');
error(-20234): Invalid time string

INSERT INTO T2 (C2) VALUES('8:30:00');
INSERT INTO T2 (C2) VALUES('8:30:1:1');
INSERT INTO T2 (C2) VALUES({t'8:30:1:1'});
```

The `SELECT` statement in the following example illustrates which `INSERT` statements successfully inserted a row:

```
SELECT C2 FROM T2;c2
   c2
  --
08:30:01
08:30:00
08:30:01
08:30:01
```

Timestamp literals

Timestamp literals specify a date and a time separated by a space, enclosed in single quotation marks (‘ ’). This is the syntax for timestamp literals:

**Syntax**

```
{ ts 'yyyy-mm-dd hh:mi:ss' }
{ ts 'yyyy-mm-dd hh:mi:ss' }
```

A timestamp literal enclosed in an escape clause is compatible with ODBC. Precede the literal string with an open brace ( { ) and a lowercase ts. End the literal with a close brace ( } ). Note that braces are part of the syntax. If you use the ODBC escape clause, you must specify the timestamp using the format `yyyy-mm-dd hh:mi:ss`. 
date_literal
A date.

time_literal
A time literal.

Examples
The following example illustrates how to INSERT a timestamp literal into a column:

```sql
INSERT INTO DTEST
VALUES ( { ts '1956-05-07 10:41:37'} ) ;
```

The following example illustrates a timestamp literal with the ODBC escape clause:

```sql
SELECT * FROM DTEST WHERE C1 = {ts '1985-08-10 05:41:37'} ;
```

Relational operators
Relational operators specify how SQL compares expressions in basic and quantified predicates. This is the syntax for relational operators:

Syntax

| = | <> | != | ^= | |= | < | <= | > | >= |

Table 5 lists the relational operators and the resulting predicates for each operator.

<table>
<thead>
<tr>
<th>Relational operator</th>
<th>Predicate for this relational operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>True if the two expressions are equal.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>!=</td>
</tr>
<tr>
<td>&lt;</td>
<td>True if the first expression is less than the second expression.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>True if the first expression is less than or equal to the second expression.</td>
</tr>
<tr>
<td>&gt;</td>
<td>True if the first expression is greater than the second expression.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>True if the first expression is greater than or equal to the second expression.</td>
</tr>
</tbody>
</table>
Relational operators

**Basic Predicate**

A basic predicate compares two values using a relational operator. If a basic predicate specifies a query expression, then the query expression must return a single value. Basic predicates often specify an inner join.

If the value of any expression is null or the query_expression does not return any value, then the result of the predicate is set to false. This is the syntax for a basic predicate:

**Syntax**

```
expression relop { expression | ( query_expression ) }
```

**Quantified Predicate**

The quantified predicate compares a value with a collection of values using a relational operator. A quantified predicate has the same form as a basic predicate with the query_expression being preceded by the ALL, ANY, or SOME keyword. The result table returned by query_expression can contain only a single column.

When you specify ALL, the predicate evaluates to true if the query_expression returns no values or the specified relationship is true for all the values returned.

When you specify SOME or ANY, the predicate evaluates to true if the specified relationship is true for at least one value returned by the query_expression. There is no difference between the SOME and ANY keywords. The predicate evaluates to false if the query_expression returns no values or if the specified relationship is false for all the values returned. This is the syntax for a quantified predicate:

**Syntax**

```
expression relop { ALL | ANY | SOME } ( query_expression )
```

**Example**

```
10 < ANY ( SELECT COUNT(*)
    FROM order_tbl
    GROUP BY custid ;
)
```

**BETWEEN Predicate**

The BETWEEN predicate can be used to determine if a value is within a specified value range or not. The first expression specifies the lower bound of the range and the second expression specifies the upper bound of the range.

The predicate evaluates to true if the value is greater than or equal to the lower bound of the range and less than or equal to the upper bound of the range. This is the syntax for a BETWEEN predicate.
Relational operators

Syntax

expression [ NOT ] BETWEEN expression AND expression

Example

salary BETWEEN 20000.00 AND 100000.00

NULL Predicate

The NULL predicate can be used for testing null values of database table columns. This is the syntax for a NULL predicate.

Syntax

column_name IS [ NOT ] NULL

Example

contact_name IS NOT NULL

LIKE Predicate

The LIKE predicate searches for strings that have a certain pattern. The pattern is specified after the LIKE keyword in a string constant. The pattern can be specified by a string in which the underscore (_ ) and percent sign ( % ) characters have special semantics.

Use the ESCAPE clause to disable the special semantics given to the characters ( _ ) and ( % ). The escape character specified must precede the special characters in order to disable their special semantics. This is the syntax for a LIKE predicate:

Syntax

column_name [ NOT ] LIKE string_constant [ ESCAPE escape_character ]

Notes

- The column_name specified in the LIKE predicate can be a column, a string constant, or an arbitrary character expression (such as SUBSTRING or LTRIM).
- The string_constant may be a string constant or a scalar function call.
- The escape_character must be a one character string constant.
- A percent sign ( % ) in the pattern matches zero or more characters of the column string.
- An underscore symbol ( _ ) in the pattern matches any single character of the column string.
• The `LIKE` predicate is multi-byte enabled. The `string_constant` and the `escape_character` may contain multi-byte characters, and the `escape_character` can be a multi-byte character. A percent sign ( `%` ) or an underscore ( `_` ) in the `string_constant` can represent a multi-byte character. However, the percent sign or underscore itself must be the single-byte ASCII encoding.

Example

This example illustrates three ways to use the `LIKE` predicate:

```sql
cust_name LIKE '%Computer%'
cust_name LIKE '___'
item_name LIKE '%\_%' ESCAPE '\'
```

In the first `LIKE` clause, for all strings with the substring 'Computer' the predicate evaluates to true. In the second `LIKE` clause, for all strings which are exactly three characters long the predicate evaluates to true. In the third `LIKE` clause the backslash character ( \ ) is specified as the escape character, which means that the special interpretation given to the underscore character ( _ ) is disabled. The pattern evaluates to `TRUE` if the `item_name` column has embedded underscore characters.

**EXISTS Predicate**

The `EXISTS` predicate can be used to check for the existence of specific rows. The `query_expression` returns rows rather than values. The predicate evaluates to true if the number of rows returned by the `query_expression` is nonzero. This is the syntax for an `EXISTS` predicate:

**Syntax**

```sql
EXISTS (query_expression)
```

**Example**

In this example, the predicate returns the item names of all items which contain sub-items:

```sql
SELECT itemname FROM pub.mtitem
   WHERE EXISTS(SELECT subitem FROM pub.mtitem);
```

**IN Predicate**

The `IN` predicate can be used to compare a value with a set of values. If an `IN` predicate specifies a query expression, then the result table it returns can contain only a single column. This is the syntax for an `IN` predicate:

**Syntax**

```sql
expression [ NOT ] IN
   { (query_expression) | (constant , constant [ , ... ] ) }
```
**Example**

```sql
address.state IN ('MA', 'NH')
```

**OUTER JOIN Predicate**

An outer join predicate specifies two tables and returns a result table that contains all the rows from one of the tables, even if there is no matching row in the other table.

**Syntax**

```
[ table_name.]column = [ table_name.]column (+)
```

**Numeric arithmetic expressions**

Numeric arithmetic expressions compute a value using addition, subtraction, multiplication, and division operations on numeric literals and expressions that evaluate to any numeric data type.

**Syntax**

```
[ + | - ]{ numeric_literal | numeric_expr }
[ { + | - | * | / } numeric_arith_expr ]
```

+ or -

Unary operators.

`numeric_literal`

Number value.

`numeric_expr`

Evaluates to a numeric data type:

+ or - or * or /

Operators for addition, subtraction, multiplication, and division. SQL evaluates numeric arithmetic expressions in the following order:

- Unary plus or minus
- Expressions in parentheses
- Multiplication and division, from left to right
- Addition and subtraction, from left to right
Date arithmetic expressions

Date arithmetic expressions compute the difference between date-time expressions in terms of days or milliseconds. SQL supports these forms of date arithmetic:

- Addition and subtraction of integers to and from date-time expressions
- Subtraction of one date-time expression from another

Syntax

```
| date_time_expr { + | - } int_expr |
| date_time_expr - date_time_expr |
```

**date_time_expr**

Returns a value of type `DATE` or `TIME` or `TIMESTAMP`. A single date-time expression cannot mix data types, however. All elements of the expression must be the same data type.

Date-time expressions can contain date-time literals, but they must be converted to `DATE` or `TIME` using the `CAST`, `CONVERT`, or `TO_DATE` functions.

**int_expr**

Returns an integer value. SQL interprets the integer differently depending on the data type of the date-time expression:

- For `DATE` expressions, integers represent days
- For `TIME` expressions, integers represent milliseconds
- For `TIMESTAMP` expressions, integers represent milliseconds

Examples

The following example manipulates `DATE` values using date arithmetic. SQL interprets integers as days and returns date differences in units of days:

```
SELECT C1, C2, C1-C2 FROM DTEST
  c1                 c2           c1-c2
  ---------------------------------------
  1956-05-07         1952-09-29  1316

select sysdate,
       sysdate - 3 ,
       sysdate - cast ('9/29/52' as date)
from dtest;
      sysdate       sysdate-3  sysdate-convert(date,9/29/52)
                  -------------------------------
  1995-03-24     1995-03-21     15516
```
The following example manipulates `TIME` values using date arithmetic. SQL interprets integers as milliseconds and returns time differences in milliseconds:

```sql
select systime,
       systime - 3000,
       systime - cast ('15:28:01' as time)
from dtest;
```

<table>
<thead>
<tr>
<th>systime</th>
<th>systime-3000</th>
<th>systime-convert(time,15:28:01)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:28:09</td>
<td>15:28:06</td>
<td>8000</td>
</tr>
</tbody>
</table>
OpenEdge SQL Elements and Statements in Backus Naur Form

This section presents OpenEdge® SQL language elements and statements in Backus Naur Form (BNF). Information on BNF elements and statements include:

- Data types syntax in BNF
- Expressions syntax in BNF
- Literals syntax in BNF
- Query Expressions syntax in BNF
- Search conditions syntax in BNF
- Statements, DDL and DML syntax in BNF

Data types syntax in BNF

Data Type

Syntax

```
data_type ::= char_data_type | exact_numeric_data_type | approx_numeric_data_type | date_time_data_type | bit_string_data_type
```
Character data type

Syntax

```
char_data_type ::= 
{ CHARACTER | CHAR } [ ( length ) ]
| { CHARACTER VARYING | CHAR VARYING | CLOB | VARCHAR }
| ( length ) ]
```

Exact numeric data type

Syntax

```
exact_numeric_data_type ::= 
TINYINT
| SMALLINT
| INTEGER
| NUMERIC | NUMBER [ ( precision [, scale ] ) ]
| DECIMAL [ ( precision , scale ) ]
```

Approximate numeric data type

Syntax

```
approx_numeric_data_type ::= 
{ REAL | DOUBLE PRECISION | FLOAT [ ( precision ) ] )
```

Date-time data type

Syntax

```
date_time_data_type ::=
DATE | TIME | TIMESTAMP | TIMESTAMP WITH TIME ZONE
```

Bit string data type

Syntax

```
bit_string_data_type ::= 
BIT | BINARY [ ( length ) ] | BLOB [ ( length ) ]
| VARBINARY [ ( length ) ] | LONG VARBINARY [ ( length ) ]
```
Expressions syntax in BNF

Expression (expr)

Syntax

```plaintext
expr ::= 
[ { table_name.| alias. }] column_name 
  | character_literal 
  | numeric_literal 
  | date-time_literal 
  | aggregate_function 
  | scalar_function 
  | numeric_arith_expr 
  | date_arith_expr 
  | conditional_expr 
  | (expr)
```

Numeric arithmetic expression

Syntax

```plaintext
numeric_arith_expr ::= 
[ + | - ] { numeric_literal | numeric_expr } 
[ { | + | - | * | / } numeric_arith_expr ]
```

Date arithmetic expression

Syntax

```plaintext
date_arith_expr ::= 
date_time_expr { + | - } int_expr 
  | date_time_expr - date_time_expr
```

Conditional expression

Case expression

A type of conditional expression.

Syntax

```plaintext
ase_expr ::= 
searched_case_expr | simple_case_expr
```
Searched case expression

Syntax

```
searched_case_expr ::= CASE 
  WHEN search_condition THEN { result_expr | NULL } [ , ... ] 
  [ ELSE expr | NULL ] 
END
```

Simple case expression

Syntax

```
simple_case_expr ::= CASE primary_expr 
  WHEN expr THEN { result_expr | NULL } [ , ... ] 
  [ ELSE expr | NULL ] 
END
```

Literals syntax in BNF

Date literal

Syntax

```
date-literal ::= ( \ ' yyyy-mm-dd ' )  
  | mm-dd-yyyy 
  | mm/dd/yyyy 
  | mm/dd/yy 
  | mm/dd/yy 
  | yyyy-mm-dd 
  | yyyy/mm/dd 
  | dd-mon-yyyy 
  | dd/month/yyyy 
  | dd-mon-yy 
  | dd/month/yy 
```

Time literal

Syntax

```
time_literal ::= ( t 'hh:mi:ss' ) | hh:mi:ss[.mls ]
```
Timestamp literal

Syntax

\[
\text{timestamp literal ::= } \\
( \text{ t 'yyyy-mm-dd hh:mm:ss' } | \text{'date literal time literal'} )
\]

Timestamp with time zone literal

Syntax

\[
\text{timestamp with time zone literal ::= } \\
( \text{ t 'yyyy-mm-dd hh:mm:ss - hh:mm:ss' } | \text{'date literal time literal'} )
\]

Query Expressions syntax in BNF

Query expression

Syntax

\[
\text{query expression ::= } \\
\text{query specification} \\
| \text{query expression set_operator query expression} \\
| ( \text{query expression} )
\]

Set operator

Syntax

\[
\text{set_operator ::= } \\
\{ \text{UNION [ ALL ]| INTERSECT | MINUS } \}
\]

Query specification

Syntax

\[
\text{query specification ::= } \\
\text{SELECT [ ALL | DISTINCT ]} \\
\text{FROM table ref [ , table ref ] ...} \\
[ \text{WHERE search condition} ] \\
[ \text{GROUP BY [ table. column name} \\
\{ [..., [ \text{table. column name } ,... \\
[ \text{HAVING search condition} ] \\
[ \text{WITH locking hints} ] \\
\]
\]
Table reference

Syntax

\[
\text{table_ref ::= }
\begin{cases}
\text{table_name [ AS ] [ alias [(column_alias [, ... ] ) ] ]} \\
\text{(query_expression) [ AS ] alias [(column_alias [, ... ] ) ]} \\
\text{( joined_table []) ]}
\end{cases}
\]

Joined table

Syntax

\[
\text{joined_table ::= }
\begin{cases}
\text{table_ref CROSS JOIN table_ref} \\
\text{table_ref [ INNER | LEFT ]} \text{ JOIN table_ref ON search_condition}
\end{cases}
\]

From clause inner join

Syntax

\[
\text{from_clause_inner_join ::= }
\begin{cases}
\text{FROM table_ref CROSS JOIN table_ref} \\
\text{FROM table_ref [ INNER ] JOIN table_ref}
\text{ ON search_condition}
\end{cases}
\]

Where clause inner join

Syntax

\[
\text{where_clause_inner_join ::= }
\begin{cases}
\text{FROM table_ref, table_ref WHERE search_condition}
\end{cases}
\]

From clause outer join

Syntax

\[
\text{from_clause_outer_join ::= }
\begin{cases}
\text{FROM table_ref LEFT OUTER JOIN table_ref}
\text{ ON search_condition}
\end{cases}
\]

Where clause outer join

Syntax

\[
\text{where_clause_outer_join ::= }
\begin{cases}
\text{WHERE [ table_name.]column (+) = [ table_name.]column} \\
\text{WHERE [ table_name.]column = [ table_name.]column (+}}
\end{cases}
\]
Search conditions syntax in BNF

Search condition

Syntax

\[
\text{search\_condition} ::= \\
\quad [ \text{NOT} ] \text{predicate} \\
\quad [ \{ \text{AND} | \text{OR} \} \{ \text{predicate} | ( \text{search\_condition} ) \} ]
\]

Predicate

Syntax

\[
\text{predicate} ::= \\
\quad \text{basic\_predicate} \\
\quad | \text{quantified\_predicate} \\
\quad | \text{between\_predicate} \\
\quad | \text{null\_predicate} \\
\quad | \text{like\_predicate} \\
\quad | \text{exists\_predicate} \\
\quad | \text{in\_predicate} \\
\quad | \text{outer\_join\_predicate}
\]

Relational operator

Syntax

\[
\text{relop} ::= \\
\quad = | <> | != | ^= | < | <= | > | >=
\]

Basic predicate

Syntax

\[
\text{basic\_predicate} ::= \\
\quad \text{expr} \text{ relop} \{ \text{expr} | ( \text{query\_expression} ) \}
\]

Quantified predicate

Syntax

\[
\text{quantified\_predicate} ::= \\
\quad \text{expr} \text{ relop} \{ \text{ALL} | \text{ANY} | \text{SOME} \} ( \text{query\_expression} )
\]

Between predicate

Syntax

\[
\text{between\_predicate} ::= \\
\quad \text{expr} \ [ \text{NOT} ] \ \text{BETWEEN} \ \text{expr} \ \text{AND} \ \text{expr}
\]
Null predicate

Syntax

\[
\text{null.predicate ::= column.name IS [ NOT ] NULL}
\]

Like predicate

Syntax

\[
\text{like.predicate ::= column.name [ NOT ] LIKE string.constant [ ESCAPE escape.character ]}
\]

Exists predicate

Syntax

\[
\text{exists.predicate ::= EXISTS (query.expression)}
\]

In predicate

Syntax

\[
\text{in.predicate ::= expr [ NOT ] IN \{ (query.expression) | (constant , constant [, ... ] ) \}}
\]

Outer join predicate

Syntax

\[
\text{outer.join.predicate ::= [ table.name.]column = [ table.name.]column (+) |
\text{[ table.name.]column (+) = [ table.name.]column}
\]
Statements, DDL and DML syntax in BNF

This section lists OpenEdge SQL Data Definition Language (DDL) and Data Manipulation Language (DML) statements in Backus-Naur Form (BNF).

ALTER USER
Syntax

```
alter user statement ::= 
ALTER USER { 'username' | 'username@domain_name' }, 'old_password', 'new_password';
```

CALL
Syntax

```
call statement ::= 
CALL proc_name ( [ parameter ] [ , ... ] ) ;
```

COMMIT
Syntax

```
commit statement ::= 
COMMIT [ WORK ] ;
```

CREATE INDEX
Syntax

```
create index statement ::= 
CREATE [ UNIQUE ] INDEX index_name 
ON table_name 
( { column_name [ ASC | DESC ] [ . ... ] } 
[ AREA area_name ] ;
```

CREATE PROCEDURE
Syntax

```
create procedure statement ::= 
CREATE PROCEDURE [ owner_name.]procname 
( [ parameter_decl [ , ... ] ] 
) 
[ RESULT ( column_name data_type [ , ... ] ) ] 
[ IMPORT 
  java_import_clause ] 
BEGIN 
java_snippet 
END
```
Parameter Declaration
Syntax

```{parameter_decl ::= { IN | OUT | INOUT } parameter_name data_type}
```

CREATE SYNONYM
Syntax

```create synonym statement ::= CREATE [ PUBLIC ] SYNONYM synonym FOR [ owner_name.] { table_name | view_name | synonym };```

CREATE TABLE
Syntax

```create table statement ::= CREATE TABLE [ owner_name.] table_name ( { column_definition | table_constraint }, ... ) [ AREA area_name ];
create table statement ::= CREATE TABLE [ owner_name.] table_name [ ( column_name [ NULL | NOT NULL ], ... ) ] [ AREA area_name ] AS query_expression ;```

Column Definition
Syntax

```column_definition ::= column_name data_type [ DEFAULT { literal | NULL | SYSDATE } ] [ column_constraint [ column_constraint ... ] ]
```

Column Constraint
Syntax

```column_constraint ::= [ CONSTRAINT constraint_name ]
NOT NULL [ PRIMARY KEY | UNIQUE ]
| REFERENCES [ owner_name.] table_name [ { column_name } ]
| CHECK ( search_condition )
```
### Table Constraint

**Syntax**

```sql
table_constraint ::= [ CONSTRAINT constraint_name ]
    PRIMARY KEY ( column [ , ... ] )
| UNIQUE ( column [ , ... ] )
| FOREIGN KEY ( column [ , ... ] )
    REFERENCES [ owner_name. ]table_name [ ( column [ , ... ] ) ]
| CHECK ( search_condition )
```

### CREATE TRIGGER

**Syntax**

```sql
create trigger statement ::= CREATE TRIGGER [ owner_name. ]trigname
{ BEFORE | AFTER }
{ INSERT | DELETE | UPDATE [ OF column_name [ , ... ] ]
ON table_name
[ REFERENCING { OLDROW [ , NEWROW ] | NEWROW [ , OLDROW ] } ]
[ FOR EACH { ROW | STATEMENT } ]
[ IMPORT
    java_import_clause ]
BEGIN
    java_snippet
END
```

### CREATE USER

**Syntax**

```sql
create user statement ::= CREATE USER
    { 'username' | 'username@domain_name' }; 'password' ;
```

### CREATE VIEW

**Syntax**

```sql
create view statement ::= CREATE VIEW [ owner_name. ]view_name
    [ ( column_name, column_name, ... ) ]
AS [ ( ) query_expression [ ] ]
[ WITH CHECK OPTION ] ;
```
**DELETE**

**Syntax**

```sql
delete statement ::= 
DELETE FROM [owner_name.]{table_name | view_name} 
[ WHERE search_condition ];
```

**DROP INDEX**

**Syntax**

```sql
drop index statement ::= 
DROP INDEX [index_owner_name.] index_name 
[ ON [table_owner_name.] table_name ]
```

**DROP PROCEDURE**

**Syntax**

```sql
drop procedure statement ::= 
DROP PROCEDURE [owner_name.] procedure_name;
```

**DROP SYNONYM**

**Syntax**

```sql
drop synonym statement ::= 
DROP [PUBLIC] SYNONYM synonym;
```

**DROP TABLE**

**Syntax**

```sql
drop table statement ::= 
DROP TABLE [owner_name.] table_name;
```

**DROP TRIGGER**

**Syntax**

```sql
drop trigger statement ::= 
DROP TRIGGER [owner_name.] trigger_name;
```

**DROP USER**

**Syntax**

```sql
drop user statement ::= 
DROP USER {username | 'username@domain_name';
```
DROP VIEW
Syntax

\[\text{drop view statement ::=} \]
\[\text{DROP VIEW} \left[ \text{owner_name.} \right] \text{view_name ;} \]

GRANT RESOURCE, DBA
Syntax

\[\text{grant resource, dba statement ::=} \]
\[\text{GRANT} \left\{ \text{RESOURCE , DBA} \right\} \text{TO user_name [ , user_name ] , ...} ; \]

GRANT PRIVILEGE
Syntax

\[\text{grant privilege statement ::=} \]
\[\text{GRANT} \left\{ \text{privilege [ , privilege ] , ... | ALL [ PRIVILEGES]} \right\} \]
\[\text{ON table_name} \]
\[\text{TO} \left\{ \text{user_name [ , user_name ] , ... | PUBLIC} \right\} \]
\[\text{[WITH GRANT OPTION] ;} \]

PRIVILEGE Syntax

Syntax

\[\text{privilege ::=} \]
\[\left\{ \text{SELECT | INSERT | DELETE | INDEX} \right\] \]
\[\text{UPDATE [ ( column , column , ... ) ]} \]
\[\text{REFERENCES [ ( column , column , ... ) ]} \]

INSERT
Syntax

\[\text{insert statement ::=} \]
\[\text{INSERT INTO} \left[ \text{owner_name.} \right] \text{table_name | view_name} \]
\[\left[ ( \text{column_name [, column_name ] , ... } \right] \]
\[\left\{ \text{VALUES [ value [, value ] , ... ] | query_expression} \right\} ; \]

LOCK TABLE
Syntax

\[\text{lock table statement ::=} \]
\[\text{LOCK TABLE} \text{table_name [, table_name ] , ... IN} \left\{ \text{SHARE | EXCLUSIVE} \right\} \text{MODE ;} \]
REVOKE RESOURCE, DBA
Syntax

```
revoke resource, dba statement ::= 
REVOKE { RESOURCE | DBA }
  FROM { user_name [ , user_name ] ... } ;
```

REVOKE PRIVILEGE
Syntax

```
revoke privilege statement ::= 
REVOKE [ GRANT OPTION FOR ]
  { privilege [ , privilege , ] ... | ALL [ PRIVILEGES ]}
  ON table_name
  FROM { user_name [ , user_name ] ... | PUBLIC }
  [ RESTRICT | CASCADE ] ;
```

PRIVILEGE Syntax
Syntax

```
privilege ::= 
{ SELECT | INSERT | DELETE | INDEX
  | UPDATE [ ( column , column , ... ) ]
  | REFERENCES [ ( column , column , ... ) ] }
```

ROLLBACK
Syntax

```
rollback statement ::= 
ROLLBACK [ WORK ] ;
```

SELECT
Syntax

```
select statement ::= 
query_expression
  ORDER BY { expr | posn } [ ASC | DESC ]
  [ , { expr | posn } [ ASC | DESC ] , ... ]
  FOR UPDATE [ OF [ table ] column_name , ... ]
 ;
```

SET SCHEMA
Syntax

```
set schema statement ::= 
SET SCHEMA { 'string_literal' | ? | USER }
UPDATE
Syntax

update statement ::= 
UPDATE table_name
   SET assignment [, assignment ], ... [ WHERE search_condition ] ;

Assignment clause
Syntax

assignment ::= 
column = { expr | NULL }
   | ( column [, column ], ... ) = ( expr [, expr ], ... )
   | ( column [, column ], ... ) = ( query_expression )

UPDATE STATISTICS
Syntax

update statistics statement ::= 
UPDATE ( [ table_name | index_name | [ ALL ] column_name ] STATISTICS
 [ AND ] ) ... [ FOR table_name ] ;
Compliance with Industry Standards

This section identifies the level of ANSI SQL-92 compliance and ODBC SQL Grammar compliance for OpenEdge® statements, and the SQL-92 and ODBC compatibility for OpenEdge SQL scalar functions, as defined in the following sections:

• Scalar functions
• SQL-92 DDL and DML statements

Scalar functions

Table 1 lists OpenEdge SQL scalar functions. A check mark identifies the compatibility of the function as SQL-92 compatible, ODBC compatible, or a Progress® extension.

Table 1: Compatibility of SQL-92 scalar functions

<table>
<thead>
<tr>
<th>Scalar function</th>
<th>SQL-92</th>
<th>ODBC</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ACOS</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADD_MONTHS</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>ASCII</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ASIN</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATAN</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ATAN2</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 1: Compatibility of SQL-92 scalar functions

<table>
<thead>
<tr>
<th>Scalar function</th>
<th>SQL-92</th>
<th>ODBC</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CEILING</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CHAR</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CHR</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>COALESCE</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CONCAT</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CONVERT</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>Requires ODBC escape clause – { fn }</td>
</tr>
<tr>
<td>CONVERT</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>Not compatible with ODBC CONVERT</td>
</tr>
<tr>
<td>COS</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CURDATE</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CURTIME</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DATABASE</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DAYNAME</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DAYOFMONTH</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DAYOFWEEK</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DAYOFYEAR</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DB_NAME</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>DECODE</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>DEGREES</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>EXP</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>FLOOR</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>GREATEST</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>HOUR</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>IFNULL</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>INITCAP</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>INSERT</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>INSTR</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>
### Table 1: Compatibility of SQL-92 scalar functions (3 of 5)

<table>
<thead>
<tr>
<th>Scalar function</th>
<th>SQL-92</th>
<th>ODBC</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_DAY</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>LCASE</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LEAST</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>LEFT</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LENGTH</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LOCATE</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LOG10</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LOWER</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LPAD</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>LTRIM</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MINUTE</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MOD</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MONTH</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MONTHNAME</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MONTHS_BETWEEN</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>NEXT_DAY</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>NOW</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>NULLIF</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>NVL</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>PI</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>POWER</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>PREFIX</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>PRO_ARR_DESCAPE</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>PRO_ARR_ESCAPE</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>PRO_ELEMENT</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>QUARTER</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>RADIAN</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>RAND</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
### Table 1: Compatibility of SQL-92 scalar functions

<table>
<thead>
<tr>
<th>Scalar function</th>
<th>SQL-92</th>
<th>ODBC</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPEAT</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>REPLACE</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>RIGHT</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ROUND</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>ROWID</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>RPAD</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>RTRIM</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SECOND</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SIGN</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SIN</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SQRT</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SUBSTR</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>SUBSTRING</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SUFFIX</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>SYSDATE</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>SYSTIME</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>SYSTIMESTAMP</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>TAN</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>TO_CHAR</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>TO_DATE</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>TO_NUMBER</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>TO_TIME</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>TO_TIMESTAMP</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>TRANSLATE</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>UCASE</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>UPPER</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>USER</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>
Table 1: Compatibility of SQL-92 scalar functions (5 of 5)

<table>
<thead>
<tr>
<th>Scalar function</th>
<th>SQL-92</th>
<th>ODBC</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEEK</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>YEAR</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

SQL-92 DDL and DML statements

Table 2 lists OpenEdge SQL DDL and DML Statements. A check mark identifies the compliance of each statement as SQL-92, a level of ODBC SQL Grammar, or as a Progress extension.

Table 2: Compliance of SQL-92 DDL and DML statements (1 of 2)

<table>
<thead>
<tr>
<th>OpenEdge SQL statement</th>
<th>SQL-92</th>
<th>ODBC SQL grammar</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER USER</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>–</td>
<td>Extended</td>
<td>–</td>
<td>Must enclose in an ODBC escape clause { fn }</td>
</tr>
<tr>
<td>COMMIT</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CONNECT</td>
<td>✓</td>
<td>–</td>
<td>USING password</td>
<td>–</td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>–</td>
<td>Core</td>
<td>AREA area_name</td>
<td>–</td>
</tr>
<tr>
<td>CREATE PROCEDURE</td>
<td>–</td>
<td>Core</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>CREATE SYNONYM</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>✓</td>
<td>Minimum</td>
<td>AREA AS query_expression</td>
<td>–</td>
</tr>
<tr>
<td>CREATE TRIGGER</td>
<td>–</td>
<td>Core</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>CREATE USER</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>✓</td>
<td>Core</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DELETE</td>
<td>✓</td>
<td>Extended</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>–</td>
<td>Core</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DROP PROCEDURE</td>
<td>–</td>
<td>Core</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>
Table 2: Compliance of SQL-92 DDL and DML statements (2 of 2)

<table>
<thead>
<tr>
<th>OpenEdge SQL statement</th>
<th>SQL-92</th>
<th>ODBC SQL grammar</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP SYNONYM</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>–</td>
<td>Minimum</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>DROP TRIGGER</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>DROP USER</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>DROP VIEW</td>
<td>–</td>
<td>Core</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>GRANT</td>
<td>✓</td>
<td>Core</td>
<td>INDEX RESOURCE DBA</td>
<td>–</td>
</tr>
<tr>
<td>INSERT</td>
<td>✓</td>
<td>Core</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LOCK TABLE</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>REVOKE</td>
<td>✓</td>
<td>Core</td>
<td>INDEX RESOURCE DBA</td>
<td>–</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SELECT</td>
<td>✓</td>
<td>Extended</td>
<td>FOR UPDATE</td>
<td>–</td>
</tr>
<tr>
<td>SET CONNECTION</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SET SCHEMA</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SET TRANSACTION ISOLATION LEVEL</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>UPDATE</td>
<td>✓</td>
<td>Extended</td>
<td>assignments of form: ( column, column ) = ( expr, expr )</td>
<td>–</td>
</tr>
<tr>
<td>UPDATE STATISTICS</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
</tbody>
</table>
Syntax for ABL Attributes

The OpenEdge SQL statements `CREATE TABLE` and `ALTER TABLE` allow you to define ABL (Advanced Business Language) attributes for tables and columns. This section lists and describes the SQL keywords to use with `CREATE TABLE` and `ALTER TABLE` syntax. For examples of syntax using SQL keywords for ABL attributes, see the `CREATE TABLE` and `ALTER TABLE` entries in the “OpenEdge SQL Statements” section on page 17.

OpenEdge SQL keywords for ABL table attributes

Table 1 lists the keywords to use when setting ABL table attributes with OpenEdge SQL statements.

Table 1: ABL table attributes used in OpenEdge SQL statements

<table>
<thead>
<tr>
<th>Attribute keyword used in SQL statement</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PRO_CAN_CREATE</code></td>
<td>Equivalent to ABL CAN-CREATE</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td><code>PRO_CAN_DELETE</code></td>
<td>Equivalent to ABL CAN-DELETE</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td><code>PRO_CAN_DUMP</code></td>
<td>Equivalent to ABL CAN-DUMP</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td><code>PRO_CAN_LOAD</code></td>
<td>Equivalent to ABL CAN-LOAD</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td><code>PRO_CAN_READ</code></td>
<td>Equivalent to ABL CAN-READ</td>
<td>Arbitrary character string</td>
</tr>
</tbody>
</table>
### Table 1: ABL table attributes used in OpenEdge SQL statements (2 of 2)

<table>
<thead>
<tr>
<th>Attribute keyword used in SQL statement</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRO_CAN_WRITE</td>
<td>Equivalent to ABL CAN-WRITE.</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_DESCRIPTION</td>
<td>Equivalent to ABL DESCRIPTION.</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_DUMP_NAME</td>
<td>Equivalent to ABL DUMP-NAME.</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_FROZEN</td>
<td>Equivalent to ABL FROZEN.</td>
<td>‘Y’</td>
</tr>
</tbody>
</table>

**Note:** OpenEdge SQL honors the value set here and does not allow modification of a frozen table using the ALTER TABLE, CREATE INDEX, CREATE PRO Word INDEX, DROP INDEX, or DROP TABLE commands.

However, the frozen attribute may be set to ‘N’ to unfreeze a frozen table. For example:

```
ALTER TABLE Customer SET PRO_FROZEN ‘N’;
```

| PRO_HIDDEN                              | Indicates whether the table is shown in ABL tools and reports. | ‘Y’|’y’|’N’|’n’ |
| PRO_LABEL                               | Equivalent to ABL LABEL. | Arbitrary character string |
| PRO_VALEXP                              | Indicates an ABL validation expression. | Arbitrary character string |
| PRO_VALMSG                              | Indicates an ABL validation message. | Arbitrary character string |
| PRO_SA_VALMS                            | Indicates an ABL string attributes validation message. | Arbitrary character string |
| PRO_SA_LABEL                            | Indicates an ABL table label. | Arbitrary character string |
| PRO_DEFAULT_INDEX                       | Determines default data-access index for a table. | Name of an index or table |

Table 2 lists the keywords to use when setting ABL column attributes with OpenEdge SQL statements.
<table>
<thead>
<tr>
<th>Attribute keyword used in SQL statement</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRO_CAN_READ</td>
<td>Equivalent to ABL CAN-READ</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_CAN_WRITE</td>
<td>Equivalent to ABL CAN-WRITE</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_COL_LABEL</td>
<td>Equivalent to ABL COL-LABEL</td>
<td>Free-form text</td>
</tr>
<tr>
<td>PRO_DESCRIPTION</td>
<td>Equivalent to ABL DESCRIPTION</td>
<td>Free-form text</td>
</tr>
<tr>
<td>PRO_FORMAT</td>
<td>Equivalent to ABL FORMAT</td>
<td>ABL format string</td>
</tr>
<tr>
<td>PRO_HELP</td>
<td>Indicates a ABL help message</td>
<td>Free-form text</td>
</tr>
<tr>
<td>PRO_LABEL</td>
<td>Indicates ABL label</td>
<td>Free-form text</td>
</tr>
<tr>
<td>PRO_RPOS</td>
<td>Indicates ABL relative record position</td>
<td>Positive integer</td>
</tr>
<tr>
<td>PRO_SQL_WIDTH</td>
<td>Indicates SQL width</td>
<td>Positive integer</td>
</tr>
<tr>
<td>PRO_VIEW_AS</td>
<td>Equivalent to ABL VIEW-AS</td>
<td>Free-form text</td>
</tr>
<tr>
<td>PRO_ORDER</td>
<td>Equivalent to ABL ORDER</td>
<td>Integer value</td>
</tr>
<tr>
<td>PRO_VALEXP</td>
<td>Indicates ABL validation expression</td>
<td>Free-form text</td>
</tr>
<tr>
<td>PRO_VALMSG</td>
<td>Indicates ABL validation message</td>
<td>Free-form text</td>
</tr>
<tr>
<td>PRO_SA_LABEL</td>
<td>Indicates ABL string attribute column label</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_SA_COL_LABEL</td>
<td>Indicates ABL string attribute column label</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_SA_FORMAT</td>
<td>Indicates ABL string attribute column label</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_SA_INITIAL</td>
<td>Indicates ABL string attribute column label</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_SA_HELP</td>
<td>Indicates ABL string attribute column label</td>
<td>Arbitrary character string</td>
</tr>
<tr>
<td>PRO_SA_VALMSG</td>
<td>Indicates ABL string attribute column label</td>
<td>Arbitrary character string</td>
</tr>
</tbody>
</table>
### Table 3: ABL index attributes used in OpenEdge SQL statements

<table>
<thead>
<tr>
<th>Attribute keyword used in SQL statement</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRO_ACTIVE</td>
<td>Changes the index’s status from active to inactive. This action must be performed offline.</td>
<td>‘n’</td>
</tr>
<tr>
<td>PRO_DESCRIPTION</td>
<td>Equivalent to ABL DESCRIPTION.</td>
<td>Free-form text</td>
</tr>
</tbody>
</table>

**Note:** DEFAULT is a common attribute of both SQL and ABL.

Indicates the default value for a column. A literal value whose type is compatible with the type of the column.

Indicates case-sensitivity: ‘Y’ | ‘y’ | ‘N’ | ‘n’

The maximum size of a BLOB or CLOB column described as a string. Free-form text. For example: ‘32M’
Part 2

JDBC Reference

Java Class Reference

JDBC Conformance Notes
Java Class Reference

This section provides information on OpenEdge™ SQL Java classes and methods. The following subjects are covered:

- Java classes and methods
- DhSQLException
- DhSQLResultSet
- SQLCursor
- SQLIStatement
- SQLPStatement

Java classes and methods

This section provides reference material on the OpenEdge SQL Java classes and methods. This section lists all the methods in the OpenEdge SQL Java classes and shows which classes declare them. Subsequent sections are arranged alphabetically and describe each class and its methods in more detail. Some Java methods are common to more than one class.
setParam

Sets the value of an SQL statement’s input parameter to the specified value; a literal, procedure variable, or procedure input parameter. The following Java classes declare setParam:

- SQLIStatement
- SQLPStatement
- SQLCursor

makeNULL

Sets the value of an SQL statement’s input parameter to NULL. The following Java classes declare makeNULL:

- SQLIStatement
- SQLPStatement
- SQLCursor

Sets a field of the currently active row in a procedure’s result set to NULL:

- DhsSQLResultSet

execute

Executes the SQL statement. The following Java classes declare execute:

- SQLIStatement
- SQLPStatement

rowCount

Returns the number of rows deleted, inserted, or updated by the SQL statement. The following Java classes declare rowCount:

- SQLIStatement
- SQLPStatement
- SQLCursor

open

Opens the result set specified by the SELECT or CALL statement. The following Java class declares open:

- SQLCursor
close
Closes the result set specified by the SELECT or CALL statement. The following Java class declares close:

• SQLCursor

fetch
Fetches the next record in a result set. The following Java class declares fetch:

• SQLCursor

found
Checks whether a fetch operation returned to a record. The following Java class declares found:

• SQLCursor

wasNULL
Checks if the value in a fetched field is NULL. The following Java class declares wasNULL:

• SQLCursor

getValue
Stores the value of a fetched field in the specified procedure variable or procedure output parameter. The following Java class declares getValue:

• SQLCursor

set
Sets the field in the currently active row of a procedure’s result set a literal, procedure variable, or procedure input parameter. The following Java class declares set:

• DhsSQLResultSet

insert
Inserts the currently active row into the result set of a procedure. The following Java class declares insert:

• DhsSQLResultSet
**getDiagnostics**

Returns the specified detail of an error message. The following Java class declares `getDiagnostics`:

- `DhSQLException`

**log**

Writes a message to the log. The following Java classes inherit the log:

- `SQLIStatement`
- `SQLPStatement`
- `SQLCursor`
- `DhSQLResult Set`
- `DhSQLException`

**err**

Writes a message to the log. The following Java classes write to the log:

- `SQLIStatement`
- `SQLPStatement`
- `SQLCursor`
- `DhSQLResult Set`
- `DhSQLException`

**DhSQLException**

Extends the general `java.lang` exception class to provide detail about errors in SQL statement execution. Any such errors raise an exception with an argument that is an `SQLException` class object. The `getDiagnostics()` method retrieves details of the error.

**Constructors**

```java
public DhSQLException(int ecode, String errMsg)
```

**Parameters**

- `ecode`
  
  The error number associated with the exception condition.

- `errMsg`
  
  The error message associated with the exception condition.
In this example, the `DhSQLException` constructor creates an exception object called `excep` and then throws the `excep` object under all conditions:

```java
CREATE PROCEDURE sp1_02()
BEGIN
  // raising exception
  DhSQLException excep = new DhSQLException(666, new String("Entered the tst02 procedure"));
  if (true)
    throw excep;
END
```

### DhSQLException.getDiagnostics

Returns the requested detail about an exception.

#### Format

```java
public String getDiagnostics(int diagType)
```

#### Returns

A string containing the information specified by the `diagType` parameter, as shown in Table 1.

#### Parameters

- `diagType`

One of the argument values listed in Table 1.

#### Table 1: Argument values for DhSQLException.getDiagnostics

<table>
<thead>
<tr>
<th>Argument value</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURNED_SQLSTATE</td>
<td>The SQLSTATE returned by execution of the previous SQL statement</td>
</tr>
<tr>
<td>MESSAGE_TEXT</td>
<td>The condition indicated by RETURNED_SQLSTATE</td>
</tr>
<tr>
<td>CLASS_ORIGIN</td>
<td>Not currently used; always returns NULL</td>
</tr>
<tr>
<td>SUBCLASS_ORIGIN</td>
<td>Not currently used; always returns NULL</td>
</tr>
</tbody>
</table>

#### Throws

- `DhSQLException`
Example

This code fragment illustrates `DhSQLException.getDiagnostics`:

```java
try {
    SQLIStatement insert_cust = new SQLIStatement(
        "INSERT INTO customer VALUES (1,2) ");
} catch (DhSQLException e) {
    errstate = e.getDiagnostics(RETURNED_SQLSTATE); 
    errmsg = e.getDiagnostics(MESSAGE_TEXT); 
    ...
    ...
}
```

**DhSQLResultSet**

Provides the stored procedure with a result set to return to the application that called the procedure.

The Java code in a stored procedure does not explicitly create `DhSQLResultSet` objects. Instead, when the SQL server creates a Java class from a `CREATE PROCEDURE` statement that contains a Result clause, it implicitly instantiates an object of type `DhSQLResultSet`, and calls it `SQLResultSet`.

Procedures invoke methods of the `SQLResultSet` instance to populate fields and rows of the result set.

**Constructors**

No explicit constructor

**Parameters**

None

**Throws**

`DhSQLException`

**DhSQLResultSet.insert**

Inserts the currently active row into a procedure’s result set.

**Format**

```java
public void insert()
```

**Returns**

None
Parameters

None

Throws

DhSQLException

Example

This code fragment illustrates SQLResultSet.set and SQLResultSet.insert:

```
CREATE PROCEDURE get_sal2 ()
RESULT (
    empname CHAR(20),
    empsal  NUMERIC,
)
BEGIN
    String ename = new String (20) ;
    BigDecimal esal = new BigDecimal () ;
    SQLCursor empcursor = new SQLCursor (  
        "SELECT name, sal FROM emp " ) ;
    empcursor.Open () ;
    do
    {  
        empcursor.Fetch () ;
        if (empcursor.found () )
        {  
            empcursor.getValue (1, ename) ;
            empcursor.getValue (2, esal) ;
            SQLResultSet.Set (1, ename) ;
            SQLResultSet.Set (2, esal) ;
            SQLResultSet.Insert () ;
        }
    } while (empcursor.found () ) ;
    empcursor.close () ;
END
```

DhSQLResultSet.makeNULL

Sets a field of the currently active row in a procedure’s result set to NULL. This method is redundant with using the DhSQLResultSet.set method to set a procedure result-set field to NULL.

Format

```
public void makeNULL(int field)
```

Returns

None

Parameters

```
field
```

An integer that specifies which field of the result-set row to set to NULL. 1 denotes the first field in the row, 2 denotes the second, n denotes the nth.
**Throws**

DhSQLException

**Example**

This code fragment illustrates SQLResultSet.set and SQLResultSet.makeNULL:

```sql
CREATE PROCEDURE test_makeNULL2(
    IN char_in CHAR(20)
    RESULT ( res_char CHAR(20) , res_vchar VARCHAR(30))
BEGIN
    SQLResultSet.set(1, char_in);
    SQLResultSet.makeNULL(2);
END
```

**DhSQLResultSet.set**

Sets the field in the currently active row of a procedure’s result set to the specified value (a literal, procedure variable, or procedure input parameter).

**Format**

```java
public void set(int field, Object val)
```

**Returns**

None

**Parameters**

- **field**

  An integer that specifies which field of the result-set row to set to the value specified by `val`. (1 denotes the first field in the row, 2 denotes the second, and so on.)

- **val**

  A literal or the name of a variable or input parameter that contains the value to be assigned to the field.

**Throws**

DhSQLException
Example

This code fragment illustrates SQLResultSet.Set:

```
CREATE PROCEDURE get_sal2 ()
RESULT (  
   empname CHAR(20),  
   empsal   NUMERIC,
)
BEGIN
    String ename = new String (20) ;
    BigDecimal esal = new BigDecimal () ;
    SQLCursor empcursor = new SQLCursor (  
        "SELECT name, sal FROM emp " ) ;

    empcursor.Open () ;
    do
      {  
        empcursor.Fetch () ;
        if (empcursor.found ())
          {  
            empcursor.getValue (1, ename) ;
            empcursor.getValue (2, esal) ;
            SQLResultSet.Set (1, ename) ;
            SQLResultSet.Set (2, esal) ;
            SQLResultSet.Insert () ;
          }
      } while (empcursor.found ()) ;
    empcursor.close () ;  
END
```

SQLCursor

Allows rows of data to be retrieved from a database or another stored procedure’s result set.

Constructors

```
SQLCursor (String statement)
```

Parameters

```

statement
```

Generates a result set. Enclose the SQL statement in double quotes. The SQL statement is either a SELECT or CALL statement.

Notes

- A SELECT statement queries the database and returns data that meets the criteria specified by the query expression in the SELECT statement.

- A CALL statement invokes another stored procedure that returns a result set specified by the RESULT clause of the CREATE PROCEDURE statement.

Throws

```
DhSQLException
```
The following excerpt from a stored procedure instantiates an SQLCursor object called `cust_cursor` that retrieves data from a database table:

```java
SQLCursor emp_cursor = new SQLCursor ( "SELECT name, sal FROM emp " ) ;
```

The following excerpt from a stored procedure instantiates an SQLCursor object called `cust_cursor` that calls another stored procedure:

```java
t_cursor = new SQLCursor ( "CALL get_customers (?) ");
```

**SQLCursor.close**

Closes the result set specified by a SELECT or CALL statement.

**Format**

```
public void close()
```

**Returns**

None

**Parameters**

None

**Throws**

DhSQLException

**Example**

This code fragment illustrates the `getValue` and `close` methods:

```java
if (cust_cursor.Found ())
{
    cust_cursor.getValue (1, cust_number);
    cust_cursor.getValue (2, cust_name);
}
else
    break;

cust_cursor.close ( );
```
**SQLCursor.fetch**

Fetches the next record in a result set, if there is one.

**Format**

```java
public void fetch()
```

**Returns**

None

**Parameters**

None

**Throws**

DhSQLException

**Example**

This code fragment illustrates the `fetch` method and the `getValue` method:

```java
for (;;) {
    cust_cursor.Fetch ();
    if (cust_cursor.Found ()
    {  
        cust_cursor.getValue (1, cust_number);
        cust_cursor.getValue (2, cust_name);
    }  
    else  
    break;
}
```

**SQLCursor.found**

Checks whether a `fetch` operation returned a record.

**Format**

```java
public boolean found()
```

**Returns**

True if the previous call to `fetch()` returned a record, false otherwise

**Parameters**

None

**Throws**

DhSQLException
Example

This code fragment illustrates the fetch, found, and getValue methods:

```java
for (;;) {
    cust_cursor.Fetch ();
    if (cust_cursor.Found ())
    {
        cust_cursor.getValue (1, cust_number);
        cust_cursor.getValue (2, cust_name);
    }
    else
    break;
}
```

**SQLCursor.getParam**

Retrieves the values of Java OUT and INOUT parameters.

**Format**

```
inout_var = getParam( int fieldIndex, short fieldType );
```

**Returns**

**OUT or INOUT variable**

**Parameters**

- **inout_var**
  
  The target variable into which the value of an OUT or INOUT parameter is stored.

- **fieldIndex**
  
  An integer that specifies the position of the parameter in the parameter list.

- **fieldType**
  
  A short integer that specifies the data type of the parameter. The allowable defined values for fieldType are listed in Table 2, grouped by category of data type.
Table 2: Allowable values for fieldType in getParam

<table>
<thead>
<tr>
<th>Character</th>
<th>Exact numeric</th>
<th>Approximate numeric</th>
<th>Date-time</th>
<th>Bit string</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>INTEGER</td>
<td>REAL</td>
<td>DATE</td>
<td>BIT</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>SMALLINT</td>
<td>FLOAT</td>
<td>TIME</td>
<td>BINARY</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>TINYINT</td>
<td>DOUBLE</td>
<td>TIMESTAMP</td>
<td>VARBINARY</td>
</tr>
<tr>
<td>–</td>
<td>NUMERIC</td>
<td>–</td>
<td>–</td>
<td>LVARBINARY</td>
</tr>
<tr>
<td>–</td>
<td>DECIMAL</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Throws

DhSQLException

Notes

- The `getParam()` method returns the value of an INOUT or OUT parameter identified by the number you specify in the `fieldIndex` parameter. `getParam()` returns the value as an object of the data type you specify in the `fieldType` parameter. Since `getParam()` returns the result as an instance of class Object, you must explicitly cast your `inout_var` variable to the correct data type.

- If the OUT or INOUT parameter is of data type CHARACTER, then `getParam` returns a Java String Object. You must declare a procedure variable of type String, and explicitly cast the value returned by `getParam` to type String. Before calling `getParam()` you must call the SQLCursor.wasNULL method to test whether the returned value is NULL. If `getParam()` is called for a NULL value, it raises a DhSQLException.

**SQLCursor.getValue**

Assigns a single value from an SQL result set to a procedure variable. The single field value is the result of an SQL query or the result from another stored procedure.

Format

```java
public Object getValue( int fieldNum, short fieldType )
```

Returns

Object

Parameters

`fieldNum`

An integer that specifies the position of the field to retrieve from the fetched record.
**fieldType**

A short integer that specifies the data type of the parameter. The allowable defined values for `fieldType` are listed in Table 3, grouped by category of data type.

### Table 3: Allowable values for `fieldType` in `getValue`

<table>
<thead>
<tr>
<th>Character</th>
<th>Exact numeric</th>
<th>Approximate numeric</th>
<th>Date-time</th>
<th>Bit string</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>INTEGER</td>
<td>REAL</td>
<td>DATE</td>
<td>BIT</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>SMALLINT</td>
<td>FLOAT</td>
<td>TIME</td>
<td>BINARY</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>TINYINT</td>
<td>DOUBLE</td>
<td>TIMESTAMP</td>
<td>VARBINARY</td>
</tr>
<tr>
<td></td>
<td>NUMERIC</td>
<td></td>
<td></td>
<td>LVARBINARY</td>
</tr>
<tr>
<td></td>
<td>DECIMAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Throws**

DhSQLException

**Notes**

- Before invoking `getValue`, you must test for the NULL condition by calling the `SQLCursor.wasNULL` method. If the value returned is NULL, you must explicitly set the target variable in the stored procedure to NULL.

- The `getValue` method returns a value from the result set identified by the number you specify in the `fieldNum` parameter. `getValue` returns the value as an object of the data type you specify in the `fieldType` parameter. Since `getValue` returns the result as an instance of class Object, you must explicitly cast your return value to the correct data type.

- If the returned value is of data type CHARACTER, then `getValue` returns a Java String Object. You must declare a procedure variable of type String and explicitly cast the value returned by `getValue` to type String.
Example
This example illustrates testing for NULL and invoking the Java getValue method:

```java
Integer   pvar_int = new Integer(0);
String    pvar_str = new String();
SQLCursor select_t1 = new SQLCursor
    ("select int_col, char_col from T1");

Select_t1.open();
Select_t1.fetch();

while(select_t1.found())
{
    // Assign values from the current row of the SQL result set
    // to the procedure variables. First check whether
    // the values fetched are null. If null then explicitly
    // set the procedure variables to null.

    if ((select_t1.wasNULL(1)) == true)
        pvar_int = null;
    else
        pvar_int = (Integer)select_t1.getValue(1, INTEGER);

    if ((select_t1.wasNULL(2)) == true)
        pvar_str = null;
    else
        pvar_str = (String)select_t1.getValue(1, CHAR);
}
```

**SQLCursor.makeNULL**

Sets the value of an SQL statement's input parameter to NULL. This method is common to the SQLCursor, SQLIStatement, and SQLPStatement classes. This method is redundant with using the setParam method to set an SQL statement's input parameter to NULL.

**Format**

```
public void makeNULL(int f)
```

**Returns**

None

**Parameters**

`f`

An integer that specifies which input parameter of the SQL statement string to set to NULL. 1 denotes the first input parameter in the statement, 2 denotes the second, n denotes the nth.

**Throws**

DhSQLException
**Example** This code fragment illustrates the `makeNULL` method:

```java
CREATE PROCEDURE sc_makeNULL()
BEGIN
  SQLCursor select_btypes = new SQLCursor (
    "SELECT small_fld from sfns where small_fld = ? ");
  select_btypes.makeNULL(1);
  select_btypes.open();
  select_btypes.fetch();
  .
  .
  select_btypes.close();
END
```

**SQLCursor.open**

Opens the result set specified by the `SELECT` or `CALL` statement.

**Format**

```java
public void open()
```

**Returns**

None

**Parameters**

None

**Throws**

DhSQLException

**Example** This code fragment illustrates the `open` method:

```java
SQLCursor empcursor = new SQLCursor ( "SELECT name, sal FROM emp " ) ;
empcursor.Open () ;
```

**SQLCursor.registerOutParam**

Registers `OUT` parameters.

**Format**

```java
registerOutParam( int fieldIndex, short fieldType [, short scale ] )
```
Returns

None

Parameters

fieldIndex

An integer that specifies the position of the parameter in the parameter list.

fieldType

A short integer that specifies the data type of the parameter.

The allowable defined values for fieldType are listed in Table 4, grouped by category of data type.

Table 4: Allowable values for fieldType in registerOutParam

<table>
<thead>
<tr>
<th>Character</th>
<th>Exact numeric</th>
<th>Approximate numeric</th>
<th>Date-time</th>
<th>Bit string</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>INTEGER</td>
<td>REAL</td>
<td>DATE</td>
<td>BIT</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>SMALLINT</td>
<td>FLOAT</td>
<td>TIME</td>
<td>BINARY</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>TINYINT</td>
<td>DOUBLE</td>
<td>TIMESTAMP</td>
<td>VARBINARY</td>
</tr>
<tr>
<td>¬</td>
<td>NUMERIC</td>
<td>¬</td>
<td>¬</td>
<td>LVARBINARY</td>
</tr>
<tr>
<td>¬</td>
<td>DECIMAL</td>
<td>¬</td>
<td>¬</td>
<td>¬</td>
</tr>
</tbody>
</table>

Throws

DhSQLException

SQLCursor.rowCount

Returns the number of rows affected (selected, inserted, updated, or deleted) by the SQL statement. This method is common to the SQLCursor, SQLIStatement, and SQLPStatement classes.

Format

public int rowCount()

Returns

An integer indicating the number of rows.

Parameters

None
Throws

DhSQLException

Example
This example uses the rowCount method of the SQLIStatement class by nesting the method invocation within SQLResultSet.set to store the number of rows affected (1, in this case) in the procedure’s result set:

```java
CREATE PROCEDURE sis_rowCount()
RESULT ( ins_recs INTEGER )
BEGIN
    SQLCursor insert_test103 = new SQLIStatement ( "INSERT INTO test103 (fld1) values (17)" );
    insert_test103.execute();
    SQLResultSet.set(1,new Long(insert_test103.rowCount()));
    SQLResultSet.insert();
END
```

**SQLCursor.setParam**

Sets the value of an SQL statement’s input parameter to the specified value (a literal, procedure variable, or procedure input parameter). This method is common to the SQLCursor, SQLIStatement, and SQLPStatement classes.

Format

```
public void setParam(int f, Object val)
```

Returns

None

Parameters

- **f**
  
  An integer that specifies which parameter marker in the SQL statement is to receive the value. 1 denotes the first parameter marker, 2 denotes the second, **n** denotes the **n**th.

- **val**
  
  A literal or the name of a variable or input parameter that contains the value to be assigned to the parameter marker.

Throws

DhSQLException
Example

This code fragment illustrates the `setParam` method:

```java
CREATE PROCEDURE sps_setParam()
BEGIN
  // Assign local variables to be used as SQL input parameter references
  Integer ins_fld_ref   = new Integer(1);
  Integer ins_small_fld = new Integer(3200);
  Integer ins_int_fld   = new Integer(21474);
  Double  ins_doub_fld  = new Double(1.797E+30);
  String  ins_char_fld  = new String("Athula");
  String  ins_vchar_fld = new String("Scientist");
  Float   ins_real_fld  = new Float(17);
  SQLIStatement insert_sfns1 = new SQLIStatement ("INSERT INTO sfns
  (fld_ref,small_fld,int_fld,doub_fld,char_fld,vchar_fld)
  values (??,?, ?,?,?, ?)");
  insert_sfns1.setParam(1,ins_fld_ref);
  insert_sfns1.setParam(2,ins_small_fld);
  insert_sfns1.setParam(3,ins_int_fld);
  insert_sfns1.setParam(4,ins_doub_fld);
  insert_sfns1.setParam(5,ins_char_fld);
  insert_sfns1.setParam(6,ins_vchar_fld);
  insert_sfns1.execute();
END
```

**SQLCursor.wasNULL**

Checks if the value in a fetched field is **NULL**.

**Format**

```
public boolean wasNULL(int field)
```

**Returns**

True if the field is **NULL**, false otherwise

**Parameters**

`field`

An integer that specifies which field of the fetched record is of interest. (1 denotes the first column of the result set, 2 denotes the second, and so on.)

`wasNULL` checks whether the value in the currently fetched record of the column denoted by `field` is **NULL**.

**Throws**

DhSQLException
Example

This code fragment illustrates the `wasNULL` method:

```java
CREATE PROCEDURE test_wasNULL()
BEGIN
  int small_sp = 0;
  SQLCursor select_btypes =
    new SQLCursor("SELECT small_fld from sfns");
  select_btypes.open();
  select_btypes.fetch();
  if ((select_btypes.wasNULL(1)) == true)
    small_sp = null;
  else
    select_btypes.getValue(1, small_sp);
  select_btypes.close();
END
```

`SQLIStatement`

Allows immediate (one-time) execution of SQL statements that do not generate a result set.

Constructors

`SQLIStatement(String statement)`

Parameters

`statement`

An SQL statement that does not generate a result set. Enclose the SQL statement in double quotes.

Throws

`DhSQLException`

Example

This code fragment illustrates the `SQLIStatement` class:

```java
CREATE PROCEDURE insert_customer (
  IN  cust_number INTEGER,
  IN  cust_name   CHAR(20)
)
BEGIN
  SQLIStatement insert_cust = new SQLIStatement ("INSERT INTO customer VALUES (?,?)");
END
```
**SQLIStatement.execute**

Executes the SQL statement. This method is common to the `SQLIStatement` and `SQLPStatement` classes.

**Format**

```java
public void execute()
```

**Returns**

None

**Parameters**

None

**Throws**

`DhSQLException`

**Example**

This code fragment illustrates the `setParam` and `execute` methods:

```java
CREATE PROCEDURE insert_customer {
IN cust_number INTEGER,
IN cust_name CHAR(20)
}
BEGIN
    SQLIStatement insert_cust = new SQLIStatement(
        "INSERT INTO customer VALUES (?,?) "
    );
    insert_cust.setParam (1, cust_number);
    insert_cust.setParam (2, cust_name);
    insert_cust.execute();
END
```

**SQLIStatement.makeNULL**

Sets the value of an SQL statement's input parameter to `NULL`. This method is common to the `SQLCursor`, `SQLIStatement`, and `SQLPStatement` classes. This method is redundant with using the `setParam` method to set an SQL statement's input parameter to `NULL`.

**Format**

```java
public void makeNULL(int f)
```

**Returns**

None
Parameters

\( f \)

An integer that specifies which input parameter of the SQL statement string to set to NULL. 1 denotes the first input parameter in the statement, 2 denotes the second, \( n \) denotes the \( n \)th.

Throws

DhSQLException

Example

This code fragment illustrates the makeNULL method:

```
CREATE PROCEDURE sis_makeNULL()
BEGIN
    SQLIStatement insert_sfns1 = new SQLIStatement ("INSERT INTO sfns
        (fld_ref,small_fld,int_fld,doub_fld,char_fld,vchar_fld)
        values (?,?,?,?,?,?)");
    insert_sfns1.setParam1(new Integer(66));
    insert_sfns1.makeNULL(2);
    insert_sfns1.makeNULL(3);
    insert_sfns1.makeNULL(4);
    insert_sfns1.makeNULL(5);
    insert_sfns1.execute();
END
```

**SQLIStatement.rowCount**

Returns the number of rows affected (selected, inserted, updated, or deleted) by the SQL statement. This method is common to the SQLCursor, SQLIStatement, and SQLPStatement classes.

Format

```
public int rowCount()
```

Returns

An integer indicating the number of rows

Parameters

None

Throws

DhSQLException
Example

This example uses the `rowCount` method of the `SQLIStatement` class by nesting the method invocation within `SQLResultSet.set` to store the number of rows affected (1, in this case) in the procedure’s result set:

```java
CREATE PROCEDURE sis_rowCount()
RESULT ( ins_recs INTEGER )
BEGIN
    SQLIStatement insert_test103 = new SQLIStatement ( "INSERT INTO test103 (fld1) values (17)" );
    insert_test103.execute();
    SQLResultSet.set(1,new Long(insert_test103.rowCount()));
    SQLResultSet.insert();
END
```

### SQLIStatement.setParam

Sets the value of an SQL statement’s input parameter to the specified value (a literal, procedure variable, or procedure input parameter). This method is common to the `SQLCursor`, `SQLIStatement`, and `SQLPStatement` classes.

**Format**

```java
public void setParam(int f, Object val)
```

**Returns**

None

**Parameters**

- `f`
  - An integer that specifies which parameter marker in the SQL statement is to receive the value (1 denotes the first parameter marker, 2 denotes the second, and so on).

- `val`
  - A literal or the name of a variable or input parameter that contains the value to be assigned to the parameter marker.

**Throws**

- `DhSQLException`
Example  This code fragment illustrates the setParam method:

```java
CREATE PROCEDURE sps_setParam()
BEGIN
  // Assign local variables to be used as SQL input parameter references
  Integer ins_fld_ref = new Integer(1);
  Integer ins_small_fld = new Integer(3200);
  Integer ins_int_fld = new Integer(21474);
  Double ins_doub_fld = new Double(1.797E+30);
  String ins_char_fld = new String("Athula");
  String ins_vchar_fld = new String("Scientist");
  Float ins_real_fld = new Float(17);

  SQLPStatement insert_sfns1 = new SQLPStatement ("INSERT INTO sfns
    (fld_ref,small_fld,int_fld,doub_fld,char_fld,vchar_fld)
    values (?,?,?,?,?,?)");

  insert_sfns1.setParam(1,ins_fld_ref);
  insert_sfns1.setParam(2,ins_small_fld);
  insert_sfns1.setParam(3,ins_int_fld);
  insert_sfns1.setParam(4,ins_doub_fld);
  insert_sfns1.setParam(5,ins_char_fld);
  insert_sfns1.setParam(6,ins_vchar_fld);
  insert_sfns1.execute();
END
```

SQLPStatement

Allows prepared (repeated) execution of SQL statements that do not generate a result set.

 Constructors

`SQLPStatement (String statement)`

Parameters

`statement`

An SQL statement that does not generate a result set. Enclose the SQL statement in double quotes.

 Throws

`DhSQLException`
Example

This code fragment illustrates the `SQLPStatement` class:

```java
SQLPStatement pstmt = new SQLPStatement ( "INSERT INTO T1 VALUES (?, ?) ");
pstmt.setParam (1, 10);
pstmt.setParam (2, 10);
pstmt.execute ();
pstmt.setParam (1, 20);
pstmt.setParam (2, 20);
pstmt.execute ();
```

**SQLPStatement.execute**

Executes the SQL statement. This method is common to the `SQLIStatement` and `SQLPStatement` classes.

**Format**

```java
public void execute()
```

**Returns**

None

**Parameters**

None

**Throws**

`DhSQLException`

Example

This code fragment illustrates the `execute` and `setParam` methods in the `SQLPStatement` class:

```java
SQLPStatement pstmt = new SQLPStatement ( "INSERT INTO T1 VALUES (?, ?) ");
pstmt.setParam (1, 10);
pstmt.setParam (2, 10);
pstmt.execute ();
pstmt.setParam (1, 20);
pstmt.setParam (2, 20);
pstmt.execute ();
```

**SQLPStatement.makeNULL**

Sets the value of an SQL statement’s input parameter to `NULL`. This method is common to the `SQLCursor`, `SQLIStatement`, and `SQLPStatement` classes. This method is redundant with using the `setParam` method to set an SQL statement’s input parameter to `NULL`.

**Format**

```java
public void makeNULL(int f)
```

**Returns**

None
Parameters

\( f \)

An integer that specifies which input parameter of the SQL statement string to set to NULL. (1 denotes the first input parameter in the statement, 2 denotes the second, and so on.)

Throws

DhSQLException

Example

This code fragment illustrates `SQLPStatement.makeNULL`:

```java
CREATE PROCEDURE sps_makeNULL()
BEGIN
    SQLPStatement insert_sfns1 = new SQLPStatement("INSERT INTO sfns
        (fld_ref,small_fld,int_fld,doub_fld,char_fld,vchar_fld)
        values (?,?,?,?,?,?)" );
    insert_sfns1.setParam(1,new Integer(666));
    insert_sfns1.makeNULL(2);
    insert_sfns1.makeNULL(3);
    insert_sfns1.makeNULL(4);
    insert_sfns1.makeNULL(5);
    insert_sfns1.execute();
END
```

`SQLPStatement.rowCount`

Returns the number of rows affected (selected, inserted, updated, or deleted) by the SQL statement. This method is common to the `SQLCursor`, `SQLIStatement`, and `SQLPStatement` classes.

Format

```
public int rowCount()
```

Returns

An integer indicating the number of rows

Parameters

None

Throws

DhSQLException
Example
This example uses the `rowCount` method of the `SQLPStatement` class by nesting the method invocation within `SQLResultSet.set` to store the number of rows affected (1, in this case) in the procedure’s result set:

```java
CREATE PROCEDURE sis_rowCount()
RESULT ( ins_recs INTEGER )
BEGIN
    SQLPStatement insert_test103 = new SQLPStatement ( "INSERT INTO test103 (fld1) values (17)" );
    insert_test103.execute();
    SQLResultSet.set(1,new Long(insert_test103.rowCount()));
    SQLResultSet.insert();
END
```

### SQLPStatement.setParam

Sets the value of an SQL statement’s input parameter to the specified value (a literal, procedure variable, or procedure input parameter). This method is common to the `SQLCursor`, `SQLIStatement`, and `SQLPStatement` classes.

**Format**

```java
public void setParam(int f, Object val)
```

**Returns**

None

**Parameters**

- `f`

  An integer that specifies which parameter marker in the SQL statement is to receive the value (1 denotes the first parameter marker, 2 denotes the second, and so on).

- `val`

  A literal or the name of a variable or input parameter that contains the value to be assigned to the parameter marker.

**Throws**

- `DhSQLException`
**Example**  This code fragment illustrates `SQLPStatement.setParam`:

```java
CREATE PROCEDURE sps_setParam()
BEGIN
  // Assign local variables to be used as SQL input parameter references
  Integer ins_fld_ref   = new Integer(1);
  Integer ins_small_fld = new Integer(3200);
  Integer ins_int_fld   = new Integer(21474);
  Double  ins_doub_fld  = new Double(1.797E+30);
  String  ins_char_fld  = new String("Athula");
  String  ins_vchar_fld = new String("Scientist");
  Float   ins_real_fld  = new Float(17);
  SQLPStatement insert_sfns1 = new SQLPStatement ("INSERT INTO sfns
(fld_ref,small_fld,int_fld,doub_fld,char_fld,vchar_fld)
values (?,?,?,?,?,?)");
  insert_sfns1.setParam(1,ins_fld_ref);
  insert_sfns1.setParam(2,ins_small_fld);
  insert_sfns1.setParam(3,ins_int_fld);
  insert_sfns1.setParam(4,ins_doub_fld);
  insert_sfns1.setParam(5,ins_char_fld);
  insert_sfns1.setParam(6,ins_vchar_fld);
  insert_sfns1.execute();
END
```
JDBC Conformance Notes

This section details the DataDirect JDBC driver’s support for the JDBC standard. Information presented in this section includes:

- Supported data types
- Return values for DatabaseMetaData

Supported data types

The Data Direct JDBC Driver supports standard JDBC mapping of JDBC data types to corresponding Java data types.

In the JDBC methods `CallableStatement.getXXX` and `PreparedStatement.setXXX` methods, `XXX` is a Java type:

- For `setXXX` methods, the driver converts the Java data type to the JDBC data type shown in Table 1 before sending it to the database.

Table 1 provides details on data type mapping between Java and JDBC data types.

<table>
<thead>
<tr>
<th>Java data type</th>
<th>JDBC data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>BIT</td>
</tr>
<tr>
<td>Byte</td>
<td>TINYINT</td>
</tr>
<tr>
<td>byte [ ]</td>
<td>LONGVARBINARY</td>
</tr>
<tr>
<td>byte [ ]</td>
<td>VARBINARY</td>
</tr>
</tbody>
</table>
For `getXXX` methods, the driver converts the JDBC data type returned by the database to the Java data type shown in Table 2, Table 3, and Table 4 before returning it to the `getXXX` method.

Table 2 details mapping between JDBC and Java data types.

**Table 1:  Mapping between Java and JDBC data types** *(2 of 2)*

<table>
<thead>
<tr>
<th>Java data type</th>
<th>JDBC data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>float</td>
<td>REAL</td>
</tr>
<tr>
<td>Int</td>
<td>INTEGER</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>NUMERIC, DECIMAL</td>
</tr>
<tr>
<td>java.sql.Date</td>
<td>DATE</td>
</tr>
<tr>
<td>java.sql.Time</td>
<td>TIME</td>
</tr>
<tr>
<td>java.sql.Timestamp</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Short</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>String</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>

**Table 2:  Mapping between JDBC and Java data types**

<table>
<thead>
<tr>
<th>JDBC data type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>boolean</td>
</tr>
<tr>
<td>CHAR</td>
<td>String</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
</tr>
<tr>
<td>TINYINT</td>
<td>byte</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>String</td>
</tr>
</tbody>
</table>

**Table 3** details mapping between SQL-92 and Java data types.

**Table 3:  Mapping between SQL-92 and Java data types** *(1 of 2)*

<table>
<thead>
<tr>
<th>SQL-92 data type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>BIT</td>
<td>boolean</td>
</tr>
<tr>
<td>CHAR, VARCHAR</td>
<td>String</td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
</tr>
</tbody>
</table>
Table 3: Mapping between SQL-92 and Java data types (2 of 2)

<table>
<thead>
<tr>
<th>SQL-92 data type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIMAL</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>Double</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Float</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
</tr>
<tr>
<td>LONGVARBINARY</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>REAL</td>
<td>Float</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
</tr>
<tr>
<td>TIME</td>
<td>java.sql.Time</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>java.sql.Timestamp</td>
</tr>
<tr>
<td>TINYINT</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>byte[ ]</td>
</tr>
</tbody>
</table>

Table 4 provides information on JDBC data type conversion.

Table 4: JDBC data type conversion (1 of 2)

<table>
<thead>
<tr>
<th>JDBC data type</th>
<th>Converts to . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>CHAR, DOUBLE, FLOAT, INTEGER, SMALLINT, TINYINT</td>
</tr>
<tr>
<td>BINARY</td>
<td>Does not convert to any other data type</td>
</tr>
<tr>
<td>BIT</td>
<td>Does not convert to any other data type</td>
</tr>
<tr>
<td>CHAR</td>
<td>BIGINT, DATE, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT, VARCHAR</td>
</tr>
<tr>
<td>DATE</td>
<td>CHAR, TIMESTAMP, VARCHAR</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>BIGINT, CHAR, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, TINYINT, VARCHAR</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>BIGINT, CHAR, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, TINYINT, VARCHAR</td>
</tr>
<tr>
<td>FLOAT</td>
<td>BIGINT, CHAR, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, TINYINT, VARCHAR</td>
</tr>
<tr>
<td>INTEGER</td>
<td>BIGINT, CHAR, DECIMAL, DOUBLE, FLOAT, NUMERIC, REAL, SMALLINT, TINYINT, VARCHAR</td>
</tr>
</tbody>
</table>
### Table 4: JDBC data type conversion

<table>
<thead>
<tr>
<th>JDBC data type</th>
<th>Converts to . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONGVARBINARY</td>
<td>Does not convert to any other data type</td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>Does not convert to any other data type</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>BIGINT, CHAR, DECIMAL, DOUBLE, FLOAT, INTEGER, REAL, SMALLINT, TINYINT, VARCHAR</td>
</tr>
<tr>
<td>REAL</td>
<td>BIGINT, CHAR, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, SMALLINT, TINYINT, VARCHAR</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>BIGINT, CHAR, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, TINYINT, VARCHAR</td>
</tr>
<tr>
<td>TIME</td>
<td>CHAR, TIMESTAMP</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>CHAR, DATE, TIME, VARCHAR</td>
</tr>
<tr>
<td>TINYINT</td>
<td>BIGINT, CHAR, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, VARCHAR</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>Does not convert to any other data type</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>BIGINT, CHAR, DATE, DECIMAL, DOUBLE, FLOAT, INTEGER, NUMERIC, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT</td>
</tr>
</tbody>
</table>

---

### Return values for DatabaseMetaData

Applications call methods of the DatabaseMetaData class to retrieve details about the JDBC support provided by the OpenEdge™ SQL JDBC driver.

*Table 5* lists each method of the DatabaseMetaData class and shows what the JDBC driver returns when an application calls the method. For details on the format and usage of each method, see the Java Core API documentation for your platform. Many of the methods return lists of information as an object of type ResultSet. Use the normal ResultSet methods, such as `getString` and `getInt`, to retrieve the data from the result sets.
### Table 5: Return values for DatabaseMetaData methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>allProceduresAreCallable()</td>
<td>Can all the procedures returned by <code>getProcedures</code> be called by the current user?</td>
<td>False</td>
</tr>
<tr>
<td>allTablesAreSelectable()</td>
<td>Can all the tables returned by <code>getTable</code> be <code>SELECTed</code> by the current user?</td>
<td>False</td>
</tr>
<tr>
<td>dataDefinitionCausesTransactionCommit()</td>
<td>Does a data definition statement within a transaction force the transaction to commit?</td>
<td>True</td>
</tr>
<tr>
<td>dataDefinitionIgnoredInTransactions()</td>
<td>Is a data definition statement within a transaction ignored?</td>
<td>False</td>
</tr>
<tr>
<td>doesMaxRowSizeIncludeBlobs()</td>
<td>Did <code>getMaxRowSize()</code> include <code>LONGVARCHAR</code> and <code>LONGVARBINARY</code> BLOBs?</td>
<td>False</td>
</tr>
<tr>
<td>getBestRowIdentifier(String, String, String, int, boolean)</td>
<td>Gets a description of a table's optimal set of columns that uniquely identifies a row.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getCatalogs()</td>
<td>Gets the catalog names available in this database.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getCatalogSeparator()</td>
<td>What is the separator between catalog and table names?</td>
<td>None</td>
</tr>
<tr>
<td>getCatalogTerm()</td>
<td>What is the database vendor's preferred term for catalog?</td>
<td>None</td>
</tr>
<tr>
<td>getColumnPrivileges(String, String, String)</td>
<td>Gets a description of the access rights for a table's columns.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getColumns(String, String, String, String)</td>
<td>Gets a description of table columns available in a catalog.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>getCrossReference(String, String, String, String, String, String)</code></td>
<td>Gets a description of the foreign key columns in the foreign key table that reference the primary key columns of the primary key table (describes how one table imports another's key). This should normally return a single foreign key/primary key pair (most tables only import a foreign key from a table once). They are ordered by <code>FKTABLE_CAT</code>, <code>FKTABLE_SCHEM</code>, <code>FKTABLE_NAME</code>, and <code>KEY_SEQ</code>.</td>
<td>(Result set)</td>
</tr>
<tr>
<td><code>getDatabaseProductName()</code></td>
<td>What is the name of this database product?</td>
<td>OPENEDGE</td>
</tr>
<tr>
<td><code>getDatabaseProductVersion()</code></td>
<td>What is the version of this database product?</td>
<td>10.0A1B</td>
</tr>
<tr>
<td><code>getDefaultTransactionIsolation()</code></td>
<td>What is the database's default transaction isolation level? The values are defined in <code>java.sql.Connection</code>.</td>
<td>TRANSACTION_READ_COMMITTED</td>
</tr>
<tr>
<td><code>getDriverMajorVersion()</code></td>
<td>What is the version of this JDBC driver?</td>
<td>1</td>
</tr>
<tr>
<td><code>getDriverMinorVersion()</code></td>
<td>What is the minor version of this JDBC driver?</td>
<td>1000</td>
</tr>
<tr>
<td><code>getDriverName()</code></td>
<td>What is the name of this JDBC driver?</td>
<td>OpenEdge</td>
</tr>
<tr>
<td><code>getDriverVersion()</code></td>
<td>What is the version of this JDBC driver?</td>
<td>4.0.00 5805 (040318.014802)</td>
</tr>
<tr>
<td><code>getExportedKeys(String, String, String)</code></td>
<td>Gets a description of the foreign key columns that reference a table's primary key columns (the foreign keys exported by a table).</td>
<td>(Result set)</td>
</tr>
<tr>
<td><code>getExtraNameCharacters()</code></td>
<td>Gets all the extra characters that can be used in unquoted identifier names (those beyond <code>a-z, A-Z, 0-9</code> and <code>_</code>).</td>
<td>&quot;&quot;, &quot;%&quot;</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>getIdentifierQuoteString ()</td>
<td>What is the string used to quote SQL identifiers? This returns a space &quot; &quot; if identifier quoting is not supported.</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>getImportedKeys(String, String, String)</td>
<td>Gets a description of the primary key columns that reference a table's foreign key columns (the primary keys imported by a table).</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getIndexInfo(String, String, String, boolean, boolean)</td>
<td>Gets a description of a table's indices and statistics.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getMaxBinaryLiteralLength()</td>
<td>How many hex characters can you have in an inline binary literal?</td>
<td>31983</td>
</tr>
<tr>
<td>getMaxCatalogNameLength()</td>
<td>What is the maximum length of a catalog name?</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No catalogs</td>
</tr>
<tr>
<td>getMaxCharLiteralLength()</td>
<td>What is the maximum length for a character literal?</td>
<td>31983</td>
</tr>
<tr>
<td>getMaxColumnNameLength()</td>
<td>What is the limit on column name length?</td>
<td>32</td>
</tr>
<tr>
<td>getMaxColumnsInGroupBy()</td>
<td>What is the maximum number of columns in a GROUP BY clause?</td>
<td>499</td>
</tr>
<tr>
<td>getMaxColumnsInIndex()</td>
<td>What is the maximum number of columns allowed in an index?</td>
<td>16</td>
</tr>
<tr>
<td>getMaxColumnsInOrderBy()</td>
<td>What is the maximum number of columns in an ORDER BY clause?</td>
<td>0</td>
</tr>
<tr>
<td>getMaxColumnsInSelect()</td>
<td>What is the maximum number of columns in a SELECT list?</td>
<td>500</td>
</tr>
<tr>
<td>getMaxColumnsInTable()</td>
<td>What is the maximum number of columns in a table?</td>
<td>500</td>
</tr>
<tr>
<td>getMaxConnections()</td>
<td>How many active connections can we have at a time to this database?</td>
<td>0</td>
</tr>
<tr>
<td>getMaxCursorNameLength()</td>
<td>What is the maximum cursor name length?</td>
<td>18</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>getMaxIndexLength()</td>
<td>What is the maximum length of an index (in bytes)?</td>
<td>113</td>
</tr>
<tr>
<td>getMaxProcedureNameLength()</td>
<td>What is the maximum length of a procedure name?</td>
<td>32</td>
</tr>
<tr>
<td>getMaxRowSize()</td>
<td>What is the maximum length of a single row?</td>
<td>31,995 bytes</td>
</tr>
<tr>
<td>getMaxSchemaNameLength()</td>
<td>What is the maximum length allowed for a schema name?</td>
<td>32</td>
</tr>
<tr>
<td>getMaxStatementLength()</td>
<td>What is the maximum length of an SQL statement?</td>
<td>131,000</td>
</tr>
<tr>
<td>getMaxStatements()</td>
<td>How many active statements can we have open at one time to this database?</td>
<td>100</td>
</tr>
<tr>
<td>getMaxTableNameLength()</td>
<td>What is the maximum length of a table name?</td>
<td>32</td>
</tr>
<tr>
<td>getMaxTablesInSelect()</td>
<td>What is the maximum number of tables in a SELECT?</td>
<td>250</td>
</tr>
<tr>
<td>getMaxUserNameLength()</td>
<td>What is the maximum length of a user name?</td>
<td>32</td>
</tr>
<tr>
<td>getNumericFunctions()</td>
<td>Gets a comma-separated list of math functions.</td>
<td>ABS, ACOS, ASIN, ATAN, ATAN2, CEILING, COS, DEGREES, EXP, FLOOR, LOG10, MOD, PI, POWER, RADIANS, RAND, ROUND, SIGN, SIN, SQRT, TAN</td>
</tr>
<tr>
<td>getPrimaryKeys(String, String, String)</td>
<td>Gets a description of a table’s primary key columns.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getProcedureColumns(String, String, String)</td>
<td>Get a description of a catalog’s stored procedure parameters and result columns.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getProcedures(String, String, String)</td>
<td>Gets a description of stored procedures available in a catalog.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getProcedureTerm()</td>
<td>What is the database vendor’s preferred term for procedure?</td>
<td>procedure</td>
</tr>
<tr>
<td>getSchemas()</td>
<td>Gets the schema names available in this database.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getSchemaTerm()</td>
<td>What is the database vendor’s preferred term for schema?</td>
<td>Owner</td>
</tr>
<tr>
<td>getSearchStringEscape()</td>
<td>This is the string that can be used to escape ‘_’ or ‘%’ in the string pattern style catalog search parameters.</td>
<td>\.</td>
</tr>
<tr>
<td>getSQLKeywords()</td>
<td>Gets a comma-separated list of all a database’s SQL keywords that are NOT also SQL keywords.</td>
<td>See the OpenEdge SQL Reserved Words section for a complete list of reserved words.</td>
</tr>
<tr>
<td>getStringFunctions()</td>
<td>Gets a comma-separated list of string functions.</td>
<td>ASCII, CHAR, CONCAT, DIFFERENCE, INSERT, LCASE, LEFT, LENGTH, LOCATE, LOCATE-2, LTRIM, REPEAT, REPLACE, RIGHT, RTRIM, SPACE, SUBSTRING, UCASE</td>
</tr>
<tr>
<td>getSystemFunctions()</td>
<td>Gets a comma-separated list of system functions.</td>
<td>USERNAME, IFNULL, DBNAME</td>
</tr>
<tr>
<td>getTablePrivileges(String, String, String)</td>
<td>Gets a description of the access rights for each table available in a catalog.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getTables(String, String, String, String [])</td>
<td>Gets a description of tables available in a catalog.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getTableTypes()</td>
<td>Gets the table types available in this database.</td>
<td>(Result set)</td>
</tr>
<tr>
<td>getTimeDateFunctions()</td>
<td>Gets a comma-separated list of time and date functions.</td>
<td>CURDATE, CURTIME, DAYNAME, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, MONTH, QUARTER, WEEK, YEAR, HOUR, MINUTE, SECOND, MONTNAME, NOW, TIMESTAMPADD, TIMESTAMPDIFF</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>getTypeInfo()</code></td>
<td>Gets a description of all the standard SQL types supported by this database.</td>
<td>(Result set)</td>
</tr>
<tr>
<td><code>getURL()</code></td>
<td>What is the URL for this database?</td>
<td>(The URL)</td>
</tr>
<tr>
<td><code>getUserName()</code></td>
<td>What is our user name as known to the database?</td>
<td>(User name)</td>
</tr>
<tr>
<td><code>getVersionColumns(String, String, String)</code></td>
<td>Gets a description of a table’s columns that are automatically updated when any value in a row is updated.</td>
<td>(Result set)</td>
</tr>
<tr>
<td><code>isCatalogAtStart()</code></td>
<td>Does a catalog appear at the start of a qualified table name? Otherwise it appears at the end.</td>
<td>False</td>
</tr>
<tr>
<td><code>isReadOnly()</code></td>
<td>Is the database in read-only mode?</td>
<td>False</td>
</tr>
<tr>
<td><code>nullPlusNonNullIsNull()</code></td>
<td>Are concatenations between NULL and non-NULL values NULL? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td><code>nullsAreSortedAtEnd()</code></td>
<td>Are NULL values sorted at the end regardless of sort order?</td>
<td>False</td>
</tr>
<tr>
<td><code>nullsAreSortedAtStart()</code></td>
<td>Are NULL values sorted at the start regardless of sort order?</td>
<td>False</td>
</tr>
<tr>
<td><code>nullsAreSortedHigh()</code></td>
<td>Are NULL values sorted high?</td>
<td>True</td>
</tr>
<tr>
<td><code>nullsAreSortedLow()</code></td>
<td>Are NULL values sorted low?</td>
<td>False</td>
</tr>
<tr>
<td><code>storesLowerCaseIdentifiers()</code></td>
<td>Does the database treat mixed-case, unquoted SQL identifiers as case insensitive and store them in lowercase?</td>
<td>False</td>
</tr>
<tr>
<td><code>storesLowerCaseQuotedIdentifiers()</code></td>
<td>Does the database treat mixed-case, quoted SQL identifiers as case insensitive and store them in lowercase?</td>
<td>False</td>
</tr>
</tbody>
</table>
### Table 5: Return values for DatabaseMetaData methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>storesMixedCaseIdentifiers()</td>
<td>Does the database treat mixed-case, unquoted SQL identifiers as case insensitive and store them in mixed case?</td>
<td>False</td>
</tr>
<tr>
<td>storesMixedCaseQuotedIdentifiers()</td>
<td>Does the database treat mixed-case, quoted SQL identifiers as case insensitive and store them in mixed case?</td>
<td>True</td>
</tr>
<tr>
<td>storesUpperCaseIdentifiers()</td>
<td>Does the database treat mixed-case, unquoted SQL identifiers as case insensitive and store them in uppercase?</td>
<td>True</td>
</tr>
<tr>
<td>storesUpperCaseQuotedIdentifiers()</td>
<td>Does the database treat mixed-case, quoted SQL identifiers as case insensitive and store them in uppercase?</td>
<td>False</td>
</tr>
<tr>
<td>supportsAlterTableWithAddColumn()</td>
<td>Is ALTER TABLE with add column supported?</td>
<td>False</td>
</tr>
<tr>
<td>supportsAlterTableWithDropColumn()</td>
<td>Is ALTER TABLE with drop column supported?</td>
<td>False</td>
</tr>
<tr>
<td>supportsANSI92EntryLevelSQL()</td>
<td>Is the ANSI92 entry level SQL grammar supported? All JDBC-compliant drivers must return true.</td>
<td>True</td>
</tr>
<tr>
<td>supportsANSI92FullSQL()</td>
<td>Is the ANSI92 full SQL grammar supported?</td>
<td>False</td>
</tr>
<tr>
<td>supportsANSI92IntermediateSQL()</td>
<td>Is the ANSI92 intermediate SQL grammar supported?</td>
<td>False</td>
</tr>
<tr>
<td>supportsCatalogsInDataManipulation()</td>
<td>Can a catalog name be used in a data manipulation statement?</td>
<td>False</td>
</tr>
<tr>
<td>supportsCatalogsInIndexDefinitions()</td>
<td>Can a catalog name be used in an index definition statement?</td>
<td>False</td>
</tr>
<tr>
<td>supportsCatalogsInPrivilegeDefinitions()</td>
<td>Can a catalog name be used in a privilege definition statement?</td>
<td>False</td>
</tr>
</tbody>
</table>
## Return values for DatabaseMetaData methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>supportsCatalogsInProcedureCalls()</code></td>
<td>Can a catalog name be used in a procedure call statement?</td>
<td>False</td>
</tr>
<tr>
<td><code>supportsCatalogsInTableDefinitions()</code></td>
<td>Can a catalog name be used in a table definition statement?</td>
<td>False</td>
</tr>
<tr>
<td><code>supportsColumnAliasing()</code></td>
<td>Is column aliasing supported? If so, the SQL AS clause can be used to provide names for computed columns or to provide alias names for columns as required.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsConvert()</code></td>
<td>Is the CONVERT function between SQL types supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsConvert(int, int)</code></td>
<td>Is CONVERT between the given SQL types supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsCoreSQLGrammar()</code></td>
<td>Is the ODBC Core SQL grammar supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsCorrelatedSubqueries()</code></td>
<td>Are correlated subqueries supported? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsDataDefinitionAndDataManipulationTransactions()</code></td>
<td>Are both data definition and data manipulation statements within a transaction supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsDataManipulationTransactionsOnly()</code></td>
<td>Are only data manipulation statements within a transaction supported?</td>
<td>False</td>
</tr>
<tr>
<td><code>supportsDifferentTableCorrelationNames()</code></td>
<td>If table correlation names are supported, are they restricted to be different from the names of the tables?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsExpressionsInOrderBy()</code></td>
<td>Are expressions in ORDER BY lists supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsExtendedSQLGrammar()</code></td>
<td>Is the ODBC Extended SQL grammar supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsFullOuterJoins()</code></td>
<td>Are full nested outer joins supported?</td>
<td>False</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>supportsGroupBy()</td>
<td>Is some form of GROUP BY clause supported?</td>
<td>True</td>
</tr>
<tr>
<td>supportsGroupByBeyondSelect()</td>
<td>Can a GROUP BY clause add columns not in the SELECT provided it specifies all the columns in the SELECT?</td>
<td>True</td>
</tr>
<tr>
<td>supportsGroupByUnrelated()</td>
<td>Can a GROUP BY clause use columns not in the SELECT?</td>
<td>False</td>
</tr>
<tr>
<td>supportsIntegrityEnhancementFacility()</td>
<td>Is the SQL Integrity Enhancement Facility supported?</td>
<td>True</td>
</tr>
<tr>
<td>supportsLikeEscapeClause()</td>
<td>Is the escape character in LIKE clauses supported? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td>supportsLimitedOuterJoins()</td>
<td>Is there limited support for outer joins? (This will be true if supportFullOuterJoins is true.)</td>
<td>False</td>
</tr>
<tr>
<td>supportsMinimumSQLGrammar()</td>
<td>Is the ODBC Minimum SQL grammar supported? All JDBC-compliant drivers must return true.</td>
<td>True</td>
</tr>
<tr>
<td>supportsMixedCaseIdentifiers()</td>
<td>Does the database treat mixed-case, unquoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-compliant driver will always return false.</td>
<td>False</td>
</tr>
<tr>
<td>supportsMixedCaseQuotedIdentifiers()</td>
<td>Does the database treat mixed-case, quoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-compliant driver will always return true.</td>
<td>True</td>
</tr>
<tr>
<td>supportsMultipleResultSets()</td>
<td>Are multiple ResultSets from a single execute supported?</td>
<td>False</td>
</tr>
<tr>
<td>supportsMultipleTransactions()</td>
<td>Can multiple transactions be open at once (on different connections)?</td>
<td>True</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Returns</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>supportsNullableColumns()</td>
<td>Can columns be defined as non-nullable? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td>supportsOpenCursorsAcrossCommit()</td>
<td>Can cursors remain open across commits?</td>
<td>True</td>
</tr>
<tr>
<td>supportsOpenCursorsAcrossRollback()</td>
<td>Can cursors remain open across rollbacks?</td>
<td>True</td>
</tr>
<tr>
<td>supportsOpenStatementsAcrossCommit()</td>
<td>Can statements remain open across commits?</td>
<td>True</td>
</tr>
<tr>
<td>supportsOpenStatementsAcrossRollback()</td>
<td>Can statements remain open across rollbacks?</td>
<td>True</td>
</tr>
<tr>
<td>supportsOrderByUnrelated()</td>
<td>Can an ORDER BY clause use columns not in the SELECT?</td>
<td>False</td>
</tr>
<tr>
<td>supportsOuterJoins()</td>
<td>Is some form of outer join supported?</td>
<td>True</td>
</tr>
<tr>
<td>supportsPositionedDelete()</td>
<td>Is positioned DELETE supported?</td>
<td>True</td>
</tr>
<tr>
<td>supportsPositionedUpdate()</td>
<td>Is positioned UPDATE supported?</td>
<td>True</td>
</tr>
<tr>
<td>supportsSchemasInDataManipulation()</td>
<td>Can a schema name be used in a data manipulation statement?</td>
<td>True</td>
</tr>
<tr>
<td>supportsSchemasInIndexDefinitions()</td>
<td>Can a schema name be used in an index definition statement?</td>
<td>True</td>
</tr>
<tr>
<td>supportsSchemasInPrivilegeDefinitions()</td>
<td>Can a schema name be used in a privilege definition statement?</td>
<td>True</td>
</tr>
<tr>
<td>supportsSchemasInProcedureCalls()</td>
<td>Can a schema name be used in a procedure call statement?</td>
<td>True</td>
</tr>
<tr>
<td>supportsSchemasInTableDefinitions()</td>
<td>Can a schema name be used in a table definition statement?</td>
<td>True</td>
</tr>
<tr>
<td>supportsSelectForUpdate()</td>
<td>Is SELECT for UPDATE supported?</td>
<td>True</td>
</tr>
<tr>
<td>supportsStoredProcedures()</td>
<td>Are stored procedure calls using the stored procedure escape syntax supported?</td>
<td>True</td>
</tr>
</tbody>
</table>
## Return values for DatabaseMetaData methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>supportsSubqueriesInComparisons()</code></td>
<td>Are subqueries in comparison expressions supported? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsSubqueriesInExists()</code></td>
<td>Are subqueries in EXISTS expressions supported? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsSubqueriesInIns()</code></td>
<td>Are subqueries in IN statements supported? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsSubqueriesInQuantifieds()</code></td>
<td>Are subqueries in quantified expressions supported? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsTableCorrelationNames()</code></td>
<td>Are table correlation names supported? A JDBC-compliant driver always returns true.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsTransactionIsolationLevel(int)</code></td>
<td>Does the database support the given transaction isolation level?</td>
<td>True (for all four transaction levels)</td>
</tr>
<tr>
<td><code>supportsTransactions ()</code></td>
<td>Are transactions supported? If not, commit is a no-op and the isolation level is TRANSACTION_NONE.</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsUnion()</code></td>
<td>Is SQL UNION supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>supportsUnionAll()</code></td>
<td>Is SQL UNION ALL supported?</td>
<td>True</td>
</tr>
<tr>
<td><code>usesLocalFilePerTable()</code></td>
<td>Does the database use a file for each table?</td>
<td>False</td>
</tr>
<tr>
<td><code>usesLocalFiles()</code></td>
<td>Does the database store tables in a local file?</td>
<td>False</td>
</tr>
</tbody>
</table>

**Example**

The following example is a code segment that illustrates calling methods of `DatabaseMetadata`:

```java
Example code...
```
Connection con = DriverManager.getConnection ( url, prop);

// Get the DatabaseMetaData object and display
// some information about the connection
DatabaseMetaData dma = con.getMetaData ();

o.println("\nConnected to " + dma.getURL());
o.println("Driver  " +
dma.getDriverName());
o.println("Version  " +
dma.getDriverVersion());
Part 3

ODBC Reference

OpenEdge SQL and ODBC Data Types
SQLGetInfo
ODBC Scalar Functions
OpenEdge SQL and ODBC Data Types

This section contains Table 1, which shows how the OpenEdge SQL data types are mapped to the standard ODBC data types:

<table>
<thead>
<tr>
<th>Progress data type</th>
<th>ODBC data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>SQL_BINARY</td>
</tr>
<tr>
<td>BIT</td>
<td>SQL_BIT</td>
</tr>
<tr>
<td>CHAR</td>
<td>SQL_CHAR</td>
</tr>
<tr>
<td>DATE</td>
<td>SQL_TYPE_DATE</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>SQL_DECIMAL</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>SQL_FLOAT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>SQL_INTEGER</td>
</tr>
<tr>
<td>REAL</td>
<td>SQL_FLOAT</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SQL_SMALLINT</td>
</tr>
<tr>
<td>TIME</td>
<td>SQL_TYPE_TIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>SQL_TYPE_TIMESTAMP</td>
</tr>
<tr>
<td>TINYINT</td>
<td>SQL_TINYINT</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>SQL_VARBINARY</td>
</tr>
<tr>
<td>LVARBINARY</td>
<td>SQL_LONGVARBINARY</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>SQL_VARCHAR</td>
</tr>
</tbody>
</table>
SQLGetInfo

This section details the information the ODBC Driver returns to SQLGetInfo.

Table 1 describes return values the ODBC driver returns to SQLGetInfo.

<table>
<thead>
<tr>
<th>Description</th>
<th>$fInfoType$ argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guaranteed execute privileges on all procedures returned by SQLProcedures</td>
<td>SQL_ACCESSIBLE_PROCEDURES</td>
<td>N</td>
</tr>
<tr>
<td>Guaranteed read access to all table names returned by SQLTables</td>
<td>SQL_ACCESSIBLE_TABLES</td>
<td>N</td>
</tr>
<tr>
<td>Maximum number of active connections</td>
<td>SQL_ACTIVE_CONNECTIONS</td>
<td>0</td>
</tr>
<tr>
<td>Maximum number of active statements supported for an active connection</td>
<td>SQL_ACTIVE_STATEMENTS</td>
<td>100</td>
</tr>
<tr>
<td>Maximum number of active environments</td>
<td>SQL_ACTIVE_ENVIRONMENTS</td>
<td>0</td>
</tr>
<tr>
<td>Description</td>
<td>InfoType argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Support for ALTER DOMAIN statement</td>
<td>SQL_ALTER_DOMAIN</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for ALTER TABLE clauses</td>
<td>SQL_ALTER_TABLE</td>
<td>0x00000000</td>
</tr>
<tr>
<td>SQL Conformance</td>
<td>SQL_SQL_CONFORMANCE</td>
<td>SQL_SC_SQL92_ENTRY</td>
</tr>
<tr>
<td>Support for datetime literals</td>
<td>SQL_DATETIME_LITERALS</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Level of asynchronous mode support</td>
<td>SQL_ASYNC_MODE</td>
<td>SQL_AM_NONE</td>
</tr>
<tr>
<td>Behavior with respect to the availability of row counts in batches</td>
<td>SQL_BATCH_ROW_COUNT</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for batches</td>
<td>SQL_BATCH_SUPPORT</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for bookmarks</td>
<td>SQL_BOOKMARK_PERSISTENCE</td>
<td>SQL_BP_UPDATE SQL_BP_SCROLL</td>
</tr>
<tr>
<td>Position of qualifier in a qualified table name</td>
<td>SQL_CATALOG_LOCATION</td>
<td>SQL_CL_START</td>
</tr>
<tr>
<td>Support for catalog names</td>
<td>SQL_CATALOG_NAME</td>
<td>Y</td>
</tr>
<tr>
<td>Character used to separate table, column qualifiers</td>
<td>SQL_CATALOG_NAME_SEPARATOR</td>
<td>“.”</td>
</tr>
<tr>
<td>Term for object that qualifies table names</td>
<td>SQL_CATALOG_TERM</td>
<td>“database”</td>
</tr>
<tr>
<td>Statements that support qualifiers</td>
<td>SQL_CATALOG_USAGE</td>
<td>SQL_CU_DML_STATEMENTS SQL_CU_PROCEDURE_INVOCATION</td>
</tr>
<tr>
<td>Default collation sequence name for the default character set</td>
<td>SQL_COLLATION_SEQ</td>
<td>“”</td>
</tr>
<tr>
<td>Support for column aliases</td>
<td>SQL_COLUMN_ALIAS</td>
<td>Y</td>
</tr>
</tbody>
</table>
## Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>fInfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result of concatenation of NULL character column with non-NULL column</td>
<td>SQL_CONCAT_NULL_BEHAVIOR</td>
<td>SQL_CB_NULL = 0</td>
</tr>
<tr>
<td>Conversion from BIGINT</td>
<td>SQL_CONVERT_BIGINT</td>
<td>SQL_CVT_CHAR, SQL_CVT_BIGINT, SQL_CVT_TINYINT, SQL_CVT_SMALLINT, SQL_CVT_INTEGER, SQL_CVT_FLOAT, SQL_CVT_DOUBLE</td>
</tr>
<tr>
<td>Conversion from BINARY</td>
<td>SQL_CONVERT_BINARY</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Conversion from BIT</td>
<td>SQL_CONVERT_BIT</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Conversion from CHAR</td>
<td>SQL_CONVERT_CHAR</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER, SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE, SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT, SQL_CVT_DATE, SQL_CVT_TIME, SQL_CVT_TIMESTAMP</td>
</tr>
<tr>
<td>Conversion from DATE</td>
<td>SQL_CONVERT_DATE</td>
<td>SQL_CVT_CHAR, SQL_CVT_VARCHAR, SQL_CVT_DATE, SQL_CVT_TIMESTAMP</td>
</tr>
<tr>
<td>Conversion from DECIMAL</td>
<td>SQL_CONVERT_DECIMAL</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER, SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE, SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
</tbody>
</table>
Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>fInfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion from DOUBLE</td>
<td>SQL_CONVERT_DOUBLE</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER, SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE, SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
<tr>
<td>Conversion from FLOAT</td>
<td>SQL_CONVERT_FLOAT</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER, SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE, SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
<tr>
<td>Support for conversion functions</td>
<td>SQL_CONVERT_FUNCTIONS</td>
<td>SQL_FN_CVT_CONVERT</td>
</tr>
<tr>
<td>Conversion from INTEGER</td>
<td>SQL_CONVERT_INTEGER</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER, SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE, SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
<tr>
<td>Conversion from INTERVAL_DAY_</td>
<td>SQL_CONVERT_INTERVAL_DAY_TIME</td>
<td>0x00000000</td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversion from INTERVAL_YEAR_</td>
<td>SQL_CONVERT_INTERVAL_YEAR_MONTH</td>
<td>0x00000000</td>
</tr>
<tr>
<td>MONTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversion from INTERVAL_DAY_</td>
<td>SQL_CONVERT_INTERVAL_DAY_TIME</td>
<td>0x00000000</td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conversion from LONGVARBINARY</td>
<td>SQL_CONVERT_LONGVARBINARY</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Conversion from LONGVARCHAR</td>
<td>SQL_CONVERT_LONGVARCHAR</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

OpenEdge® Data Management: SQL Reference
<table>
<thead>
<tr>
<th>Description</th>
<th>infoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion from NUMERIC</td>
<td>SQL_CONVERT_NUMERIC</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
<tr>
<td>Conversion from REAL</td>
<td>SQL_CONVERT_REAL</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
<tr>
<td>Conversion from SMALLINT</td>
<td>SQL_CONVERT_SMALLINT</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
<tr>
<td>Conversion from TIME</td>
<td>SQL_CONVERT_TIME</td>
<td>SQL_CVT_CHAR, SQL_CVT_TIME, SQL_CVT_TIMESTAMP</td>
</tr>
<tr>
<td>Conversion from TIMESTAMP</td>
<td>SQL_CONVERT_TIMESTAMP</td>
<td>SQL_CVT_CHAR, SQL_CVT_VARCHAR, SQL_CVT_DATE, SQL_CVT_TIME, SQL_CVT_TIMESTAMP</td>
</tr>
<tr>
<td>Conversion from TINYINT</td>
<td>SQL_CONVERT_TINYINT</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT</td>
</tr>
<tr>
<td>Conversion from VARBINARY</td>
<td>SQL_CONVERT_VARBINARY</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>InfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion from VARCHAR</td>
<td>SQL_CONVERT_VARCHAR</td>
<td>SQL_CVT_CHAR, SQL_CVT_NUMERIC, SQL_CVT_DECIMAL, SQL_CVT_INTEGER, SQL_CVT_SMALLINT, SQL_CVT_FLOAT, SQL_CVT_REAL, SQL_CVT_DOUBLE, SQL_CVT_VARCHAR, SQL_CVT_TINYINT, SQL_CVT_BIGINT, SQL_CVT_DATE, SQL_CVT_TIME, SQL_CVT_TIMESTAMP</td>
</tr>
<tr>
<td>Conversion from WCHAR</td>
<td>SQL_CONVERT_WCHAR</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Conversion from WLONGVARCHAR</td>
<td>SQL_CONVERT_WLONGVARCHAR</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Conversion from WVARCHAR</td>
<td>SQL_CONVERT_WVARCHAR</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Support for table correlation names</td>
<td>SQL_CORRELATION_NAME</td>
<td>SQL_CN_DIFFERENT</td>
</tr>
<tr>
<td>Support for CREATE ASSERTION statement</td>
<td>SQL_CREATE_ASSERTION</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Support for CREATE CHARACTER SET statement</td>
<td>SQL_CREATE_CHARACTER_SET</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Support for CREATE COLLATION statement</td>
<td>SQL_CREATE_COLLATION</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Support for CREATE DOMAIN statement</td>
<td>SQL_CREATE_DOMAIN</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Support for CREATE SCHEMA statement</td>
<td>SQL_CREATE_SCHEMA</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Support for CREATE TABLE statement</td>
<td>SQL_CREATE_TABLE</td>
<td>SQL_CT_CREATE_TABLE, SQL_CT_COLUMN_CONSTRAINT, SQL_CT_TABLE_CONSTRAINT</td>
</tr>
<tr>
<td>Support for CREATE TRANSLATION statement</td>
<td>SQL_CREATE_TRANSLATION</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Support for CREATE VIEW statement</td>
<td>SQL_CREATE_VIEW</td>
<td>SQL_CV_CREATE_VIEW, SQL_CV_CHECK_OPTION</td>
</tr>
</tbody>
</table>
Table 1: Information the ODBC driver returns to SQLGetInfo (7 of 19)

<table>
<thead>
<tr>
<th>Description</th>
<th>InfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effect of COMMIT operation on cursors and prepared statements</td>
<td>SQL_CURSOR_COMMIT_BEHAVIOR</td>
<td>SQL_CB_PRESERVE</td>
</tr>
<tr>
<td>Effect of ROLLBACK operation on cursors and prepared statements</td>
<td>SQL_CURSOR_ROLLBACK_BEHAVIOR</td>
<td>SQL_CB_PRESERVE</td>
</tr>
<tr>
<td>Support for cursor sensitivity</td>
<td>SQL_CURSOR_SENSITIVITY</td>
<td>SQL_INSENSITIVE</td>
</tr>
<tr>
<td>Name of the data source as specified to the ODBC Administrator</td>
<td>SQL_DATA_SOURCE_NAME</td>
<td>(String containing the name)</td>
</tr>
<tr>
<td>Access limited to read-only</td>
<td>SQL_DATA_SOURCE_READ_ONLY</td>
<td>N (Read-write access)</td>
</tr>
<tr>
<td>Name of the Progress SQL-92 ODBC data source on the server system</td>
<td>SQL_DATABASE_NAME</td>
<td>(String containing the name)</td>
</tr>
<tr>
<td>Name of the database product supporting the data source</td>
<td>SQL_DBMS_NAME</td>
<td>OPENEDGE</td>
</tr>
<tr>
<td>Version of the database product</td>
<td>SQL_DBMS_VER</td>
<td>10.1B</td>
</tr>
<tr>
<td>Default transaction isolation level</td>
<td>SQL_DEFAULT_TXN_ISOLATION</td>
<td>SQL_TXN_READ_COMMITTED</td>
</tr>
<tr>
<td>Support for describing parameters via DESCRIBE INPUT statement</td>
<td>SQL_DESCRIBE_PARAMETER</td>
<td>Y (Supports)</td>
</tr>
<tr>
<td>Version of the driver manager</td>
<td>SQL_DM_VER</td>
<td>03.52.1117.0000</td>
</tr>
<tr>
<td>Connection handle determined by the argument InfoType</td>
<td>SQL_DRIVER_HDBC</td>
<td>0x017E4538</td>
</tr>
</tbody>
</table>
## Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>InfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver's descriptor handle determined by the Driver Manager's descriptor handle</td>
<td>SQL_DRIVER_HDESC</td>
<td>0x017E68A8</td>
</tr>
<tr>
<td>Environment handle determined by the argument InfoType</td>
<td>SQL_DRIVER_HENV</td>
<td>0x017E4090</td>
</tr>
<tr>
<td>Handle library from the load library returned to the Driver Manager when it loaded the driver DLL</td>
<td>SQL_DRIVER_HLIB</td>
<td>0x28660000</td>
</tr>
<tr>
<td>Driver's statement handle determined by the Driver Manager's statement handle</td>
<td>SQL_DRIVER_HSTMT</td>
<td>0x01828050</td>
</tr>
<tr>
<td>Name of the dynamic link library file for the ODBC Driver</td>
<td>SQL_DRIVER_NAME</td>
<td>Windows pgoe1022.DLL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AIX, SOLARIS, LINUX pgoe1022.SO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HPIIX pgoe1022.SL</td>
</tr>
<tr>
<td>Supported ODBC version</td>
<td>SQL_DRIVER_ODBC_VER</td>
<td>03.52</td>
</tr>
<tr>
<td>Current version of the ODBC Driver</td>
<td>SQL_DRIVER_VER</td>
<td>05.20.0039 (b0034, u0022)</td>
</tr>
<tr>
<td>Support for DROP ASSERTION statement</td>
<td>SQL_DROP_ASSERTION</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for DROP CHARACTER SET statement</td>
<td>SQL_DROP_CHARACTER_SET</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for DROP COLLATION statement</td>
<td>SQL_DROP_COLLATION</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for DROP DOMAIN statement</td>
<td>SQL_DROP_DOMAIN</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for DROP SCHEMA statement</td>
<td>SQL_DROP_SCHEMA</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Description</td>
<td>$fInfoType$ argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Support for <strong>DROP TABLE</strong> statement</td>
<td>SQL_DROP_TABLE</td>
<td>SQL_DT_DROP_TABLE</td>
</tr>
<tr>
<td>Support for <strong>DROP TRANSLATION</strong> statement</td>
<td>SQL_DROP_TRANSLATION</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for <strong>DROP VIEW</strong> statement</td>
<td>SQL_DROP_VIEW</td>
<td>SQL_DV_DROP_VIEW</td>
</tr>
<tr>
<td>Supported attributes of a dynamic cursor: subset 1</td>
<td>SQL_DYNAMIC_CURSOR_ATTRIBUTES1</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Supported attributes of a dynamic cursor: subset 2</td>
<td>SQL_DYNAMIC_CURSOR_ATTRIBUTES2</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for expressions in <strong>ORDER BY</strong> clause</td>
<td>SQL.Expressions_IN_ORDERBY</td>
<td>Y</td>
</tr>
<tr>
<td>Supported fetch direction option.</td>
<td>SQL_FETCH_DIRECTION</td>
<td>SQL_FD_FETCH_NEXT, SQL_FD_FETCH_FIRST, SQL_FD_FETCH_LAST, SQL_FD_FETCH_PRIOR, SQL_FD_FETCH_ABSOLUTE, SQL_FD_FETCH_RELATIVE, SQL_FD_FETCH_ABSOLUTE, SQL_FD_FETCH_BOOKMARK</td>
</tr>
<tr>
<td>Single-tier driver behavior</td>
<td>SQL_FILE_USAGE</td>
<td>SQL_FILE_NOT_SUPPORTED</td>
</tr>
<tr>
<td>Supported attributes of a forward-only cursor: subset 1</td>
<td>SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1</td>
<td>SQL_CA1_NEXT, SQL_CA1_BULK_ADD</td>
</tr>
<tr>
<td>Supported attributes of a forward-only cursor: subset 2</td>
<td>SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2</td>
<td>SQL_CA2_MAX_ROWS_SELECT, SQL_CA2_MAX_ROWS_CATALOG</td>
</tr>
<tr>
<td>Supported extensions to <strong>SQLGetData</strong></td>
<td>SQL_GETDATA_EXTENSIONS</td>
<td>SQL_GD.AnyCOLUMN, SQL_GD.AnyORDER, SQL_GD.BLOCK, SQL_GD.BOUND</td>
</tr>
<tr>
<td>Relationship between <strong>GROUP BY</strong> clause and columns in the select list</td>
<td>SQL_GROUP_BY</td>
<td>SQL_GB_GROUP_BY_CONTAINS_SELECT</td>
</tr>
</tbody>
</table>
### Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th><em>fInfoType</em> argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case sensitivity of user-supplied names</td>
<td>SQL_IDENTIFIER_CASE</td>
<td>SQL_IC_UPPER</td>
</tr>
<tr>
<td>Character used to enclose delimited identifiers</td>
<td>SQL_IDENTIFIER_QUOTE_CHAR</td>
<td>* (Double quotation mark)</td>
</tr>
<tr>
<td>Keywords supported in the CREATE INDEX statement</td>
<td>SQL_INDEX_KEYWORDS</td>
<td>SQL_IK_ASC</td>
</tr>
<tr>
<td>Supported views in INFORMATION_SCHEMA</td>
<td>SQL_INFO_SCHEMA_VIEWS</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Support for Integrity Enhancement Facility</td>
<td>SQL_INTEGRITY</td>
<td>Y</td>
</tr>
<tr>
<td>Supported attributes of a keyset cursor: subset 1</td>
<td>SQL_KEYSET_CURSOR_ATTRIBUTE1</td>
<td>SQL_CA1_NEXT</td>
</tr>
<tr>
<td>Supported attributes of a keyset cursor: subset 2</td>
<td>SQL_KEYSET_CURSOR_ATTRIBUTE2</td>
<td>SQL_CA2_READ_ONLY_ CONCURRENCY</td>
</tr>
<tr>
<td>Data source specific keywords</td>
<td>SQL_KEYWORDS</td>
<td>See the OpenEdge SQL Reserved Words section for a list of SQL Keywords.</td>
</tr>
<tr>
<td>Description</td>
<td>fInfoType argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Support for escape clause in LIKE predicates</td>
<td>SQL_LIKE_ESCAPE_CLAUSE</td>
<td>Y</td>
</tr>
<tr>
<td>Support for lock types</td>
<td>SQL_LOCK_TYPES</td>
<td>SQL_LCK_NO_CHANGE</td>
</tr>
<tr>
<td>Maximum number of active concurrent statements in asynchronous mode</td>
<td>SQL_MAX_ASYNC_CONCURRENT__STATEMENTS</td>
<td>0</td>
</tr>
<tr>
<td>Maximum length in hexadecimal characters of binary literals</td>
<td>SQL_MAX_BINARY_LITERAL_LEN</td>
<td>31,983</td>
</tr>
<tr>
<td>Maximum length of a table or column qualifier</td>
<td>SQL_MAX_CATALOG_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum length in characters of character string literals</td>
<td>SQL_MAX_CHAR_LITERAL_LEN</td>
<td>31,983</td>
</tr>
<tr>
<td>Maximum length of a column name</td>
<td>SQL_MAX_COLUMN_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of columns allowed in GROUP BY clause</td>
<td>SQL_MAX_COLUMNS_IN_GROUP_BY</td>
<td>5000</td>
</tr>
<tr>
<td>Maximum number of columns allowed in an index</td>
<td>SQL_MAX_COLUMNS_IN_INDEX</td>
<td>16</td>
</tr>
<tr>
<td>Maximum number of columns allowed in ORDER BY clause</td>
<td>SQL_MAX_COLUMNS_IN_ORDER_BY</td>
<td>0</td>
</tr>
<tr>
<td>Maximum number of columns allowed in a SELECT list</td>
<td>SQL_MAX_COLUMNS_IN_SELECT</td>
<td>5000</td>
</tr>
<tr>
<td>Maximum number of columns allowed in a table</td>
<td>SQL_MAX_COLUMNS_IN_TABLE</td>
<td>5000</td>
</tr>
<tr>
<td>Description</td>
<td><code>fInfoType</code> argument</td>
<td>Returns</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>--------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Maximum number of active SQL statements</td>
<td><code>SQL_MAX_CONCURRENT_ACTIVITIES</code></td>
<td>100</td>
</tr>
<tr>
<td>Maximum length of a cursor name</td>
<td><code>SQL_MAX_CURSOR_NAME_LEN</code></td>
<td>18</td>
</tr>
<tr>
<td>Maximum number of active connections</td>
<td><code>SQL_MAX_DRIVER_CONNECTIONS</code></td>
<td>0</td>
</tr>
<tr>
<td>Maximum length of user-defined names</td>
<td><code>SQL_MAX_IDENTIFIER_LEN</code></td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of bytes allowed in the combined fields of an index</td>
<td><code>SQL_MAX_INDEX_SIZE</code></td>
<td>115</td>
</tr>
<tr>
<td>Maximum length of a procedure name</td>
<td><code>SQL_MAX_PROCEDURE_NAME_LEN</code></td>
<td>32</td>
</tr>
<tr>
<td>Maximum length in bytes of a table row</td>
<td><code>SQL_MAX_ROW_SIZE</code></td>
<td>31995</td>
</tr>
<tr>
<td>Whether maximum row size includes <code>LONGVARCHAR</code> and <code>LONGVARBINARY</code></td>
<td><code>SQL_MAX_ROW_SIZE_INCLUDES_LONG</code></td>
<td>N</td>
</tr>
<tr>
<td>Maximum length of an owner name</td>
<td><code>SQL_MAX_SCHEMA_NAME_LEN</code></td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of characters in an SQL statement</td>
<td><code>SQL_MAX_STATEMENT_LEN</code></td>
<td>131,000</td>
</tr>
<tr>
<td>Maximum length of a table name</td>
<td><code>SQL_MAX_TABLE_NAME_LEN</code></td>
<td>32</td>
</tr>
<tr>
<td>Maximum number of tables allowed in <code>FROM</code> clause</td>
<td><code>SQL_MAX_TABLES_IN_SELECT</code></td>
<td>250</td>
</tr>
<tr>
<td>Maximum length of a user name</td>
<td><code>SQL_MAX_USER_NAME_LEN</code></td>
<td>32</td>
</tr>
<tr>
<td>Maximum length of owner name</td>
<td><code>SQL_MAX_OWNER_NAME_LEN</code></td>
<td>32</td>
</tr>
<tr>
<td>Description</td>
<td>fInfoType argument</td>
<td>Returns</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Maximum length of a qualifier name</td>
<td>SQL_MAX_QUALIFIER_NAME_LEN</td>
<td>32</td>
</tr>
<tr>
<td>Support for multiple result sets</td>
<td>SQL_MULT_RESULT_SETS</td>
<td>N</td>
</tr>
<tr>
<td>Support for active transactions on multiple connections</td>
<td>SQL_MULTIPLE_ACTIVE_TXN</td>
<td>Y</td>
</tr>
<tr>
<td>Whether data source requires length of LONGVARCHAR and LONGVARBINARY data</td>
<td>SQL_NEED_LONG_DATA_LEN</td>
<td>Y</td>
</tr>
<tr>
<td>Support for \texttt{NOT NULL} clause in CREATE TABLE statement</td>
<td>SQL_NON_NULLABLE_COLUMNS</td>
<td>SQL_NNC_NON_NULL</td>
</tr>
<tr>
<td>Where \texttt{NULL} values are sorted in a list</td>
<td>SQL_NULL_COLLATION</td>
<td>SQL_NC_HIGH</td>
</tr>
<tr>
<td>Numeric functions supported</td>
<td>SQL_NUMERIC_FUNCTIONS</td>
<td>SQL_FN_NUM_ABS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_ACOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_ASIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_ATAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_ATAN2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_CEILING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_COS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_EXP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_FLOOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_MOD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_PI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_POWER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_RADIANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_RAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_ROUND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_SIGN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_SIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_SQRT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_TAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_DEGREES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_NUM_LOG10</td>
</tr>
<tr>
<td>Level of ODBC conformance</td>
<td>SQL_ODBC_API_CONFORMANCE</td>
<td>SQL_OAC_LEVEL1</td>
</tr>
<tr>
<td>Level of ODBC 3.x interface conformance</td>
<td>SQL_ODBC_INTERFACE_CONFORMANCE</td>
<td>SQL_OIC_CORE</td>
</tr>
<tr>
<td>Description</td>
<td>fInfoType argument</td>
<td>Returns</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>SQL Access Group (SAG) conformance</td>
<td>SQL_ODBC_SAG_CLI_CONFORMANCE</td>
<td>SQL_OSCC_COMPLIANT</td>
</tr>
<tr>
<td>Level of SQL conformance</td>
<td>SQL_ODBC_SQL_CONFORMANCE</td>
<td>SQL_OSC_EXTENDED</td>
</tr>
<tr>
<td>Referential integrity syntax support</td>
<td>SQL_ODBC_SQL_OPT_IEF</td>
<td>Y</td>
</tr>
<tr>
<td>ODBC version supported by driver manager</td>
<td>SQL_ODBC_VER</td>
<td>03.52.0000</td>
</tr>
<tr>
<td>Types of outer joins supported</td>
<td>SQL_OJ_CAPABILITIES</td>
<td>SQL_OJ_LEFT \n SQL_OJ_RIGHT \n SQL_OJ_NOT_ORDERED \n SQL_OJ_INNER \n SQL_OJ_ALL_COMPARISON_OPS</td>
</tr>
<tr>
<td>Whether columns in ORDER BY clause must also be in select list</td>
<td>SQL_ORDER_BY_COLUMNS_IN_SELECT</td>
<td>N</td>
</tr>
<tr>
<td>Support for outer joins</td>
<td>SQL_OUTER_JOINS</td>
<td>Y</td>
</tr>
<tr>
<td>Name for an owner</td>
<td>SQL_OWNER_TERM</td>
<td>owner</td>
</tr>
<tr>
<td>Statements in which owner can be used</td>
<td>SQL_OWNER_USAGE</td>
<td>SQL_OU_DML_STATEMENTS \n SQL_OU_PROCEDURE_INVOCATION \n SQL_OU_TABLE_DEFINITION \n SQL_OU_INDEX_DEFINITION \n SQL_OU_PRIVILEGE_DEFINITION</td>
</tr>
<tr>
<td>Characteristics of row counts available in a parameterized execution</td>
<td>SQL_PARAM_ARRAY_ROW_COUNTS</td>
<td>SQL_PARC_NO_BATCH</td>
</tr>
<tr>
<td>Characteristics of result sets available in a parameterized execution</td>
<td>SQL_PARAM_ARRAY_SELECTS</td>
<td>SQL_PAS_NO_SELECT</td>
</tr>
</tbody>
</table>
Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>fInfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported operations in SQLSetPos</td>
<td>SQL_POS_OPERATIONS</td>
<td>SQL_POS_POSITION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_POS_REFRESH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_POS_UPDATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_POS_DELETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_POS_ADD</td>
</tr>
<tr>
<td>Supported positioned SQL statements</td>
<td>SQL_POSITIONED_STATEMENTS</td>
<td>SQL_PS_POSITIONED_DELETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_PS_POSITIONED_UPDATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_PS_SELECT_FOR_UPDATE</td>
</tr>
<tr>
<td>Term for procedures</td>
<td>SQL_PROCEDURE_TERM</td>
<td>procedure</td>
</tr>
<tr>
<td>SQL procedures support</td>
<td>SQL_PROCEDURES</td>
<td>Y</td>
</tr>
<tr>
<td>Support for qualifiers</td>
<td>SQL_QUALIFIER_USAGE</td>
<td>SQL_CU_DML_STATEMENTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CU_PROCEDURE_INVOCATION</td>
</tr>
<tr>
<td>Case sensitivity of quoted user-supplied names</td>
<td>SQL_QUOTED_IDENTIFIER_CASE</td>
<td>SQL_IC_MIXED</td>
</tr>
<tr>
<td>Separator character used between qualifier name and element</td>
<td>SQL_QUALIFIER_NAME_SEPARATOR</td>
<td>&quot; . &quot;</td>
</tr>
<tr>
<td>Term used for a qualifier</td>
<td>SQL_QUALIFIER_TERM</td>
<td>&quot;database&quot;</td>
</tr>
<tr>
<td>Position of the qualifier in a qualified table name</td>
<td>SQL_QUALIFIER_LOCATION</td>
<td>SQL_CL_START</td>
</tr>
<tr>
<td>Detect changes to any row in mixed-cursor operations</td>
<td>SQL_ROW_UPDATES</td>
<td>Y</td>
</tr>
<tr>
<td>Term for entity that has owner privileges on objects</td>
<td>SQL_SCHEMA_TERM</td>
<td>owner</td>
</tr>
</tbody>
</table>
### Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th><code>fInfoType</code> argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statements that support use of owner qualifiers</td>
<td>SQL_SCHEMA_USAGE</td>
<td>SQL_OU_DML_STATEMENTS, SQL_OU_PROCEDURE_INVOCATION, SQL_OU_TABLE_DEFINITION, SQL_OU_INDEX_DEFINITION, SQL_OU_PRIVILEGE_DEFINITION</td>
</tr>
<tr>
<td>Options supported for scrollable cursors</td>
<td>SQL_SCROLL_OPTIONS</td>
<td>SQL_SO_FORWARD_ONLY, SQL_SO_STATIC, SQL_SO_KEYSET_DRIVEN</td>
</tr>
<tr>
<td>Support for scrollable cursors</td>
<td>SQL_SCROLL_CONCURRENCY</td>
<td>SQL_SCCO_READ_ONLY, SQL_SCCO_OPT_VALUES</td>
</tr>
<tr>
<td>Character to permit wildcard characters in search strings</td>
<td>SQL_SEARCH_PATTERN_ESCAPE</td>
<td>\ (Backslash)</td>
</tr>
<tr>
<td>Name of the system where the ODBC data source resides</td>
<td>SQL_SERVER_NAME</td>
<td>(String containing the name)</td>
</tr>
<tr>
<td>Special characters allowed in user-supplied names</td>
<td>SQL_SPECIAL_CHARACTERS</td>
<td>“_”, “%”</td>
</tr>
<tr>
<td>Datetime scalar functions supported</td>
<td>SQL_SQL92_DATETIME_FUNCTIONS</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Behavior of DELETE statement that refers to a foreign key.</td>
<td>SQL_SQL92_FOREIGN_KEY_DELETE_RULE</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>Behavior of UPDATE statement that refers to a foreign key</td>
<td>SQL_SQL92_FOREIGN_KEY_UPDATE_RULE</td>
<td>0x0000000000</td>
</tr>
<tr>
<td>GRANT statement clauses supported</td>
<td>SQL_SQL92_GRANT</td>
<td>SQL_SD_DELETE_TABLE, SQL_SD_INSERT_TABLE, SQL_SD_INSERT_COLUMN, SQL_SD_REFERENCES_TABLE, SQL_SD_REFERENCES_COLUMN, SQL_SD_SELECT_TABLE, SQL_SD_UPDATE_TABLE, SQL_SD_UPDATE_COLUMN</td>
</tr>
</tbody>
</table>
Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>fInfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric scalar functions supported</td>
<td>SQL_SQL92_NUMERIC_VALUE_FUNCTIONS</td>
<td>SQL_SNVF_CHAR_LENGTH, SQL_SNVF_CHARACTER_LENGTH</td>
</tr>
<tr>
<td>Predicates supported</td>
<td>SQL_SQL92_PREDICATES</td>
<td>SP_EXISTS, SQL_SP_ISNOTNULL, SQL_SP_ISNULL, SQL_SP_UNIQUE, SQL_SPLIKE, SQL_SP_IN, SQL_SP_BETWEEN</td>
</tr>
<tr>
<td>Relational join operators supported</td>
<td>SQL_SQL92_RELATIONAL_JOIN_OPERATORS</td>
<td>0x00000000</td>
</tr>
<tr>
<td>REVOKE statement clauses supported</td>
<td>SQL_SQL92_REVOKE</td>
<td>SQL_SR_GRANT_OPTION_FOR, SQL_SR.Cascade, SQL_SR_RESTRICT, SQL_SR_DELETE_TABLE, SQL_SR_INSERT_TABLE, SQL_SR_INSERT_COLUMN, SQL_SR_REFERENCES_TABLE, SQL_SR_REFERENCES_COLUMN, SQL_SR_SELECT_TABLE, SQL_SR_UPDATE_TABLE, SQL_SR_UPDATE_COLUMN</td>
</tr>
<tr>
<td>Row value constructor expressions supported</td>
<td>SQL_SQL92_ROW_VALUE_CONSTRUCTOR</td>
<td>0x00000000</td>
</tr>
<tr>
<td>String scalar functions supported</td>
<td>SQL_SQL92_STRING_FUNCTIONS</td>
<td>SQL_SSF_CONVERT, SQL_SSF_LOWER, SQL_SSF_UPPER, SQL_SSF_SUBSTRING, SQL_SSF_TRANSLATE, SQL_SSF_TRIM_LEADING, SQL_SSF_TRIM_TRAILING</td>
</tr>
<tr>
<td>Value expressions supported</td>
<td>SQL_SQL92_VALUE_EXPRESSIONS</td>
<td>SQL_SVE_COALESCE, SQL_SVE_NULLIF</td>
</tr>
<tr>
<td>CLI standards to which the driver conforms</td>
<td>SQL_STANDARD_CLI_CONFORMANCE</td>
<td>SQL_SCC_XOPEN_CLI_VERSION1</td>
</tr>
<tr>
<td>Supported attributes of a static cursor: subset 1</td>
<td>SQL_STATIC_CURSOR_ATTRIBUTES1</td>
<td>SQL_CAI_NEXT, SQL_CAI_ABSOLUTE, SQL_CAI_RELATIVE, SQL_CAI_BOOKMARK, SQL_CAI_LOCK_NO_CHANGE, SQL_CAI_POS_POSITION, SQL_CAI_POS_UPDATE, SQL_CAI_POS_DELETE, SQL_CAI_POS_REFRESH, SQL_CAI_POSITIONED_UPDATE, SQL_CAI_POSITIONED_DELETE, SQL_CAI_SELECT_FOR_UPDATE, SQL_CAI_BULK_ADD</td>
</tr>
<tr>
<td>Description</td>
<td>fInfoType argument</td>
<td>Returns</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Supported attributes of a static cursor: subset 2</td>
<td>SQL_STATIC_CURSOR _ATTRIBUTES2</td>
<td>SQL_CA2_READ_ONLY _CONCURRENCY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CA2_OPT_VALUES _CONCURRENCY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CA2_SENSITIVITY _UPDATES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CA2_MAX_ROWS_SELECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CA2_CRC_EXACT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_CA2_SIMULATE _TRY_UNIQUE</td>
</tr>
<tr>
<td>Support for detection of changes made to a static or key-set driven cursor through SQLSetPos</td>
<td>SQL_STATICSENSITIVITY</td>
<td>0x00000000</td>
</tr>
<tr>
<td>String functions supported</td>
<td>SQL_STRING_FUNCTIONS</td>
<td>SQL_FN_STR_CONCAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_INSERT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_LEFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_LTRIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_LENGTH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_LOCATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_LCASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_REPEAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_REPLACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_REPLACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_RIGHT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_RTRIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_SUBSTRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_UCASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_ASCII</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_DIFFERENCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_LOCATE_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_SPACE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_CHAR_LENGTH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_FN_STR_CHARACTER_LENGTH</td>
</tr>
<tr>
<td>Predicates that support subqueries</td>
<td>SQL_SUBQUERIES</td>
<td>SQL_SQ_COMPARISON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SQ_EXISTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SQ_IN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SQ_QUANTIFIED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SQL_SQ_CORRELATED_SUBQUERIES</td>
</tr>
<tr>
<td>System functions supported</td>
<td>SQL_SYSTEM_FUNCTIONS</td>
<td>SQL_FN_SYS_USERNAME</td>
</tr>
<tr>
<td>Term for tables</td>
<td>SQL_TABLE_TERM</td>
<td>&quot;table&quot;</td>
</tr>
</tbody>
</table>
### Table 1: Information the ODBC driver returns to SQLGetInfo

<table>
<thead>
<tr>
<th>Description</th>
<th>fInfoType argument</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp intervals supported for TIMESTAMPADD function</td>
<td>SQL_TIMEDATE_ADD_INTERVALS</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Timestamp intervals supported for TIMESTAMPDIFF function</td>
<td>SQL_TIMEDATE_DIFF_INTERVALS</td>
<td>0x00000000</td>
</tr>
<tr>
<td>Date-time functions supported</td>
<td>SQL_TIMEDATE_FUNCTIONS</td>
<td>SQL_FN_TD_NOW, SQL_FN_CURDATE,...</td>
</tr>
<tr>
<td>Support for DML, DDL within transactions</td>
<td>SQL_TXN_CAPABLE</td>
<td>SQL_TC_ALL</td>
</tr>
<tr>
<td>Options for setting transaction isolation levels</td>
<td>SQL_TXN_ISOLATION_OPTION</td>
<td>SQL_TXN_READ_UNCOMMITTED, ...</td>
</tr>
<tr>
<td>UNION support</td>
<td>SQL_UNION</td>
<td>SQL_U_UNION, SQL_U_UNION_ALL</td>
</tr>
<tr>
<td>Name of user connected to the data source</td>
<td>SQL_USER_NAME</td>
<td>(String containing the name)</td>
</tr>
<tr>
<td>Year of publication of the X/Open specification with which the driver complies</td>
<td>SQL_XOPEN_CLI_YEAR</td>
<td>1995</td>
</tr>
</tbody>
</table>
ODBC Scalar Functions

This section lists scalar functions that ODBC supports and are available to use in OpenEdge SQL statements, as described in the following sections:

- Scalar functions
- System functions

Scalar functions

Table 2, Table 3, and Table 4 list the scalar functions that ODBC supports. You can use these functions in SQL statements using the following syntax:

Syntax

```
(fn scalar-function)
```

`scalar-function` is one of the functions listed in the following tables. For example:

```
SELECT (fn UCASE(NAME)) FROM EMP
```

String functions

Table 1 lists the string functions that ODBC supports.

The string functions listed can take the following arguments:

- `string_exp` can be the name of a column, a string literal, or the result of another scalar function, where the underlying data type is `SQL_CHAR` or `SQL_VARCHAR`.
• start, length, and count can be the result of another scalar function or a literal numeric value, where the underlying data type is SQL_TINYINT, SQL_SMALLINT, or SQL_INTEGER.

The string functions are one-based; that is, the first character in the string is character 1.

Character string literals must be surrounded in single quotation marks.

### Table 1: Scalar string functions

<table>
<thead>
<tr>
<th>String function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII(string_exp)</td>
<td>ASCII code value of the leftmost character of string_exp as an integer.</td>
</tr>
<tr>
<td>BIT_LENGTH(string_exp) ODBC 3.0</td>
<td>The length in bits of the string expression.</td>
</tr>
<tr>
<td>CHAR(code)</td>
<td>The character with the ASCII code value specified by code. code should be between 0 and 255; otherwise, the return value is data-source dependent.</td>
</tr>
<tr>
<td>CHAR_LENGTH(string_exp) ODBC 3.0</td>
<td>The length in characters of the string expression, if the string expression is of a character data type; otherwise, the length in bytes of the string expression (the smallest integer not less than the number of bits divided by 8). (This function is the same as the CHARACTER_LENGTH function.)</td>
</tr>
<tr>
<td>CHARACTER_LENGTH(string_exp) ODBC 3.0</td>
<td>The length in characters of the string expression, if the string expression is of a character data type; otherwise, the length in bytes of the string expression (the smallest integer not less than the number of bits divided by 8). (This function is the same as the CHAR_LENGTH function.)</td>
</tr>
<tr>
<td>CONCAT(string_exp1,string_exp)</td>
<td>The string resulting from concatenating string_exp2 and string_exp1. The string is system dependent.</td>
</tr>
<tr>
<td>DIFFERENCE(string_exp2 and string_exp1)</td>
<td>An integer value that indicates the difference between the values returned by the SOUNDEX function for string_exp2 and string_exp1.</td>
</tr>
<tr>
<td>INSERT(string_exp1 , start,length,string_exp2)</td>
<td>A string where length characters have been deleted from string_exp1 beginning at start and where string_exp2 has been inserted into string_exp, beginning at start.</td>
</tr>
<tr>
<td>LCASE(string_exp)</td>
<td>Uppercase characters in string_exp converted to lowercase.</td>
</tr>
<tr>
<td>LEFT(string_exp, count)</td>
<td>The count of characters of string_exp.</td>
</tr>
</tbody>
</table>
### Table 1: Scalar string functions (2 of 2)

<table>
<thead>
<tr>
<th>String function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LENGTH</strong>(string_exp)</td>
<td>The number of characters in string_exp.</td>
</tr>
<tr>
<td><strong>LOCATE</strong>(string_exp1, string_exp2 [,start,])</td>
<td>The starting position of the first occurrence of string_exp1 within string_exp2. If start is not specified the search begins with the first character position in string_exp2. If start is specified, the search begins with the character position indicated by the value of start. The first character position in string_exp2 is indicated by the value 1. If string_exp1 is not found, 0 is returned.</td>
</tr>
<tr>
<td><strong>LTRIM</strong>(string_exp)</td>
<td>The characters of string_exp, with leading blanks removed.</td>
</tr>
<tr>
<td><strong>OCTET_LENGTH</strong>(string_exp)</td>
<td>The length in bytes of the string expression. The result is the smallest integer not less than the number of bits divided by 8.</td>
</tr>
<tr>
<td><strong>POSITION</strong>(character_exp IN character_exp)</td>
<td>The position of the first character expression in the second character expression. The result is an exact numeric with an implementation-defined precision and a scale of 0.</td>
</tr>
<tr>
<td><strong>REPEAT</strong>(string_exp, count)</td>
<td>A string composed of string_exp repeated count times.</td>
</tr>
<tr>
<td><strong>REPLACE</strong>(string_exp1, string_exp2, string_exp3)</td>
<td>Replaces all occurrences of string_exp2 in string_exp1 with string_exp3.</td>
</tr>
<tr>
<td><strong>RIGHT</strong>(string_exp, count)</td>
<td>The rightmost count of characters in string_exp.</td>
</tr>
<tr>
<td><strong>RTRIM</strong>(string_exp)</td>
<td>The characters of string_exp with trailing blanks removed.</td>
</tr>
<tr>
<td><strong>SPACE</strong>(count)</td>
<td>A string consisting of count spaces.</td>
</tr>
<tr>
<td><strong>SUBSTRING</strong>(string_exp, start, length)</td>
<td>A string derived from string_exp beginning at the character position start for length characters.</td>
</tr>
<tr>
<td><strong>UCASE</strong>(string_exp)</td>
<td>Lowercase characters in string_exp converted to uppercase.</td>
</tr>
</tbody>
</table>
**Numeric functions**

Table 2 lists the numeric functions that ODBC supports.

The numeric functions listed can take the following arguments:

- `numeric_exp` can be a column name, a numeric literal, or the result of another scalar function, where the underlying data type is `SQL_NUMERIC`, `SQL_DECIMAL`, `SQL_TINYINT`, `SQL_SMALLINT`, `SQL_INTEGER`, `SQL_FLOAT`, `SQL_REAL`, or `SQL_DOUBLE`.

- `float_exp` can be a column name, a numeric literal, or the result of another scalar function, where the underlying data type is `SQL_FLOAT`.

- `integer_exp` can be a column name, a numeric literal, or the result of another scalar function, where the underlying data type is `SQL_TINYINT`, `SQL_SMALLINT`, or `SQL_INTEGER`.

**Table 2: Scalar numeric functions**

<table>
<thead>
<tr>
<th>Numeric function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(numeric_exp)</td>
<td>Absolute value of numeric_exp.</td>
</tr>
<tr>
<td>ACOS(float_exp)</td>
<td>Arccosine of float_exp as an angle in radians.</td>
</tr>
<tr>
<td>ASIN(float_exp)</td>
<td>Arcsine of float_exp as an angle in radians.</td>
</tr>
<tr>
<td>ATAN(float_exp)</td>
<td>Arctangent of float_exp as an angle in radians.</td>
</tr>
<tr>
<td>ATAN2(float_exp1, float_exp2)</td>
<td>Arctangent of the x and y coordinates, specified by float_exp1 and float_exp2 as an angle in radians.</td>
</tr>
<tr>
<td>CEILING(numeric_exp)</td>
<td>Smallest integer greater than or equal to numeric_exp.</td>
</tr>
<tr>
<td>COS(float_exp)</td>
<td>Cosine of float_exp as an angle in radians.</td>
</tr>
<tr>
<td>COT(float_exp)</td>
<td>Cotangent of float_exp as an angle in radians.</td>
</tr>
<tr>
<td>DEGREES(numeric_exp)</td>
<td>Number of degrees converted from numeric_exp radians.</td>
</tr>
<tr>
<td>EXP(float_exp)</td>
<td>Exponential value of float_exp.</td>
</tr>
<tr>
<td>FLOOR(numeric_exp)</td>
<td>Largest integer less than or equal to numeric_exp.</td>
</tr>
<tr>
<td>LOG(float_exp)</td>
<td>Natural log of float_exp.</td>
</tr>
<tr>
<td>LOG10(float_exp)</td>
<td>Base 10 log of float_exp.</td>
</tr>
</tbody>
</table>
Date and time functions

Table 3 lists the date and time functions that ODBC supports.

The date and time functions listed can take the following arguments:

- `date-exp` can be a column name, a date or timestamp literal, or the result of another scalar function, where the underlying data type can be represented as `SQL_CHAR`, `SQL_VARCHAR`, `SQL_DATE`, or `SQL_TIMESTAMP`.

- `time_exp` can be a column name, a timestamp or timestamp literal, or the result of another scalar function, where the underlying data type can be represented as `SQL_CHAR`, `SQL_VARCHAR`, `SQL_TIME`, or `SQL_TIMESTAMP`.

### Table 3: Scalar numeric functions

<table>
<thead>
<tr>
<th>Numeric function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MOD(integer_exp1, integer_exp2)</code></td>
<td>Remainder of <code>integer_exp1</code> divided by <code>integer_exp2</code>.</td>
</tr>
<tr>
<td><code>PI()</code></td>
<td>Constant value of pi as a floating-point number.</td>
</tr>
<tr>
<td><code>POWER(numeric_exp, integer_exp)</code></td>
<td>Value of <code>numeric_exp</code> to the power of <code>integer_exp</code>.</td>
</tr>
<tr>
<td><code>RADIANS(numeric_exp)</code></td>
<td>Number of radians converted from <code>numeric_exp</code> degrees.</td>
</tr>
<tr>
<td><code>RAND([ integer_exp ]))</code></td>
<td>Random floating-point value using <code>integer_exp</code> as the optional seed value.</td>
</tr>
<tr>
<td><code>ROUND(numeric_exp, integer_exp)</code></td>
<td><code>numeric_exp</code> rounded to <code>integer_exp</code> places right of the decimal (left of the decimal if <code>integer_exp</code> is negative).</td>
</tr>
<tr>
<td><code>SIGN(numeric_exp)</code></td>
<td>Indicator of the sign of <code>numeric_exp</code>. If <code>numeric_exp</code> &lt; 0, -1 is returned. If <code>numeric_exp</code> = 0, 0 is returned. If <code>numeric_exp</code> &gt; 0, 1 is returned.</td>
</tr>
<tr>
<td><code>SIN(float_exp)</code></td>
<td>Sine of <code>float_exp</code>, where <code>float_exp</code> is an angle in radians.</td>
</tr>
<tr>
<td><code>SQRT(float_exp)</code></td>
<td>Square root of <code>float_exp</code>.</td>
</tr>
<tr>
<td><code>TAN(float_exp)</code></td>
<td>Tangent of <code>float_exp</code>, where <code>float_exp</code> is an angle in radians.</td>
</tr>
<tr>
<td><code>TRUNCATE(numeric_exp, integer_exp)</code></td>
<td><code>numeric_exp</code> truncated to <code>integer_exp</code> places right of the decimal. (If <code>integer_exp</code> is negative, truncation is to the left of the decimal.)</td>
</tr>
</tbody>
</table>

Date and time functions

Table 3 lists the date and time functions that ODBC supports.

The date and time functions listed can take the following arguments:

- `date-exp` can be a column name, a date or timestamp literal, or the result of another scalar function, where the underlying data type can be represented as `SQL_CHAR`, `SQL_VARCHAR`, `SQL_DATE`, or `SQL_TIMESTAMP`.

- `time_exp` can be a column name, a timestamp or timestamp literal, or the result of another scalar function, where the underlying data type can be represented as `SQL_CHAR`, `SQL_VARCHAR`, `SQL_TIME`, or `SQL_TIMESTAMP`. 
- `timestamp_exp` can be a column name; a time, date, or timestamp literal; or the result of another scalar function, where the underlying data type can be represented as SQL_CHAR, SQL_VARCHAR, SQL_TIME, SQL_DATE, or SQL_TIMESTAMP.

### Table 3: Date and time functions supported by ODBC (1 of 2)

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT_DATE() (ODBC 3.6)</td>
<td>Current date.</td>
</tr>
<tr>
<td>CURRENT_TIME[(time-precision)] (ODBC 3.6)</td>
<td>Current local time. The <code>time-precision</code> argument determines the seconds precision of the returned value.</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP[(timestamp-precision)] (ODBC 3.6)</td>
<td>Current local date and local time as a timestamp value. The <code>timestamp-precision</code> argument determines the seconds precision of the returned timestamp.</td>
</tr>
<tr>
<td>CURDATE()</td>
<td>Current date as a date value.</td>
</tr>
<tr>
<td>CURTIME()</td>
<td>Current local time as a time value.</td>
</tr>
<tr>
<td>DAYNAME(date_exp)</td>
<td>Character string containing a date source-specific name of the day for the day portion of <code>date_exp</code>.</td>
</tr>
<tr>
<td>DAYOFMONTH(date_exp)</td>
<td>Day of the month in <code>date_exp</code> as an integer value (1–31).</td>
</tr>
<tr>
<td>DAYOFWEEK(date_exp)</td>
<td>Day of the week in <code>date_exp</code> as an integer value (1–7).</td>
</tr>
<tr>
<td>DAYOFYEAR(date_exp)</td>
<td>Day of the year in <code>date_exp</code> as an integer value (1–366).</td>
</tr>
<tr>
<td>HOUR(time_exp)</td>
<td>Hour in <code>time_exp</code> as an integer value (0–23).</td>
</tr>
<tr>
<td>MINUTE(time_exp)</td>
<td>Minute in <code>time_exp</code> as an integer value (0–59).</td>
</tr>
<tr>
<td>MONTH(date_exp)</td>
<td>Month in <code>date_exp</code> as an integer value (1–366).</td>
</tr>
<tr>
<td>MONTHNAME(date_exp)</td>
<td>Character string containing the data source-specific name of the month.</td>
</tr>
<tr>
<td>NOW()</td>
<td>Current date and time as a timestamp value.</td>
</tr>
<tr>
<td>QUARTER(date_exp)</td>
<td>Quarter in <code>date_exp</code> as an integer value (1–4).</td>
</tr>
<tr>
<td>SECOND(time_exp)</td>
<td>Second in <code>date_exp</code> as an integer value (0–59).</td>
</tr>
</tbody>
</table>
System functions

Table 4 lists the scalar system functions that ODBC supports.

Table 4: Scalar system functions supported by ODBC

<table>
<thead>
<tr>
<th>System function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE()</td>
<td>Name of the database, corresponding to the connection handle (odbc)</td>
</tr>
<tr>
<td>IFNULL(exp, value)</td>
<td>value, if exp is null</td>
</tr>
<tr>
<td>ROWID(extension)</td>
<td>The row identifier of the current row in a table</td>
</tr>
<tr>
<td>USER()</td>
<td>Authorization name of the user</td>
</tr>
<tr>
<td>tenantname()</td>
<td>Name of the tenant corresponding to the connection handle (odbc)</td>
</tr>
</tbody>
</table>
Table 4: Scalar system functions supported by ODBC

<table>
<thead>
<tr>
<th>System function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>tenantid()</td>
<td>Unique ID associated with the name of the tenant.</td>
</tr>
<tr>
<td>tenantName_tbl([owner_name.</td>
<td>catalog_name. owner_name.]table_name)</td>
</tr>
<tr>
<td>tenantid_tbl([owner_name.</td>
<td>catalog_name. owner_name.]table_name)</td>
</tr>
</tbody>
</table>
Part 4

ESQL Reference

Embedded SQL
Embedded SQL

In OpenEdge Release 10, the ESQL interface is being deprecated. It is provided to help OpenEdge customers transition to the use of other interfaces. For the long term, you should use ODBC or JDBC to access SQL data. These newer interfaces offer better performance and access by many development and reporting tools, as described in the following sections:

- ESQL elements and statements
- ESQL elements and statements in Backus Naur Form (BNF)
- Compliance with industry standards

ESQL elements and statements

This section provides detailed information on OpenEdge SQL statements. A description for each statement provides the following information:

- Definition of the statement
- Syntax of the statement’s proper usage
- A code sample that shows how the statement works
- Any associated notes
- Authorization required in order to use the statement
- Related statements
BEGIN-END DECLARE SECTION

Declares variables and types used by the precompiler. Any variables you refer to in an embedded SQL statement must be declared in a DECLARE SECTION. This section starts with a BEGIN DECLARE SECTION statement and ends with an END DECLARE SECTION statement. Each variable must be declared as a host language data type.

Syntax

```sql
EXEC SQL BEGIN DECLARE SECTION
host_lang_type variable_name ;
.
.
EXEC SQL END DECLARE SECTION
```

A conventional C Language variable declaration. This form of variable declaration conforms to the ANSI standard for the C Language.

Syntax

```plaintext
{ char | short | long | float | double }
```

Example

```sql
EXEC SQL BEGIN DECLARE SECTION ;
short InvTransNum_v ;
short Qty_v ;
short OrderNum_v ;
EXEC SQL END DECLARE SECTION ;
```

Notes

- The C Language type int is not supported by ESQL. Type int maps to 16 or 32 bits, depending on the machine architecture. This can create rounding errors at run time, as values are passed across different machine architectures.

- Variables you declare in a BEGIN-END DECLARE SECTION can be used in C Language statements as if they are declared outside the DECLARE SECTION.

- The scope of variables follows host language scoping rules. The ESQL variables are not visible outside the file in which they are declared.

- DECLARE sections are permissible only where host language declarations are permissible in the host language syntax. This restriction is due to how DECLARE SECTION blocks are translated into the main body of host language declarations.

- Avoid DECLARE sections in header files that are included by more than one source file. This can cause duplicate variables with the same name.

- The form of the variable created by ESQL for each type is specified so that it can be manipulated from host language statements. Declaring variables allows you to use the variables in both host language and embedded SQL statements.
Authorization

None

Related statements

Static Array Types

CLOSE

Closing a cursor changes the state of the cursor from open to closed.

Syntax

EXEC SQL CLOSE cursor_name ;

cursor_name

An identifier named earlier in a DECLARE CURSOR statement and an OPEN CURSOR statement.

Example

EXEC SQL CLOSE dyncur ;
EXEC SQL COMMIT WORK ;

Notes

• Only a cursor in the open state can be set to the closed state.

• When a transaction ends, any cursors in the open state are automatically set to the closed state.

• When a cursor is in the closed state, you cannot perform FETCH, DELETE, or UPDATE operations using that cursor.

• It is good practice to close cursors explicitly.

Authorization

None

Related statements

DELETE, OPEN, FETCH, positioned UPDATE, positioned DELETE
CONNECT

Establishes a connection to a database. Optionally, the `CONNECT` statement can also specify a name for the connection and a `username` and `password` for authentication.

Syntax

```sql
CONNECT TO connect_string
[ AS connection_name ]
[ USER username ]
[ USING password ];
```

**connect_string**

**Syntax**

```sql
{ DEFAULT | db_name | db_type:T:host_name:port_num:db_name }
```

**Note:** Arguments to `CONNECT` must be either string literals enclosed in quotation marks or character-string host variables.

**connect_string**

Specifies to which database to connect. If the `CONNECT` statement specifies `DEFAULT`, SQL tries to connect to the environment-defined database, if any. The value of the `DB_NAME` environment variable specifies the default connect string.

The `connect_string` can be a simple database name or a complete `connect_string`. A complete connect string has the components shown in the following table:

<table>
<thead>
<tr>
<th>Connect string</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>db_type</code></td>
<td>Type of database. The only currently supported database type is <code>progress</code></td>
</tr>
<tr>
<td><code>T</code></td>
<td><code>T</code> directs the SQL engine to use the TCP/IP protocol</td>
</tr>
<tr>
<td><code>host_name</code></td>
<td>Name of the system where the database resides</td>
</tr>
<tr>
<td><code>port_num</code></td>
<td>Port number to use for the connection</td>
</tr>
<tr>
<td><code>db_name</code></td>
<td>Name of the database</td>
</tr>
</tbody>
</table>

**connection_name**

The name of the connection as either a character literal or host variable. If the `CONNECT` statement omits a connection name, the default is the name of the database. Connection names must be unique.
**username**

User name for authentication of the connection. SQL verifies the user name against a corresponding password before it connects to the database. The value of the `DH_USER` environment variable specifies the default user name. If `DH_USER` is not set, the value of the `USER` environment variable specifies the default user name.

**password**

Password for authentication of the connection. SQL verifies the password against a corresponding user name before it connects to the database.

The value of the `DH_PASSWD` environment variable determines the default password.

**Notes**

- Arguments to `CONNECT` must be either string literals enclosed in quotation marks or character string host variables.

- An application can connect to more than one database at a time, with a maximum of 10 connections. However, the application can actually gain access to only one database at a time. The database name specified in the `CONNECT` statement becomes the active one.

- If an application executes a SQL statement before connecting to a database, an attempt is made to connect to the environment-defined database, if any. If the connection is successful, the SQL statement is executed on that database.

**Examples**

The following example illustrates the `CONNECT` statement:

```sql
CONNECT TO "salesdb" AS "sales_conn";
CONNECT TO "progress:T:localhost:custdb" AS "cust_conn";
CONNECT TO DEFAULT;
```

- The first statement shown connects to the `salesdb` database on the local system.
- The second statement connects to the `custdb` database on the local system.
- The last statement connects to the environment-defined database by default.

**Authorization**

None

**Related statements**

`DISCONNECT`, `SET CONNECTION`
DECLARE CURSOR

Associates a cursor with a static query or a prepared dynamic query statement. The query or the prepared statement can have references to host variables.

Syntax

DECLARE cursor_name CURSOR FOR
    { query_expression [ ORDER BY clause ] [ FOR UPDATE clause ]
    | prepared_statement_name
    } ;

cursor_name

A name you assign to the cursor. The name must meet the requirements for an identifier.

query_expression [ ORDER BY clause ] [ FOR UPDATE clause ]

A complete query expression.

prepared_statement_name

The name assigned to a prepared SQL statement in an earlier PREPARE statement.

Examples

EXEC SQL WHENEVER SQLERROR GOTO selerr ;
EXEC SQL DECLARE stcur CURSOR FOR
    SELECT InvTransNum, Qty, OrderNum FROM PUB.InventoryTrans ;
EXEC SQL OPEN stcur ;
EXEC SQL WHENEVER NOT FOUND GOTO seldone ;

EXEC SQL WHENEVER SQLERROR GOTO selerr ;
EXEC SQL PREPARE stmtid from :sel_stmt_v ;
EXEC SQL DECLARE dyncur CURSOR FOR stmtid ;
EXEC SQL OPEN dyncur ;
EXEC SQL WHENEVER NOT FOUND GOTO seldone ;

Notes

- You must declare a cursor before any OPEN, FETCH, or CLOSE statement.
- The scope of the cursor declaration is the entire source file in which it is declared. The operations on the cursor, such as OPEN, CLOSE, and FETCH statements, can occur only within the same compilation unit as the cursor declaration.
- The use of a cursor allows the execution of the positioned forms of the UPDATE and DELETE statements.
• If the DECLARE statement corresponds to a static SQL statement with parameter references:
  – The DECLARE statement must be executed before each execution of an OPEN statement for the same cursor.
  – The DECLARE statement and the OPEN statement that follows must occur within the same transaction within the same task.
  – If the statement contains parameter references to automatic variables or function arguments, the DECLARE statement and the following OPEN statement for the same cursor must occur within the same C function.

Authorization

None

Related statements

PREPARE, OPEN, FETCH, CLOSE SELECT

DESCRIBE

Writes information about a prepared statement to the SQL Descriptor Area (SQLDA). You use a DESCRIBE statement in a series of steps that allows a program to accept SQL statements at run time. Dynamically generated statements are not part of a program’s source code; they are generated at run time.

There are two forms of the DESCRIBE statement:

• The DESCRIBE BIND VARIABLES statement writes information about input variables in an expression to an SQLDA. These variables can be substitution variable names or parameter markers.

• The DESCRIBE SELECT LIST statement writes information about select list items in a prepared SELECT statement to an SQLDA.

Syntax

```
DESCRIBE [ BIND VARIABLES | SELECT LIST ] FOR statement_name
INTO input_sqlda_name ;
```

The SQLDA is a host language data structure used in dynamic SQL processing. DESCRIBE statements write information about the number, data types, and sizes of input variables or select list items to SQLDA structures. Program logic then processes that information to allocate storage. OPEN, EXECUTE, and FETCH statements read the SQLDA structures for the addresses of the allocated storage.
DESCRIBE BIND VARIABLES

Writes information about any input variables in the prepared statement to an input SQLDA structure.

Syntax

```
DESCRIBE BIND VARIABLES FOR statement_name INTO input_sqlda_name ;
```

*statement_name*

The name of an input SQL statement to be processed using dynamic SQL steps. Typically, this is the same *statement_name* used in the PREPARE statement.

*input_sqlda_name*

The name of the SQLDA structure to which DESCRIBE will write information about input variables. Input variables represent values supplied to INSERT and UPDATE statements at run time, and to predicates in DELETE, UPDATE, and SELECT statements at run time.

To utilize the DESCRIBE BIND VARIABLES statement in your application, issue statements in the following order:

1. PREPARE
2. DESCRIBE BIND VARIABLES
3. EXECUTE or OPEN CURSOR

The DESCRIBE BIND VARIABLES statement writes the number of input variables to the sqld_nvars field of the SQLDA. If the sqld_size field of the SQLDA is not equal to or greater than this number, DESCRIBE writes the value as a negative number to sqld_nvars. Design your application to check sqld_nvars for a negative number to determine if a particular SQLDA is large enough to process the current input statement.

Input variables in dynamic SQL statements are identified by parameter markers or as substitution names.

Authorization

None

Related statements

PREPARE, DECLARE CURSOR, OPEN, FETCH, CLOSE
DESCRIBE SELECT LIST

Writes information about select list items in a prepared SELECT statement to an output SQLDA structure.

Syntax

```
DESCRIBE SELECT LIST FOR statement_name INTO output_sqlda_name;
```

- **statement_name**
  - The name of a SELECT statement to be processed using dynamic SQL steps. Typically, this is the same `statement_name` as in the PREPARE statement.

- **output_sqlda_name**
  - The name of the SQLDA structure to which DESCRIBE will write information about select list items.

Note

Select list items are column names and expressions in a SELECT statement. A FETCH statement writes the values returned by a SELECT statement to the addresses stored in an output SQLDA.

To utilize the DESCRIBE SELECT LIST statement in your application, issue statements in the following order:

1. DECLARE CURSOR
2. PREPARE
3. OPEN
4. DESCRIBE SELECT LIST
5. FETCH

A DESCRIBE SELECT LIST statement writes the number of select list items to the `sqlid_nvars` field of an output SQLDA. If the `sqlid_size` field of the SQLDA is not equal to or greater than this number, DESCRIBE writes the value as a negative number to `sqlid_nvars`. Design your application to check `sqlid_nvars` for a negative number to determine if a particular output SQLDA is large enough to process the current SELECT statement.

Authorization

None

Related statements

PREPARE, DECLARE CURSOR, OPEN, FETCH, CLOSE
**DISCONNECT**

Terminates the connection between an application and the database to which it is connected.

**Syntax**

```sql
DISCONNECT { 'connection_name' | CURRENT | ALL | DEFAULT } ;
```

*connection_name*

The name of the connection as either a character literal or host variable.

*CURRENT*

Disconnects the current connection.

*ALL*

Disconnects all established connections.

*DEFAULT*

Disconnects the connection to the default database.

**Examples**

This example illustrates `CONNECT TO AS connection_name` and `DISCONNECT connection_name`:

```sql
EXEC SQL
   CONNECT TO 'progress:T:localhost:6745:salesdb' AS 'conn_1' ;
/*
** C Language and embedded SQL application processing against the
** database in the connect_string
*/
.
.
.
EXEC SQL
   DISCONNECT 'conn_1' ;
```

The following example illustrates `CONNECT TO DEFAULT` and `DISCONNECT DEFAULT`:

```sql
EXEC SQL
   CONNECT TO DEFAULT ;
/*
** C Language and embedded SQL application processing against the
** database in the connect_string
*/
.
.
.
EXEC SQL
   DISCONNECT DEFAULT ;
```
After you issue **DISCONNECT ALL** there is no current connection. The following example disconnects all database connections:

```sql
EXEC SQL
    DISCONNECT ALL;
```

The following example illustrates the **CONNECT**, **SET CONNECTION**, and **DISCONNECT** statements in combination using these steps:

1. **CONNECT TO** `connect_string` **AS** `connection_name`, which establishes a connection to the database in the `connect_string`; the connection has the name 'conn_1'.

2. **CONNECT TO** DEFAULT, which establishes a connection to the DEFAULT database and sets this connection current.

3. **DISCONNECT** DEFAULT, which disconnects the connection to the DEFAULT database.

4. **SET CONNECTION** `connection_name`, which sets the 'conn_1' connection current.

5. **DISCONNECT** CURRENT, which disconnects the 'conn_1' connection.

```sql
/*
 * 1. CONNECT TO 'connect_string'
 */
EXEC SQL
    CONNECT TO 'progress:T:localhost:6745:salesdb' AS 'conn_1' ;
/*
 * 2. CONNECT TO DEFAULT. This suspends the conn_1 connection
 *    and sets the DEFAULT connection current
 */
EXEC SQL
    CONNECT TO DEFAULT ;
/*
 * Application processing against the DEFAULT database
 */
/*
 * 3. DISCONNECT DEFAULT
 */
EXEC SQL
    DISCONNECT DEFAULT ;
/*
 * 4. Set the first connection, conn_1, current
 */
EXEC SQL
    SET CONNECTION conn_1 ;
/*
 * Application processing against the database in the connect_string
 */
/*
 * 5. DISCONNECT the conn_1 connection, which is the current connection.
 */
EXEC SQL
    DISCONNECT CURRENT ;
```
Notes

- When you specify `DISCONNECT connection_name` or `DISCONNECT CURRENT` and there is also an established connection to the `DEFAULT` database, the connection to the `DEFAULT` database becomes the current connection. If there is no `DEFAULT` database, there is no current connection after the SQL engine processes the `DISCONNECT`.

- The `DISCONNECT DEFAULT` statement terminates the connection to the `DEFAULT` database. If this connection is the current connection, there is no current connection after this `DISCONNECT` statement is executed.

Authorization

None

Related statements

CONNECT, SET CONNECTION

EXEC SQL delimiter

In C Language programs, you must precede embedded SQL statements with the `EXEC SQL` delimiter so that the precompiler can distinguish statements from the host language statements.

Note: Constructs within a `BEGIN-END DECLARE SECTION` do not require the `EXEC SQL` delimiter.

Syntax

```
EXEC SQL sql_statement;
```

`sql_statement`

An SQL statement to be processed by the ESQL precompiler. You must terminate each SQL statement with a semicolon to mark the end of the statement.

Example

```
EXEC SQL WHENEVER SQLERROR GOTO selerr ;
EXEC SQL PREPARE stmtid from :sel_stmt_v ;
EXEC SQL DECLARE dyncur CURSOR FOR stmtid ;
EXEC SQL OPEN dyncur ;
EXEC SQL WHENEVER NOT FOUND GOTO seldone ;
```

Note

In general, the ESQL precompiler does not parse host language statements and therefore does not detect any syntax or semantic errors in host language statements. The exceptions to this rule are:

- Recognition of host language blocks. The precompiler recognizes host language blocks in order to determine the scope of variables and types.

- Constants defined with the `#define` preprocessor command. To evaluate these constants, the ESQL precompiler invokes the C language preprocessor before beginning embedded SQL processing.
EXECUTE

Executes the statement specified in `statement_name`.

**Syntax**

```
EXECUTE statement_name
  [ USING
    { [ SQL ] DESCRIPTOR structure_name
      | :host_variable [ [ INDICATOR ] :ind_variable ] , ... }
  ] ;
```

- **statement_name**
  - Name of the prepared SQL statement.

- **structure_name**
  - Name of an SQL descriptor area (SQLDA).

**Example**

```sql
/*
** Process the non-SELECT input statement
**  PREPARE the statement
**  EXECUTE the prepared statement
**  COMMIT WORK
*/

EXEC SQL PREPARE dynstmt FROM :sql_stmt_v ;
EXEC SQL EXECUTE dynstmt ;
EXEC SQL COMMIT WORK ;
```

**Notes**

- A statement must be processed with a **PREPARE** statement before it can be processed with an **EXECUTE** statement.

- A prepared statement can be executed multiple times in the same transaction. Typically, each call to the **EXECUTE** statement supplies a different set of host variables.

- If there is no **DESCRIPTOR** in the **USING** clause, the **EXECUTE** statement is restricted to the number of variables specified in the host variable list. The number and type of the variables must be known at compile time. The host variables must be declared in the **DECLARE SECTION** before they can be used in the **USING** clause of the **EXECUTE** statement.

- If there is a **DESCRIPTOR** in the **USING** clause, the program can allocate space for the input host variables at run time.

**Related statements**

```
EXECUTE IMMEDIATE, PREPARE
```
EXECUTE IMMEDIATE

Executes the statement specified in a `statement_string` or `host_variable`.

Syntax

```
EXECUTE IMMEDIATE { statement_string | host_variable };
```

`statement_string`

The character string form of the statement.

`host_variable`

The host variable name which passes values between the database and the application program.

Notes

- The character string form of the statement is referred to as a statement string. An `EXECUTE IMMEDIATE` statement accepts either a statement string or a host variable as input.

- A statement string must not contain host variable references or parameter markers.

- A statement string must not begin with `EXEC SQL` delimiter and must not end with a semicolon.

- When an `EXECUTE IMMEDIATE` statement is executed, the SQL engine parses the statement and checks it for errors. Any error in the execution of the statement is reported in the SQLCA.

- If the same SQL statement is to be executed multiple times, it is more efficient to use `PREPARE` and `EXECUTE` statements, rather than an `EXECUTE IMMEDIATE` statement.

Related statement

`EXECUTE`

FETCH

Moves the position of the cursor to the next row of the active set and fetches the column values of the current row into the specified host variables.

Syntax

```
FETCH cursor_name
{ USING SQL DESCRIPTOR structure_name
   | INTO :host_var_ref [[ INDICATOR ] :ind_var_ref ] , ...
} ;
```

`cursor_name`

A name identified in an earlier `DECLARE CURSOR` statement and an `OPEN CURSOR` statement.
USING SQL DESCRIPTOR structure_name

Directs the SQL engine to FETCH data into storage addressed by an SQLDA structure.

INTO :host_var_ref [ [ INDICATOR ] :ind_var_ref ]

Directs the SQL engine to FETCH data into the identified host variables, and to set values in the identified indicator variables.

Example

/*
** One way to limit the number of rows returned is to
** set a new value for "j" here. As supplied in the SPORTS2000 database,
** the PUB.InventoryTrans table contains 75 rows.
*/

j = 100;
for (i = 0; i < j; i++)
{
  EXEC SQL FETCH dyncur INTO
    :int_p1_v, :int_p2_v, :char_p_v ;
  if (i == 0)
    {
    printf (" 1st col  2nd col  3rd col ");
    printf (" -------  -------  ------- ");
    }
    printf (" %d  %d  %s ",
              int_p1_v, int_p2_v, char_p_v) ;
}

Notes

- A FETCH operation requires that the cursor be open.

- The positioning of the cursor for each FETCH operation is as follows:
  
  - The first time you execute a FETCH statement after opening the cursor, the cursor is positioned to the first row of the active set.
  
  - Subsequent FETCH operations advance the cursor position in the active set. The next row becomes the current row.
  
  - When the current row is deleted using a positioned DELETE statement, the cursor is positioned before the row after the deleted row in the active set.

- The cursor can only be moved forward in the active set by executing FETCH statements. To move the cursor to the beginning of the active set, you must CLOSE the cursor and OPEN it again.

- If the cursor is positioned on the last row of the active set or if the active set does not contain any rows, executing a FETCH will return the status code SQL_NOT_FOUND in the SQLDA.

- After a successful FETCH, the total row count fetched so far for this cursor is returned in sqlca.sqlerrd[2]. The count is set to zero after an OPEN cursor operation.
• You can FETCH multiple rows in one FETCH operation by using array variables in the INTO clause. The SQL_NOT_FOUND status code is returned in the SQLCA when the end of the active set is reached, even if the current FETCH statement returns one or more rows.

• If you use array variables in a FETCH statement, the array sizes are set to the number of rows fetched after the FETCH statement is executed.

Authorization

None

Related statements

DECLARE CURSOR, OPEN, CLOSE

GET DIAGNOSTICS

Retrieves information about the execution of the previous SQL statement from the SQL diagnostics area. The diagnostics area is a data structure that contains information about the execution status of the most recent SQL statement. Specifically, GET DIAGNOSTICS extracts information about the SQL statement as a whole from the SQL diagnostics area’s header component.

Note: The GET DIAGNOSTICS EXCEPTION number extracts detail information.

Syntax

GET DIAGNOSTICS
   :param = header_info_item
   [ , :param = header_info_item ] , ... ;

param

A host-language variable to receive the information returned by the GET DIAGNOSTICS statement. The host-language program must declare a param compatible with the SQL data type of the information item.

header_info_item

One of the following keywords, which returns associated information about the diagnostics area or the SQL statement:

Syntax

| NUMBER | MORE | COMMAND_FUNCTION | DYNAMIC_FUNCTION | ROW_COUNT

NUMBER

The number of detail areas in the diagnostics area. Currently, NUMBER is always 1. NUMBER is type NUMERIC with a scale of 0.
MORE

A one-character string with a value of Y (all conditions are detailed in the diagnostics area) or N (all conditions are not detailed) that tells whether the diagnostics area contains information on all the conditions resulting from the statement.

COMMAND_FUNCTION

Contains the character-string code for the statement (as specified in the SQL standard), if the statements is a static SQL statement. If the statement is a dynamic statement, contains the character string EXECUTE or EXECUTE IMMEDIATE.

DYNAMIC_FUNCTION

Contains the character-string code for the statement (as specified in the SQL standard). For dynamic SQL statements only (as indicated by EXECUTE or EXECUTE IMMEDIATE in the COMMAND_FUNCTION item).

ROW_COUNT

The number of rows affected by the SQL statement.

Example

The GET DIAGNOSTICS example extracts header information about the last SQL statement executed. The information is assigned to host variables that are defined in the DECLARE SECTION of an embedded SQL program, as shown in the following example:

```
GET DIAGNOSTICS :num = NUMBER, :cmdfunc = COMMAND_FUNCTION ;
```

The GET DIAGNOSTICS statement itself does not affect the contents of the diagnostics area. This means applications can issue multiple GET DIAGNOSTICS statements to retrieve different items of information about the same SQL statement.

Related statements

GET DIAGNOSTICS EXCEPTION, WHENEVER
GET DIAGNOSTICS EXCEPTION

Retrieves information about the execution of the previous SQL statement from the SQL diagnostics area. The diagnostics area is a data structure that contains information about the execution status of the most recent SQL statement. Specifically, GET DIAGNOSTICS EXCEPTION extracts information about the SQL statement as a whole from the SQL diagnostics area’s detail component.

The detail area contains information for a particular condition (an error, warning, or success condition) associated with execution of the last SQL statement. The diagnostics area can potentially contain multiple detail areas corresponding to multiple conditions generated by the SQL statement described by the header. The SQL diagnostics area currently supports only one detail area.

**Note:** The GET DIAGNOSTICS statement extracts header information.

Syntax

```sql
GET DIAGNOSTICS EXCEPTION number
 :param = detail_info_item
    [, :param = detail_info_item ], ... ;
```

**EXCEPTION number**

Specifies that GET DIAGNOSTICS EXCEPTION extracts detail information. *number* specifies which of multiple detail areas GET DIAGNOSTICS extracts. Currently, *number* must be the integer 1.

**param**

Receives the information returned by the GET DIAGNOSTICS EXCEPTION statement. The host-language program must declare a *param* compatible with the SQL data type of the information item.
**detail_info_item**

One of the following keywords, which returns associated information about the particular error condition:

**Syntax**

<table>
<thead>
<tr>
<th>CONDITION_NUMBER</th>
<th>RETURNED_SQLSTATE</th>
<th>CLASS_ORIGIN</th>
<th>SUBCLASS_ORIGIN</th>
<th>ENVIRONMENT_NAME</th>
<th>CONNECTION_NAME</th>
<th>CONSTRAINT_CATALOG</th>
<th>CONSTRAINT_SCHEMA</th>
<th>CONSTRAINT_NAME</th>
<th>CATALOG_NAME</th>
<th>SCHEMA_NAME</th>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
<th>CURSOR_NAME</th>
<th>MESSAGE_TEXT</th>
<th>MESSAGE_LENGTH</th>
<th>MESSAGE_OCTET_LENGTH</th>
</tr>
</thead>
</table>

**CONDITION_NUMBER**

The sequence of this detail area in the diagnostics area. Currently, **CONDITION_NUMBER is always 1**.

**RETURNED_SQLSTATE**

The **SQLSTATE** value that corresponds to the condition.

**CLASS_ORIGIN**

The general type of error. For example, connection exception or data exception.

**SUBCLASS_ORIGIN**

The specific error. Usually the same as the message text.

**ENVIRONMENT_NAME**

Not currently supported.

**CONNECTION_NAME**

Not currently supported.

**CONSTRAINT_CATALOG**

Not currently supported.
ESQL elements and statements

CONSTRAINT_SCHEMA

Not currently supported.

CONSTRAINT_NAME

Not currently supported.

CATALOG_NAME

Not currently supported.

SCHEMA_NAME

Not currently supported.

TABLE_NAME

The name of the table, if the error condition involves a table.

COLUMN_NAME

The name of the affected columns, if the error condition involves a column.

CURSOR_NAME

Not currently supported.

MESSAGE_TEXT

The associated message text for the error condition.

MESSAGE_LENGTH

The length in characters of the message in the MESSAGE_LENGTH item.

MESSAGE_OCTET_LENGTH

Not currently supported.

Example

The GET DIAGNOSTICS EXCEPTION example extracts detailed information into host variables that are defined in the DECLARE SECTION of an embedded SQL program:

```
GET DIAGNOSTICS EXCEPTION :num :sstate = RETURNED_SQLSTATE,
                      :msgtxt = MESSAGE_TEXT ;
```

Note

The GET DIAGNOSTICS statement itself does not affect the contents of the diagnostics area. This means applications can issue multiple GET DIAGNOSTICS statements to retrieve different items of information about the same SQL statement.

Related statements

GET DIAGNOSTICS, WHENEVER
OPEN

Executes a prepared SQL query associated with a cursor and creates a result set composed of the rows that satisfy the query. This set of rows is called the active set.

Syntax

```
OPEN cursor_name
[ USING { SQL DESCRIPTOR structure_name
  | :host_variable [ INDICATOR ] :ind_variable ] , ... } ] ;
```

cursor_name

An identifier named in an earlier DECLARE CURSOR statement.

USING [ SQL DESCRIPTOR structure_name]

Directs the SQL engine to create the result set in storage addressed by the identified SQLDA structure.

USING :host_variable [ INDICATOR ] :ind_variable ]

Directs the SQL engine to create the result set in storage addressed by host variables.

Example

```
/*
** 5.  Name WHENEVER routine to handle SQLERROR.
**
** 6.  DECLARE cursor for the SELECT statement.
**     NOTE: You must set input parameter values before OPEN CURSOR.
**     The static query in this program does not have input parameters.
**
** 7.  OPEN the cursor.
**     NOTE: For static statements, if a DECLARE CURSOR
**     statement contains references to automatic variables,
**     the OPEN CURSOR statement must be in the same C function.
**
** 8.  Name WHENEVER routine to handle NOT FOUND condition.
*/

EXEC SQL WHENEVER SQLERROR GOTO selerr ;
EXEC SQL DECLARE stcur CURSOR FOR
  SELECT InvTransNum, Qty,
  OrderNum FROM PUB.InventoryTrans ;
EXEC SQL OPEN stcur ;
EXEC SQL WHENEVER NOT FOUND GOTO seldone ;
```

Notes

- Executing an OPEN cursor statement sets the cursor to the open state.
- After the OPEN cursor statement is executed, the cursor is positioned just before the first row of the active set.
- For a single execution of an OPEN cursor statement, the active set does not change and the host variables are not re-examined.
- If you elect to retrieve a new active set and a host variable value has changed, you must CLOSE the cursor and OPEN it again.
ESQL elements and statements

- Execution of a COMMIT statement or ROLLBACK statement implicitly closes the cursors that have been opened in the current transaction.

- It is good practice to CLOSE cursors explicitly.

- When a cursor is in the open state, executing an OPEN statement on that cursor results in an error.

- If a DECLARE cursor statement is associated with a static SQL statement containing parameter markers, the following requirements apply:
  - You must execute the DECLARE statement before executing the OPEN statement for that cursor.
  - The DECLARE cursor statement and the OPEN statement for the same cursor must occur in the same transaction.
  - If the statement contains parameter markers for stack variables, the DECLARE cursor statement and the following OPEN statement for the same cursor must occur in the same C Language function.

Authorization

Must have DBA privilege of SELECT privilege on all the tables and views referenced in the SELECT statement associated with the cursor.

Related statements

DECLARE CURSOR, CLOSE, FETCH, positioned UPDATE, positioned DELETE

PREPARE

Parses and assigns a name to an ad hoc or dynamically generated SQL statement for execution. You use a PREPARE statement in a series of steps that allows a program to accept or generate SQL statements at run time.

Syntax

```
PREPARE statement_name FROM statement_string;
```

`statement_name`

A name for the dynamically generated statement. DESCRIBE, EXECUTE, and DECLARE CURSOR statements refer to this `statement_name`. A `statement_name` must be unique in a program.

`statement_string`

Specifies the SQL statement to be prepared for dynamic execution. You can use either the name of a C Language string variable containing the SQL statement, or you can specify the SQL statement as a quoted literal. If there is an SQL syntax error, the PREPARE statement returns an error in the SQLCA.
Syntax

\[
\{ :host_variable \mid quoted_literal \}
\]

Examples

The first example is a code fragment from the DynUpd function in sample program 3DynUpd.pc, which illustrates dynamic processing of an UPDATE statement:

```sql
EXEC SQL PREPARE dynstmt FROM :sql_stmt_v ;
EXEC SQL EXECUTE dynstmt ;
EXEC SQL COMMIT WORK ;
```

This example is a code fragment from the DynSel function in sample program 4DynSel.pc, which illustrates dynamic processing of a SELECT statement:

```sql
EXEC SQL PREPARE stmtid from :sel_stmt_v ;
EXEC SQL DECLARE dyncur CURSOR FOR stmtid ;
EXEC SQL OPEN dyncur ;
EXEC SQL WHENEVER NOT FOUND GOTO seldone ;
```

Notes

- A statement string can have one or more references to input variables. These variables represent values supplied at run time to:
  - INSERT and UPDATE statements
  - Predicates in DELETE, UPDATE, and SELECT statements

- A program supplies an input variable to a PREPARE statement either as a substitution name or as a parameter marker. For example:
  - A substitution name is a name preceded by a colon (:) in a statement string. This name does not refer to a C Language variable, but acts only as a placeholder for input variables.
  - A parameter marker is a question mark (?) in the statement string, serving as a placeholder for input variables.
ESQL elements and statements

- The **USING** clauses of **EXECUTE** and **OPEN** statements identify host language storage. The values in this storage expand a statement string, replacing a substitution name or a parameter marker. You can design your program to execute the same prepared statement many times in a transaction, supplying different values for input variables for each execution. If you **COMMIT** or **ROLLBACK** the transaction, you must **PREPARE** the statement string again.

**Authorization**

Must have DBA privileges or authorization for the SQL statement being used.

**Related statements**

**EXECUTE**, **OPEN**, **CLOSE**, **FETCH**

---

**SET CONNECTION**

Switches the application from one established connection to another. This resumes the connection associated with the specified **connection_name**, restoring the context of that database connection to the same state it was in when suspended.

**Syntax**

```
SET CONNECTION { 'connection_name' | DEFAULT } ;
```

**connection_name**

The name of the connection as either a character literal or host variable. If the **SET CONNECTION** statement omits a connection name, the default is the name of the database. Connection names must be unique.

**DEFAULT**

Sets the **DEFAULT** connection as the current connection.

**Examples**

The first example shows how to establish a database as the current database:

```
EXEC SQL
SET CONNECTION 'conn_1';
```

The **SET CONNECTION** command sets the database associated with the connection named **conn_1** to the status of current database. The connection named **conn_1** must be associated with an established connection. Use **SET CONNECTION DEFAULT** to set current the database associated with the **DEFAULT** connection. In this example, the statement suspends the **conn_1** connection, which had been current:

```
EXEC SQL
SET CONNECTION DEFAULT ;
```

See also the last example for the **DISCONNECT** statement, which illustrates the **CONNECT**, **SET CONNECTION**, and **DISCONNECT** statements in combination.
Authorization

None

Related statements

CONNECT, DISCONNECT

WHENEVER

Specifies actions for three SQL run-time exceptions.

Syntax

WHENEVER

{ NOT FOUND | SQLERROR | SQLWARNING }

{ STOP | CONTINUE | { GOTO | GO TO } host_lang_label } ;

Examples

/*
** Name WHENEVER routine to handle SQLERROR condition.
*/
EXEC SQL WHENEVER SQLERROR GOTO mainerr ;

/*
** Name WHENEVER routines to handle NOT FOUND and SQLERROR
*/
EXEC SQL WHENEVER SQLERROR GOTO nodyn ;
EXEC SQL WHENEVER NOT FOUND GOTO nodyn ;
Notes

- You can place multiple \texttt{WHENEVER} statements for the same exception in a source file. Each \texttt{WHENEVER} statement overrides the previous \texttt{WHENEVER} statement specified for the same exception.

- Correct operation of a \texttt{WHENEVER} statement with a \texttt{GOTO host\_language\_label} or a \texttt{GO TO host\_language\_label} is subject to the scoping rules of the C Language. The \texttt{host\_language\_label} must be within the scope of all SQL statements for which the action is active. The \texttt{GO TO} or \texttt{GOTO} action is active starting from the corresponding \texttt{WHENEVER} statement until another \texttt{WHENEVER} statement for the same exception, or until end of the file.

Authorization

None

Related statements

\texttt{FETCH}

\begin{table}
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{ESQL elements and statements in Backus Naur Form (BNF)} \\
\hline
\textbf{BEGIN-END DECLARE SECTION} \\
\hline
\textbf{Syntax} \\
\hline
\texttt{begin declare section ::=
EXEC SQL BEGIN DECLARE SECTION
host\_lang\_type variable\_name ;
. \\
. \\
. \\
END DECLARE SECTION ::= 
EXEC SQL END DECLARE SECTION} \\
\hline
\end{tabular}
\end{table}

\begin{table}
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Host Language Type} \\
\hline
\textbf{Syntax} \\
\hline
\texttt{host\_language\_type ::= 
\{ char \\
| short \\
| long \\
| float \\
| double \\
\} \\
\hline
\end{tabular}
\end{table}
CLOSE
Syntax

```
close ::= EXEC SQL CLOSE cursor_name ;
```

CONNECT
Syntax

```
connect statement ::= CONNECT TO connect_string
                  [ AS connection_name ]
                  [ USER user_name ]
                  [ USING password ];
```

CONNECT STRING
Syntax

```
connect_string ::= { DEFAULT | db_name
                  | db_type:T:host_name:port_num:db_name }
```

DECLARE CURSOR
Syntax

```
declare cursor ::= EXEC SQL DECLARE cursor_name CURSOR FOR
                  { query_expr [ ORDER BY clause ] [ FOR UPDATE clause ]
                    | prepared_statement_name
                  } ;
```

DESCRIBE BIND VARIABLES
Syntax

```
describe bind variables ::= EXEC SQL DESCRIBE BIND VARIABLES FOR statement_name
                          INTO input_sqlda_name ;
```

DESCRIBE SELECT LIST
Syntax

```
describe select list ::= EXEC SQL DESCRIBE SELECT LIST FOR statement_name
                      INTO output_sqlda_name ;
```
**DISCONNECT**

Syntax

```
disconnect statement ::= 
   DISCONNECT { 'connection_name' | CURRENT | ALL | DEFAULT } ;
```

**EXEC SQL**

Syntax

```
EXEC SQL ::= 
   EXEC SQL  sql_statement ;
```

**EXECUTE**

Syntax

```
EXECUTE ::= 
   EXEC SQL EXECUTE  statement_name 
     [ USING { [ SQL ] DESCRIPTOR structure_name 
          | :host_variable [ [ INDICATOR ] :ind_variable ] , ... } 
     ] ;
```

**EXECUTE IMMEDIATE**

Syntax

```
EXECUTE IMMEDIATE ::= 
   EXEC SQL EXECUTE IMMEDIATE 
     { statement_string | host_variable } ;
```

**FETCH**

Syntax

```
fetch ::= 
   EXEC SQL FETCH  cursor_name 
     { USING SQL DESCRIPTOR structure_name 
       | INTO :host_var_ref [ [ INDICATOR ] :ind_var_ref ] , ... 
     } ;
```

**GET DIAGNOSTICS**

Syntax

```
get diagnostics statement ::= 
   GET DIAGNOSTICS  
     :param = header_info_item 
     [ , :param = header_info_item ] , ... 
   ;
```
Header Info Item

Syntax

```
header_info_item ::= 
{ NUMBER 
  | MORE 
  | COMMAND_FUNCTION 
  | DYNAMIC_FUNCTION 
  | ROW_COUNT 
}
```

GET DIAGNOSTICS EXCEPTION

Syntax

```
get diagnostics exception statement ::= 
GET DIAGNOSTICS EXCEPTION number 
  :param = detail_info_item 
  [, :param = detail_info_item ], ... 
;
```

Detail Info Item

Syntax

```
detail_info_item ::= 
{ CONDITION_NUMBER 
  | RETURNED_SQLSTATE 
  | CLASS_ORIGIN 
  | SUBCLASS_ORIGIN 
  | ENVIRONMENT_NAME 
  | CONNECTION_NAME 
  | CONSTRAINT_CATALOG 
  | CONSTRAINT_SCHEMA 
  | CONSTRAINT_NAME 
  | CATALOG_NAME 
  | SCHEMA_NAME 
  | TABLE_NAME 
  | COLUMN_NAME 
  | CURSOR_NAME 
  | MESSAGE_TEXT 
  | MESSAGE_LENGTH 
  | MESSAGE_OCTET_LENGTH 
}
```
OPEN
Syntax

```
open ::= EXEC SQL OPEN cursor_name
       [ USING { [ SQL ] DESCRIPTOR structure_name
       | : host_variable [ [ INDICATOR ] : ind_variable ] , ... } ];
```

PREPARE
Syntax

```
prepare ::= EXEC SQL PREPARE statement_name FROM statement_string ;
```

SET CONNECTION
Syntax

```
set connection statement ::= SET CONNECTION { 'connection_name' | DEFAULT } ;
```

SET TRANSACTION ISOLATION LEVEL
Syntax

```
set transaction isolation level statement ::= SET TRANSACTION ISOLATION LEVEL isolation_level_name ;
```

ISOLATION LEVEL NAME
Syntax

```
isolation_level_name ::= READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE
```

WHENEVER
Syntax

```
whenever ::= EXEC SQL WHENEVER
             { NOT FOUND | SQLERROR | SQLWARNING }
             { STOP | CONTINUE | { GOTO | GO TO } host_lang_label } ;
```
Compliance with industry standards

Table 1 provides details on SQL DDL and DML compliance with industry standards. A check mark indicates compliance.

### Table 1: Compliance of SQL DDL and DML statements

<table>
<thead>
<tr>
<th></th>
<th>SQL</th>
<th>ODBC SQL grammar</th>
<th>Progress extension</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN-END DECLARE SECTION</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>Compliant if C language types used Embedded SQL only</td>
</tr>
<tr>
<td>CLOSE</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>DECLARE CURSOR</td>
<td>✓</td>
<td>–</td>
<td>prepared_stmt_name</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>EXEC SQL</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>EXECUTE IMMEDIATE</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>FETCH</td>
<td>✓</td>
<td>–</td>
<td>USING DESCRIPTOR</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>GET DIAGNOSTICS</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>OPEN</td>
<td>✓</td>
<td>–</td>
<td>USING DESCRIPTOR</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>PREPARE</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>Embedded SQL only</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SELECT</td>
<td>✓</td>
<td>Extended</td>
<td>FOR UPDATE</td>
<td>–</td>
</tr>
<tr>
<td>WHenever</td>
<td>✓</td>
<td>–</td>
<td>SQLWARNING STOP ACTION</td>
<td>Embedded SQL only</td>
</tr>
</tbody>
</table>
Third Party Acknowledgements

One or more products in the Progress OpenEdge v11.3 release includes third party components covered by licenses that require that the following documentation notices be provided:

Progress OpenEdge v11.3 may incorporate ANT v1.5.4. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1, applies to all versions of up to ant 1.6.0 included. The Apache Software License, Version 1.1 - Copyright (C) 2000-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Ant" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
Appendix A: Third Party Acknowledgements

ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate Apache Avalon 2002/08/06, Apache Batik v1.1, Apache Jakarta POI v1.1, Apache POI 2003/05/14, Regexp (Pure Java Regular Expression) v1.2, and XMLTask v1.15.1 from The Apache Software Foundation. Such technologies are subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 2000 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Portions of this software are based upon public domain software originally written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

Progress OpenEdge v11.3 may incorporate Apache JAXP v1.3.04 from The Apache Software Foundation. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 1999-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments
normally appear. 4. The names "Xalan" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, Lotus Development Corporation., http://www.lotus.com. For more information on the Apache Software Foundation, please see http://www.apache.org/.

Progress OpenEdge v11.3 may incorporate Crimson v1.1.3 from Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 1999-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.
Progress OpenEdge v11.3 may incorporate jfor v0.7.2 from jfor. Such technology is subject to the following terms and conditions: jfor Apache-Style Software License. Copyright (c) 2002 by the jfor project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the jfor project (http://www.jfor.org)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The name "jfor" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact info@jfor.org. 5. Products derived from this software may not be called "jfor", nor may "jfor" appear in their name, without prior written permission of info@jfor.org. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JFOR PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Apache LogKit v1.2 from The Apache Software Foundation. Such technologies are subject to the following terms and conditions: The Apache Software License, Version 1.1 -- Copyright (C) 1997-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "LogKit", "Jakarta" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate Xerces for Java XML Parser v2.6.2. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 1999 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate SOAP v2.3.1 from Apache Foundation. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1

Copyright (c) 1999 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
Appendix A: Third Party Acknowledgements

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "SOAP" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED `AS IS' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate ANTLR (Another Tool for Language Recognition) v2.7.6. Such technology is subject to the following terms and conditions: ANTLR 3 License [The BSD License] Copyright (c) 2003-2006, Terence Parr All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Castor v0.9.3 from castor.org. Such technology is subject to the following terms and conditions: Original Intalio license -
Copyright 1999-2004 (C) Intalio Inc., and others. All Rights Reserved. Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The name "ExoLab" must not be used to endorse or promote products derived from this Software without prior written permission of Intalio Inc. For written permission, please contact info@exolab.org. 4. Products derived from this Software may not be called "Castor" nor may "Castor" appear in their names without prior written permission of Intalio Inc. Exolab, Castor and Intalio are trademarks of Intalio Inc. 5. Due credit should be given to the ExoLab Project (http://www.exolab.org/). THIS SOFTWARE IS PROVIDED BY INTALIO AND CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTALIO OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Decimal Conversion Code (dtoa.c; g_fmt.c; md_prod.s; decstrod.c; decsrtodf.c; dmisc.c; gdtoa.c; gdtoa.h; gdtaiamp.h; gethex.c; gmisc.c; hd_init.c; misc.c; smisc.c; strtodg.c; strtod.c; sum.c; ulp.c). Such technologies are subject to the following terms and conditions: dtoa.c License: The author of this software is David M. Gay. Copyright (c) 1991, 2000, 2001 by Lucent Technologies. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. g_fmt.c License: The author of this software is David M. Gay. Copyright (c) 1991, 1996 by Lucent Technologies. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

md_prod.s License: The author of this software is David M. Gay. Copyright (c) 1991 by Lucent Technologies. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. decstrtod.c License: The author of this software is David M. Gay. Copyright (C) 1998-2001 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. decstrtof.c License: The author of this software is David M. Gay. Copyright (C) 1998, 2000 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. dmisc.c License: The author of this software is David M. Gay. Copyright (C) 1998 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. gdtoa.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

hd_init.c License: The author of this software is David M. Gay. Copyright (C) 2000 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

misc.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

smisc.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

strtodg.c License: The author of this software is David M. Gay. Copyright (C) 1998-2001 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

strtord.c License: The author of this software is David M. Gay. Copyright (C) 1998, 2000 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

sum.c License: The author of this software is David M. Gay. Copyright (C) 1998 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

ulp.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Progress OpenEdge v11.3 may incorporate DOM4J v1.6.1 from MetaStuff, Ltd. Such technology is subject to the following terms and conditions: Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved. Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The name "DOM4J" must not be used to endorse or promote products derived from this Software without prior written permission of MetaStuff, Ltd. For written permission, please contact dom4j-info@metastuff.com. 4. Products derived from this Software may not be called "DOM4J" nor may "DOM4J" appear in their names without prior written permission of MetaStuff, Ltd. DOM4J is a registered trademark of MetaStuff, Ltd. 5. Due credit should be given to the DOM4J Project - http://www.dom4j.org THIS SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate GraphicsMagick v1.3.14 from GraphicsMagick Group. Such technology is subject to the following terms and conditions:

.. This text is in reStucturedText format, so it may look a bit odd. .. See http://docutils.sourceforge.net/rst.html for details.

=================================================================

GraphicsMagick Copyrights and Licenses

=================================================================

This file is part of the GraphicsMagick software distributed by the GraphicsMagick Group.

[*Please note that the legal community considers 15 or more total lines of code or text (not necessarily contiguous) to

be significant for the purposes of copyright. Repeated changes such as renaming a symbol has similar significance

to changing one line of code.]*

The licenses which components of this software fall under are as follows.

1) In November 2002, the GraphicsMagick Group created GraphicsMagick from ImageMagick Studio's ImageMagick and applied the "MIT" style license: Copyright (C) 2002 - 2012 GraphicsMagick Group
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2) In October 1999, ImageMagick Studio assumed the responsibility for the development of ImageMagick (forking from the distribution by E. I. du Pont de Nemours and Company) and applied a new license:

Copyright (C) 2002 ImageMagick Studio, a non-profit organization dedicated to making software imaging solutions freely available.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (“ImageMagick”), to deal in ImageMagick without restriction, including without limitation

the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall ImageMagick Studio be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the ImageMagick Studio shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the ImageMagick Studio.

3) From 1991 to October 1999 (through ImageMagick 4.2.9), ImageMagick was developed and distributed by E. I. du Pont de Nemours and Company: Copyright 1999 E. I. du Pont de Nemours and Company

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (“ImageMagick”), to deal in ImageMagick without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.
The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.

In no event shall E. I. du Pont de Nemours and Company be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the E. I. du Pont de Nemours and Company shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the E. I. du Pont de Nemours and Company.

4) The GraphicsMagick Base64Decode() and Base64Encode() functions are based on source code obtained from OpenSSH. This source code is distributed under the following license: Copyright (c) 2000 Markus Friedl. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5) Many of the pattern images in coders/logo.c are derived from XFig, which is distributed under the following license:

| FIG : Facility for Interactive Generation of figures |
| Copyright (c) 1985-1988 by Supoj Sutanthavibul |
| Parts Copyright (c) 1989-2000 by Brian V. Smith |
| Parts Copyright (c) 1991 by Paul King |

Any party obtaining a copy of these files is granted, free of charge, a full and unrestricted irrevocable, world-wide, paid up, royalty-free, nonexclusive right and license to deal in this software and documentation files (the "Software"), including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons who receive copies from any such party to do so, with the only requirement being that this copyright notice remain intact.
6) The documentation for the composition operators is copied from the rlecomp manual page, which is authored by Rod Bogart and John W. Peterson. Rlecomp is part of the Utah Raster Toolkit distributed by the University of Michigan and the University of Utah. The copyright for this manual page is as follows: Copyright (c) 1986, University of Utah

This software is copyrighted as noted below. It may be freely copied, modified, and redistributed, provided that the copyright notice is preserved on all copies.

There is no warranty or other guarantee of fitness for this software, it is provided solely "as is". Bug reports or fixes may be sent to the author, who may or may not act on them as he desires.

You may not include this software in a program or other software product without supplying the source, or without informing the end-user that the source is available for no extra charge.

If you modify this software, you should include a notice giving the name of the person performing the modification, the date of modification, and the reason for such modification.

7) The source code comprising swab.c is originally derived from libtiff which has the following license:

| Copyright (c) 1988-1997 Sam Leffler |
| Copyright (c) 1991-1997 Silicon Graphics, Inc. |

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

8) The C++ API known as "Magick++", and which resides in the Magick++ directory, is distributed under the following license: Copyright 1999 - 2003 Bob Friesenhahn <bfriesen@simple.dallas.tx.us>

Permission is hereby granted, free of charge, to any person obtaining a copy of the source files and associated documentation files ("Magick++"), to deal in Magick++ without restriction, including without limitation of the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of Magick++, and to permit persons to whom the Magick++ is furnished to do so, subject to the following conditions:

This copyright notice shall be included in all copies or substantial portions of Magick++. The copyright to Magick++ is retained by its author and shall not be subsumed or replaced by any other copyright.
The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall Bob Friesenhahn be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with Magick++ or the use or other dealings in Magick++.

9) The GraphicsMagick HaldClutImagePixels() function in magick/hclut.c is based on source code from the HaldCLUT package by Eskil Steenberg (http://www.quelsolaar.com/technology/clut.html) which is distributed under the following license: Copyright (c) 2005 Eskil Steenberg. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR \"AS IS\" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10) GraphicsMagick makes use of third-party "delegate" libraries to support certain optional features. These libraries bear their own copyrights and licenses, which may be more or less restrictive than the GraphicsMagick license. For convenience, when GraphicsMagick is bundled with (or compiled with) "delegate" libraries, a copy of the licenses for these libraries is provided in a "licenses" directory.

-------------------------------------------------------------------------

... |copy|   unicode:: U+000A9 .. COPYRIGHT SIGN

Copyright |copy| GraphicsMagick Group 2002 - 2011

OVERVIEW and LEGAL ISSUES from jpeg.txt file (from GraphicsMagick):

The Independent JPEG Group's JPEG software

=========================================================================

README for release 6b of 27-Mar-1998

========================================================================
This distribution contains the sixth public release of the Independent JPEG Group’s free JPEG software. You are welcome to redistribute this software and to use it for any purpose, subject to the conditions under LEGAL ISSUES, below.

Serious users of this software (particularly those incorporating it into larger programs) should contact IJG at jpeg-info@uunet.uu.net to be added to our electronic mailing list. Mailing list members are notified of updates and have a chance to participate in technical discussions, etc.

This software is the work of Tom Lane, Philip Gladstone, Jim Boucher, Lee Crocker, Julian Minguillon, Luis Ortiz, George Phillips, Davide Rossi, Guido Vollbeding, Ge' Weijers, and other members of the Independent JPEG Group. IJG is not affiliated with the official ISO JPEG standards committee.

OVERVIEW
========

This package contains C software to implement JPEG image compression and decompression. JPEG (pronounced "jay-peg") is a standardized compression method for full-color and gray-scale images. JPEG is intended for compressing "real-world" scenes; line drawings, cartoons and other non-realistic images are not its strong suit. JPEG is lossy, meaning that the output image is not exactly identical to the input image. Hence you must not use JPEG if you have to have identical output bits. However, on typical photographic images, very good compression levels can be obtained with no visible change, and remarkably high compression levels are possible if you can tolerate a low-quality image. For more details, see the references, or just experiment with various compression settings. This software implements JPEG baseline, extended-sequential, and progressive compression processes. Provision is made for supporting all variants of these processes, although some uncommon parameter settings aren't implemented yet.

For legal reasons, we are not distributing code for the arithmetic-coding variants of JPEG; see LEGAL ISSUES. We have made no provision for supporting the hierarchical or lossless processes defined in the standard.

We provide a set of library routines for reading and writing JPEG image files, plus two sample applications "cjpeg" and "djpeg", which use the library to perform conversion between JPEG and some other popular image file formats. The library is intended to be reused in other applications.

In order to support file conversion and viewing software, we have included considerable functionality beyond the bare JPEG coding/decoding capability; for example, the color quantization modules are not strictly part of JPEG decoding, but they are essential for output to colormapped file formats or colormapped displays. These extra functions can be compiled out of the library if not required for a particular application. We have also included "jpegtran", a utility for lossless transcoding between different JPEG processes, and "rdjpgcom" and "wrjpgcom", two simple applications for inserting and extracting textual comments in JFIF files.

The emphasis in designing this software has been on achieving portability and flexibility, while also making it fast enough to be useful. In particular, the software is not intended to be read as a tutorial on JPEG. (See the REFERENCES section for introductory material.) Rather, it is intended to be reliable, portable, industrial-strength code. We do not claim to have achieved that goal in every aspect of the software, but we strive for it.
We welcome the use of this software as a component of commercial products.

No royalty is required, but we do ask for an acknowledgement in product documentation, as described under LEGAL ISSUES.

LEGAL ISSUES

============

In plain English:

1. We don’t promise that this software works. (But if you find any bugs, please let us know!)

2. You can use this software for whatever you want. You don’t have to pay us.

3. You may not pretend that you wrote this software. If you use it in a program, you must acknowledge somewhere in your documentation that you’ve used the IJG code.

In legalese:

The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy.

This software is copyright (C) 1991-1998, Thomas G. Lane.

All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions:

(1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation.

(2) If only executable code is distributed, then the accompanying documentation must state that “this software is based in part on the work of the Independent JPEG Group”.

(3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us.

Permission is NOT granted for the use of any IJG author’s name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group’s software".

We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

ansi2knr.c is included in this distribution by permission of L. Peter Deutsch, sole proprietor of its copyright holder, Aladdin Enterprises of Menlo Park, CA.
ansi2krr.c is NOT covered by the above copyright and conditions, but instead by the usual distribution terms of the Free Software Foundation; principally, that you must include source code if you redistribute it. (See the file ansi2krr.c for full details.) However, since ansi2krr.c is not needed as part of any program generated from the IJG code, this does not limit you more than the foregoing paragraphs do.

The Unix configuration script "configure" was produced with GNU Autoconf.

It is copyright by the Free Software Foundation but is freely distributable.

The same holds for its supporting scripts (config.guess, config.sub, ltconfig, ltmain.sh). Another support script, install-sh, is copyright by M.I.T. but is also freely distributable.

It appears that the arithmetic coding option of the JPEG spec is covered by patents owned by IBM, AT&T, and Mitsubishi. Hence arithmetic coding cannot legally be used without obtaining one or more licenses. For this reason, support for arithmetic coding has been removed from the free JPEG software. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.)

So far as we are aware, there are no patent restrictions on the remaining code.

The IJG distribution formerly included code to read and write GIF files.

To avoid entanglement with the Unisys LZW patent, GIF reading support has been removed altogether, and the GIF writer has been simplified to produce "uncompressed GIFs". This technique does not use the LZW algorithm; the resulting GIF files are larger than usual, but are readable by all standard GIF decoders.

We are required to state that "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

Contents of png.txt file (from GraphicsMagick):

This copy of the libpng notices is provided for your convenience. In case of any discrepancy between this copy and the notices in the file png.h that is included in the libpng distribution, the latter shall prevail.

COPYRIGHT NOTICE, DISCLAIMER, and LICENSE:

If you modify libpng you may insert additional notices immediately following this sentence.

libpng versions 1.0.7, July 1, 2000, through 1.2.0, September 1, 2001, are Copyright (c) 2000 Glenn Randers-Pehrson and are distributed according to the same disclaimer and license as libpng-1.0.6 with the following individuals added to the list of Contributing Authors

Simon-Pierre Cadieux
Eric S. Raymond
Gilles Vollant

and with the following additions to the disclaimer:

There is no warranty against interference with your enjoyment of the library or against infringement. There is no warranty that our efforts or the library will fulfill any of your
particular purposes or needs. This library is provided with all faults, and the entire risk of satisfactory quality, performance, accuracy, and effort is with the user.

libpng versions 0.97, January 1998, through 1.0.6, March 20, 2000, are Copyright (c) 1998, 1999 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-0.96, with the following individuals added to the list of Contributing Authors:

   Tom Lane
   Glenn Randers-Pehrson
   Willem van Schaik

libpng versions 0.89, June 1996, through 0.96, May 1997, are Copyright (c) 1996, 1997 Andreas Dilger

Distributed according to the same disclaimer and license as libpng-0.88, with the following individuals added to the list of Contributing Authors:

   John Bowler
   Kevin Bracey
   Sam Bushell
   Magnus Holmgren
   Greg Roelofs
   Tom Tanner

libpng versions 0.5, May 1995, through 0.88, January 1996, are Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc. For the purposes of this copyright and license, "Contributing Authors" is defined as the following set of individuals:

   Andreas Dilger
   Dave Martindale
   Guy Eric Schalnat
   Paul Schmidt
   Tim Wegner

The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage. Permission is hereby granted to use, copy, modify, and distribute this source code, or portions hereof, for any purpose, without fee, subject to the following restrictions:

1. The origin of this source code must not be misrepresented.

2. Altered versions must be plainly marked as such and must not be misrepresented as being the original source.
3. This Copyright notice may not be removed or altered from any source or altered source distribution.

The Contributing Authors and Group 42, Inc. specifically permit, without fee, and encourage the use of this source code as a component to supporting the PNG file format in commercial products. If you use this source code in a product, acknowledgment is not required but would be appreciated.

A "png_get_copyright" function is available, for convenient use in "about" boxes and the like:

```
printf("%s",png_get_copyright(NULL));
```

Also, the PNG logo (in PNG format, of course) is supplied in the files "pngbar.png" and "pngbar.jpg (88x31) and "pngnow.png" (98x31).

Libpng is OSI Certified Open Source Software. OSI Certified Open Source is a certification mark of the Open Source Initiative.

Glenn Randers-Pehrson
randeg@alum.rpi.edu
September 1, 2001

Contents of tiff.txt file (from GraphicsMagick):

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific prior written permission of Sam Leffler and Silicon Graphics. THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,

EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Contents of zlib.txt file (from GraphicsMagick):

zlib 1.1.3 is a general purpose data compression library. All the code is thread safe. The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files ftp://ds.internic.net/rfc/rfc1950.txt (zlib format), rfc1951.txt (deflate format) and rfc1952.txt (gzip format). These documents are also available in other formats from ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html

All functions of the compression library are documented in the file zlib.h (volunteer to write man pages welcome, contact jloup@gzip.org). A usage example of the library is
given in the file example.c which also tests that the library is working correctly. Another example is given in the file minigzip.c. The compression library itself is composed of all source files except example.c and minigzip.c.

To compile all files and run the test program, follow the instructions given at the top of Makefile. In short "make test; make install" should work for most machines. For Unix: "configure; make test; make install"

For MSDOS, use one of the special makefiles such as Makefile.msc.

For VMS, use Make_vms.com or descrip.mms.

Questions about zlib should be sent to <zlib@quest.jpl.nasa.gov>, or to Gilles Vollant <info@winimage.com> for the Windows DLL version.

The zlib home page is http://www.cdrom.com/pub/infozip/zlib/

The official zlib ftp site is ftp://ftp.cdrom.com/pub/infozip/zlib/

Before reporting a problem, please check those sites to verify that you have the latest version of zlib; otherwise get the latest version and check whether the problem still exists or not.

Mark Nelson <markn@tiny.com> wrote an article about zlib for the Jan. 1997 issue of Dr. Dobb's Journal; a copy of the article is available in http://web2.airmail.net/markn/articles/zlibtool/zlibtool.htm

The changes made in version 1.1.3 are documented in the file ChangeLog.

The main changes since 1.1.2 are:
- fix "an inflate input buffer bug that shows up on rare but persistent occasions" (Mark)
- fix gzread and gztell for concatenated .gz files (Didier Le Botlan)
- fix gzseek(..., SEEK_SET) in write mode
- fix crc check after a gzeek (Frank Faubert)
- fix miniunzip when the last entry in a zip file is itself a zip file (J Lilge)
- add contrib/asm586 and contrib/asm686 (Brian Raiter)
  See http://www.muppetlabs.com/~breadbox/software/assembly.html
- add support for Delphi 3 in contrib/delphi (Bob Dellaca)
- add support for C++Builder 3 and Delphi 3 in contrib/delphi2 (Davide Moretti)
- do not exit prematurely in untgz if 0 at start of block (Magnus Holmgren)
- use macro EXTERN instead of extern to support DLL for BeOS (Sander Stoks)
- added a FAQ file

plus many changes for portability.
Unsupported third party contributions are provided in directory "contrib". A Java implementation of zlib is available in the Java Development Kit 1.1
http://www.javasoft.com/products/JDK/1.1/docs/api/Package-java.util.zip.html

See the zlib home page http://www.cdrom.com/pub/infozip/zlib/ for details.

A Perl interface to zlib written by Paul Marquess <pmarquess@bfsec.bt.co.uk> is in the CPAN (Comprehensive Perl Archive Network) sites, such as:

A Python interface to zlib written by A.M. Kuchling <amk@magnet.com> is available in Python 1.5 and later versions, see
http://www.python.org/doc/lib/module-zlib.html

A zlib binding for TCL written by Andreas Kupries <a.kupries@westend.com> is available at http://www.westend.com/~kupries/doc/trf/man/man.html

An experimental package to read and write files in .zip format, written on top of zlib by Gilles Vollant <info@winimage.com>, is available at http://www.winimage.com/zLibDll/unzip.html and also in the contrib/minizip directory of zlib.

Notes for some targets:

- To build a Windows DLL version, include in a DLL project zlib.def, zlib.rc and all .c files except example.c and minigzip.c; compile with -DZLIB_DLL

  The zlib DLL support was initially done by Alessandro Iacopetti and is now maintained by Gilles Vollant <info@winimage.com>. Check the zlib DLL home page at http://www.winimage.com/zLibDll

  From Visual Basic, you can call the DLL functions which do not take a structure as argument: compress, uncompress and all gz* functions.

  See contrib/visual-basic.txt for more information, or get http://www.tcfb.com/dowseware/cmp-z-it.zip

- For 64-bit Irix, deflate.c must be compiled without any optimization. With -O, one libpng test fails. The test works in 32 bit mode (with the -n32 compiler flag). The compiler bug has been reported to SGI.

- zlib doesn't work with gcc 2.6.3 on a DEC 3000/300LX under OSF/1 2.1 it works when compiled with cc.

- on Digital Unix 4.0D (formerly OSF/1) on AlphaServer, the cc option -std1 is necessary to get gzprintf working correctly. This is done by configure.

- zlib doesn't work on HP-UX 9.05 with some versions of /bin/cc. It works with other compilers. Use "make test" to check your compiler.

- gzdopen is not supported on RISCOS, BEOS and by some Mac compilers.

- For Turbo C the small model is supported only with reduced performance to avoid any far allocation; it was tested with -DMAX_WBITS=11 -DMAX_MEM_LEVEL=3

- For PalmOs, see http://www.cs.uit.no/~perm/PASTA/pilot/software.html
Per Harald Myrvang <perm@stud.cs.uit.no>
The deflate format used by zlib was defined by Phil Katz. The deflate and zlib specifications were written by L. Peter Deutsch. Thanks to all the people who reported problems and suggested various improvements in zlib; they are too numerous to cite here.

Copyright notice:

(C) 1995-1998 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu

If you use the zlib library in a product, we would appreciate *not* receiving lengthy legal documents to sign. The sources are provided for free but without warranty of any kind. The library has been entirely written by Jean-loup Gailly and Mark Adler; it does not include third-party code.

If you redistribute modified sources, we would appreciate that you include in the file ChangeLog history information documenting your changes.

Progress OpenEdge v11.3 may incorporate International Classes for Unicode (International Components for Unicode) v2.4 from IBM. Such technology is subject to the following terms and conditions: ICU License - The ICU project is licensed under the X License (see also the x.org original), which is compatible with GPL but non-copyleft. The license allows ICU to be incorporated into a wide variety of software projects using the GPL license. The X license is compatible with the GPL, while also allowing ICU to be incorporated into non-open source products. License ICU License - ICU 1.8.1 and later COPYRIGHT AND PERMISSION NOTICE Copyright (c) 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR
ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name
of a copyright holder shall not be used in advertising or otherwise to promote the sale,
use or other dealings in this Software without prior written authorization of the copyright
holder.

-------------------------------------------------------------------------

All trademarks and registered trademarks mentioned herein are the property of their
respective owners.

Progress OpenEdge v11.3 may incorporate International Components for Unicode
v4.8.0. Such technology is subject to the following terms and conditions: ICU License
- ICU 1.8.1 and later COPYRIGHT AND PERMISSION NOTICE Copyright (c)
1995-2011 International Business Machines Corporation and others All rights
reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this
software and associated documentation files (the "Software"), to deal in the Software
without restriction, including without limitation the rights to use, copy, modify, merge,
publish, distribute, and/or sell copies of the Software, and to permit persons to whom
the Software is furnished to do so, provided that the above copyright notice(s) and this
permission notice appear in all copies of the Software and that both the above
copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE
COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR
ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR
ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in
advertising or otherwise to promote the sale, use or other dealings in this Software
without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their
respective owners.

Progress OpenEdge v11.3 may incorporate java.net args4j v2.0.12 from Kohsuke
Kawaguchi. Such technology is subject to the following terms and conditions:
Copyright (c) 2003, Kohsuke Kawaguchi All rights reserved. Redistribution and use in
source and binary forms, with or without modification, are permitted provided that the
following conditions are met: * Redistributions of source code must retain the above
copyright notice, this list of conditions and the following disclaimer. * Redistributions in
binary form must reproduce the above copyright notice, this list of conditions and the
following disclaimer in the documentation and/or other materials provided with the
distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Jaxen v1.1-beta-3 from jaxen.org. Such technology is subject to the following terms and conditions: Project License $Id: LICENSE.txt 1128 2006-02-05 21:49:04Z elharo $ Copyright 2003-2006 The Werken Company. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the Jaxen Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Jing 20030619 portion of Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: Jing Copying Conditions. Copyright (c) 2001-2003 Thai Open Source Software Center Ltd. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the Thai Open Source Software Center Ltd nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE. Third-party JARs - This distribution includes some additional JAR files, which have their own copying conditions:

saxon.jar Comes from the Saxon 6.5.2 distribution and is covered by these conditions.
xercesImpl.jar, xml-apis.jar Come from the Xerces-J 2.4.0 distribution and are covered by the Apache Software License.

isorelax.jar Comes from the ISO RELAX 2003/01/08 distribution and is covered by the following license: Copyright (c) 2001-2002, SourceForge ISO-RELAX Project (ASAMI Tomoharu, Daisuke Okajima, Kohsuke Kawaguchi, and MURATA Makoto) Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Progress OpenEdge v11.3 may incorporate Sun's Jmaki framework v1.0 and Sun's Jmaki framework v1.0.3 from Sun Microsystems, Inc. Such technology is subject to the following terms and conditions: Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

Progress OpenEdge v11.3 may incorporate Jmaki framework v1.8.0 from Sun Microsystems, Inc. Such technology is subject to the following terms and conditions: Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer. *
Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

Progress OpenEdge v11.3 may incorporate JSTL v1.0 from Sun Microsystems, Inc. Such technologies are subject to the following terms and conditions: Code sample License Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

Progress OpenEdge v11.3 may incorporate Looks v2.1.2 from JGoodies. Such technology is subject to the following terms and conditions: The BSD License for the JGoodies Looks - Copyright (c) 2001-2007 JGoodies Karsten Lentzsch. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of JGoodies Karsten Lentzsch nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate OpenSSL v0.9.8 from The OpenSSL
Project. Such technology is subject to the following terms and conditions:
OpenSSL License. Copyright (c) 1998-2006 The OpenSSL Project. All rights
reserved. Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met: 1. Redistributions of source
code must retain the above copyright notice, this list of conditions and the following
disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation and/or other
materials provided with the distribution. 3. All advertising materials mentioning features
or use of this software must display the following acknowledgment: "This product
includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit.
(http://www.openssl.org/)." 4. The names "OpenSSL Toolkit" and "OpenSSL
Project" must not be used to endorse or promote products derived from this software
without prior written permission. For written permission, please contact
openssl-core@openssl.org. 5. Products derived from this software may not be called
"OpenSSL" nor may "OpenSSL" appear in their names without prior written permission
of the OpenSSL Project. 6. Redistributions of any form whatsoever must retain the
following acknowledgment: "This product includes software developed by the
OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)." THIS
SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY
EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTicular PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL
PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenSSL includes cryptographic software written by Eric Young (eay@cryptsoft.com).
This product includes software written by Tim Hudson (tjh@cryptsoft.com). Original
SSLeay License. Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights
reserved. This package is an SSL implementation written by Eric Young
eay@cryptsoft.com). The implementation was written so as to conform with Netscapes
SSL. This library is free for commercial and non-commercial use as long as the
following conditions are aheared to. The following conditions apply to all code found in
this distribution, be it the RC4, RSA, Ihhash, DES, etc., code; not just the SSL code. The
SSL documentation included with this distribution is covered by the same copyright
terms except that the holder is Tim Hudson (tjh@cryptsoft.com). Copyright remains
Eric Young's, and as such any Copyright notices in the code are not to be removed. If
this package is used in a product, Eric Young should be given attribution as the author
of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)." The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-). 4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)." THIS SOFTWARE IS PROVIDED BY ERIC YOUNG `AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Progress OpenEdge v11.3 may incorporate Quartz Enterprise Job Scheduler v1.3.2 from James House. Such technologies are subject to the following terms and conditions: Copyright James House (c) 2001-2003 All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS `AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION). HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This product uses and includes within its distribution, software developed by the Apache Software Foundation (http://www.apache.org/).

Progress OpenEdge v11.3 may incorporate relaxngDatatype.jar portion of Progress Extensions for Eclipse v2.2. Such technology is subject to the following terms and conditions: Copyright (c) 2001, Thai Open Source Software Center Ltd, Sun Microsystems. All rights reserved. Redistribution and use in source and binary forms,
with or without modification, are permitted provided that the following conditions are met:
Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the names of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate SLF4J v1.6.2 and SLF4J Log4J v1.6.1 from slf4j.org. Such technology is subject to the following terms and conditions:
Copyright (c) 2004-2008 QOS.ch All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Progress OpenEdge v11.3 may incorporate STAX API (JSR 173) v3.1.1 from Woodstox Project. Such technology is subject to the following terms and conditions:
Copyright (c) 2004-2010, Woodstox Project (http://woodstox.codehaus.org/) All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the Woodstox XML Processor nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Trang 20030619 portion of Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: Copyright (c) 2002, 2003 Thai Open Source Software Center Ltd. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the Thai Open Source Software Center Ltd nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate xpp3-1.1.3.4.O portion of Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: Indiana University Extreme! Lab Software License Version 1.1.1 Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)."
Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Indiana Univeristy" and "Indiana Univeristy Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/.

5. Products derived from this software may not use "Indiana Univeristy" name nor may "Indiana Univeristy" appear in their name, without prior written permission of the Indiana University.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Xpp3 v1.1.3.4 from Indiana University. Such technology is subject to the following terms and conditions: Indiana University Extreme! Lab Software License Version 1.1.1 Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Indiana Univeristy" and "Indiana Univeristy Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/. 5. Products derived from this software may not use "Indiana Univeristy" name nor may "Indiana Univeristy" appear in their name, without prior written permission of the Indiana University. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Appendix A: Third Party Acknowledgements

Progress OpenEdge v11.3 may incorporate XStream v1.3.1 from XStream. Such technology is subject to the following terms and conditions: Copyright (c) 2003-2006, Joe Walnes Copyright (c) 2006-2007, XStream Committers All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of XStream nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate YAJL v0.4.0 from Lloyd Hilaiel. Such technology is subject to the following terms and conditions: Copyright 2007, Lloyd Hilaiel. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate GNU ZLIB v1.1.4 and GNU ZLIB v1.2.3 from Jean-loup Gailly & Mark Adler. Such technology is subject to the following terms and conditions: Copyright notice: (C) 1995-2002 Jean-loup Gailly and Mark Adler This software is provided 'as-is', without any express or implied warranty. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be
misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu

Progress OpenEdge v11.3 may incorporate ZLIB.NET Free v1.0.4 from ComponentAce. Such technology is subject to the following terms and conditions: Copyright (c) 2006-2007, ComponentAce http://www.componentace.com All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of ComponentAce nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate LALR Parser Generator in Java v0.10k. Such technology is subject to the following terms and conditions: CUP Parser Generator Copyright Notice, License, and Disclaimer – Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian - Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the names of the authors or their employers not be used in advertising or publicity pertaining to distribution of the software without specific written prior permission. The authors and their employers disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the authors or their employers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

Progress OpenEdge v11.3 includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. (MD5 Encryption Library v3.0 and MD5 Encryption vMD5C.C) These technologies are subject to the following terms and conditions: RSA Data Security MD5 message-digest algorithm RSA Data Security, Inc. MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function. License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest
Algorithm® in all material mentioning or referencing the derived work. RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind. These notices must be retained in any copies of any part of this documentation and/or software.
# Index

## A

<table>
<thead>
<tr>
<th>Function/Statement</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS function</td>
<td>130</td>
</tr>
<tr>
<td>ACOS function</td>
<td>131</td>
</tr>
<tr>
<td>ADD_MONTHS</td>
<td>132</td>
</tr>
<tr>
<td>ADD_MONTHS function</td>
<td>132</td>
</tr>
<tr>
<td>Aggregate functions</td>
<td>129</td>
</tr>
<tr>
<td>ALTER DATABASE SET</td>
<td>20</td>
</tr>
<tr>
<td>PRO_ENABLE_64BIT_SEQUENCES 'Y' statement</td>
<td>20</td>
</tr>
<tr>
<td>ALTER DATABASE SET</td>
<td>19</td>
</tr>
<tr>
<td>PRO_ENABLE_LARGE_KEYS 'Y' statement</td>
<td>19</td>
</tr>
<tr>
<td>ALTER SEQUENCE statement</td>
<td>22</td>
</tr>
<tr>
<td>ALTER TABLE statement</td>
<td>24</td>
</tr>
<tr>
<td>ALTER TENANT statement</td>
<td>29</td>
</tr>
<tr>
<td>ALTER USER statement</td>
<td>31</td>
</tr>
<tr>
<td>domain_name</td>
<td>31</td>
</tr>
<tr>
<td>Approximate numeric data types</td>
<td>262</td>
</tr>
<tr>
<td>ARRAY data types</td>
<td>175</td>
</tr>
<tr>
<td>PRO_ELEMENT function support</td>
<td>252</td>
</tr>
<tr>
<td>ASCII function</td>
<td>132</td>
</tr>
<tr>
<td>ASIN function</td>
<td>133</td>
</tr>
<tr>
<td>ATAN function</td>
<td>134</td>
</tr>
<tr>
<td>ATAN2 function</td>
<td>135</td>
</tr>
<tr>
<td>AUDIT INSERT statement</td>
<td>32</td>
</tr>
<tr>
<td>AUDIT SET statement</td>
<td>33</td>
</tr>
<tr>
<td>AVG function</td>
<td>136</td>
</tr>
</tbody>
</table>

## B

<table>
<thead>
<tr>
<th>Statement/Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backus Naur Form</td>
<td>281</td>
</tr>
<tr>
<td>data types</td>
<td>281</td>
</tr>
<tr>
<td>DDL syntax</td>
<td>287</td>
</tr>
<tr>
<td>DML statements</td>
<td>289</td>
</tr>
<tr>
<td>ESQL elements and statements</td>
<td>412</td>
</tr>
<tr>
<td>expressions</td>
<td>281, 283</td>
</tr>
<tr>
<td>literals</td>
<td>284</td>
</tr>
<tr>
<td>query expressions</td>
<td>285</td>
</tr>
<tr>
<td>search conditions</td>
<td>287</td>
</tr>
<tr>
<td>SQL elements and statements</td>
<td>281</td>
</tr>
<tr>
<td>Basic predicates</td>
<td>274</td>
</tr>
<tr>
<td>query expressions in</td>
<td>274</td>
</tr>
<tr>
<td>BEGIN-END DECLARE SECTION</td>
<td>388</td>
</tr>
<tr>
<td>BETWEEN predicate</td>
<td>274</td>
</tr>
<tr>
<td>Bit string data types</td>
<td>263</td>
</tr>
</tbody>
</table>

## C

<table>
<thead>
<tr>
<th>Statement/Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL statement</td>
<td>34, 38</td>
</tr>
<tr>
<td>CASE function</td>
<td>136</td>
</tr>
<tr>
<td>CAST function</td>
<td>139</td>
</tr>
<tr>
<td>CHAR function</td>
<td>140</td>
</tr>
<tr>
<td>Index</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Character data types 259</td>
<td></td>
</tr>
<tr>
<td>Character-string literals 269</td>
<td></td>
</tr>
<tr>
<td>CHR function 141</td>
<td></td>
</tr>
<tr>
<td>Clauses</td>
<td></td>
</tr>
<tr>
<td>COLUMN_LIST 92</td>
<td></td>
</tr>
<tr>
<td>FOR UPDATE 106</td>
<td></td>
</tr>
<tr>
<td>FROM 96</td>
<td></td>
</tr>
<tr>
<td>GROUP BY 99</td>
<td></td>
</tr>
<tr>
<td>HAVING 101</td>
<td></td>
</tr>
<tr>
<td>ORDER BY 101</td>
<td></td>
</tr>
<tr>
<td>WHERE 98</td>
<td></td>
</tr>
<tr>
<td>WITH 105</td>
<td></td>
</tr>
<tr>
<td>CLOSE statement 389</td>
<td></td>
</tr>
<tr>
<td>COALESCE function 142</td>
<td></td>
</tr>
<tr>
<td>Column constraints 35</td>
<td></td>
</tr>
<tr>
<td>COLUMN_LIST clause 92</td>
<td></td>
</tr>
<tr>
<td>Commit statement 37</td>
<td></td>
</tr>
<tr>
<td>CONCAT function 142</td>
<td></td>
</tr>
<tr>
<td>Concatenation operator 261</td>
<td></td>
</tr>
<tr>
<td>CONNECT statement 390</td>
<td></td>
</tr>
<tr>
<td>Conventional identifiers 254</td>
<td></td>
</tr>
<tr>
<td>CONVERT function (ODBC Compatible) 143</td>
<td></td>
</tr>
<tr>
<td>CONVERT function (Progress Extension) 144</td>
<td></td>
</tr>
<tr>
<td>COS function 145</td>
<td></td>
</tr>
<tr>
<td>COUNT function 145</td>
<td></td>
</tr>
<tr>
<td>CREATE DOMAIN Statement 39</td>
<td></td>
</tr>
<tr>
<td>Create DOMAIN statement</td>
<td></td>
</tr>
<tr>
<td>SUPER_TENANT super-tenant_name 39</td>
<td></td>
</tr>
<tr>
<td>CREATE INDEX statement 38, 39, 42</td>
<td></td>
</tr>
<tr>
<td>CREATE PROCEDURE statement 45</td>
<td></td>
</tr>
<tr>
<td>CREATE SEQUENCE statement 47</td>
<td></td>
</tr>
<tr>
<td>CREATE SUPER-TENANT statement 49</td>
<td></td>
</tr>
<tr>
<td>CREATE SYNONYM statement 49</td>
<td></td>
</tr>
<tr>
<td>CREATE TABLE statement 51</td>
<td></td>
</tr>
<tr>
<td>CREATE TENANT statement 57</td>
<td></td>
</tr>
<tr>
<td>CREATE TRIGGER statement 60</td>
<td></td>
</tr>
<tr>
<td>CREATE USER statement 64</td>
<td></td>
</tr>
<tr>
<td>CREATE VIEW statement 65</td>
<td></td>
</tr>
<tr>
<td>CURDATE function 146</td>
<td></td>
</tr>
<tr>
<td>CURRVAL function 147</td>
<td></td>
</tr>
<tr>
<td>CURTIME function 146</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>Data types</td>
<td></td>
</tr>
<tr>
<td>ABL 251</td>
<td></td>
</tr>
<tr>
<td>approximate numeric 262</td>
<td></td>
</tr>
<tr>
<td>character 259</td>
<td></td>
</tr>
<tr>
<td>date-time 263</td>
<td></td>
</tr>
<tr>
<td>definition 259</td>
<td></td>
</tr>
<tr>
<td>exact numeric 261</td>
<td></td>
</tr>
<tr>
<td>OpenEdge SQL 251</td>
<td></td>
</tr>
<tr>
<td>DATABASE function 148</td>
<td></td>
</tr>
<tr>
<td>DatabaseMetaData 340</td>
<td></td>
</tr>
<tr>
<td>Date arithmetic expressions 278</td>
<td></td>
</tr>
<tr>
<td>Date formats 256</td>
<td></td>
</tr>
<tr>
<td>Date-time data types 263</td>
<td></td>
</tr>
<tr>
<td>Date-time formats 255</td>
<td></td>
</tr>
<tr>
<td>Date-time literals 270</td>
<td></td>
</tr>
<tr>
<td>DAYNAME function 149</td>
<td></td>
</tr>
<tr>
<td>DAYOFMONTH function 149</td>
<td></td>
</tr>
<tr>
<td>DAYOFWEEK function 150</td>
<td></td>
</tr>
<tr>
<td>DAYOFYEAR function 151</td>
<td></td>
</tr>
<tr>
<td>DB_NAME function 151</td>
<td></td>
</tr>
<tr>
<td>DECLARE CURSOR statement 392</td>
<td></td>
</tr>
<tr>
<td>DECODE function 152</td>
<td></td>
</tr>
<tr>
<td>DEGREES function 153</td>
<td></td>
</tr>
<tr>
<td>Delimited identifiers 254</td>
<td></td>
</tr>
<tr>
<td>DESCRIBE BIND VARIABLES statement 394</td>
<td></td>
</tr>
<tr>
<td>DESCRIBE SELECT LIST statement 395</td>
<td></td>
</tr>
<tr>
<td>DESCRIBE statement 393</td>
<td></td>
</tr>
<tr>
<td>DhSQLException 312</td>
<td></td>
</tr>
<tr>
<td>DhSQLException.getDiagnostics 313</td>
<td></td>
</tr>
</tbody>
</table>
DhSQLResultSet 314
DhSQLResultSet.insert 314
DhSQLResultSet.makeNULL 315
DhSQLResultSet.set 316
DISCONNECT CATALOG statement 68
DISCONNECT statement 396
DROP DOMAIN statement 68
DROP INDEX statement 68
DROP PROCEDURE statement 70
DROP SYNONYM Statement 71
DROP SYNONYM statement 72
DROP TABLE statement 73
DROP TENANT statement 74
DROP TRIGGER statement 74
DROP USER statement 76
DROP VIEW statement 77

E
Embedded SQL 387
Error Codes
  messages 206
  SQLSTATE values 206
Escape clause
  in LIKE predicate 275
ESQL
  compliance with industry standards 417
  statements 387
Exact numeric data types 261
EXEC SQL delimiter 398
EXECUTE IMMEDIATE statement 400
EXECUTE statement 399
EXISTS predicate 276
EXPfunction 153
Expressions
  date arithmetic 278
  numeric arithmetic 277

F
FETCH statement 400
FLOOR function 154
FOR UPDATE clause 106
Formats
  date 256
  date-time 255
  number 255
  time 258
FROM clause 96

G
GET DIAGNOSTICS EXCEPTION statement 404
GET DIAGNOSTICS statement 402
GRANT statement 78
  WITH GRANT OPTION 80
GREATEST function 154
GROUP BY clause 99

H
HAVING clause 101
HOUR function 155

I
Identifiers
  conventional 253
  delimited 253
IFNULL function 155
IN predicate 276
  query expressions in 276
Industry standards 297
INITCAP function 156
INSERT function 156
INSERT statement 81
INSTR function 157
Internationalization
  specific elements
    LIKE predicate 276
Index

J
Java Class Reference
  close 311
  err 312
  execute 310
  fetch 311
  found 311
  getDiagnostics 312
  getValue 311
  insert 311
  makeNULL 310
  open 310
  rowCount 310
  set 311
  setParam 310
  wasNULL 311

JDBC conformance
  datatype conversion 339
  SQL-92 338
  supported data types 337

L
Language elements 253
  LAST_DAY function 158
  LCASE function 159
  LEAST function 160
  LEFT function 160
  LENGTH function 161
  LIKE predicate 275

Literals
  character-string 269
  date-time 270
  numeric 269
  time 271
  timestamp 272

Literals syntax in BNF 284

LOCATE function 162

LOCK TABLE statement 84

LOG10 function 163

LOWER function 163

LPAD function 164

LTRIM function 165

M
MAX function 165
MIN function 166
MINUTE function 166
MOD function 167
MONTH 167
MONTH function 167
MONTHNAME function 168
MONTHS_BETWEEN function 168

N
NEXT_DAY function 169

NOW function 171

NULL predicate 275

NULLIF function 171

Number formats 255

Numeric arithmetic expressions 277

Numeric literals 269, 277

NVL function 172

O

ODBC
  data types and OpenEdge 355
  date and time functions 381
  numeric functions 380
  Scalar functions 377
  SQLGetInfo 357
  string functions 377
  system functions 383

OPEN statement 407

OpenEdge SQL reserved words 201

OpenEdge SQL system limits 223

Operators
  relational
    in quantified predicates 274

ORDER BY clause 101

OUTER JOIN predicate 277
P

PI function 172
POWER function 173

Predicates
- basic 274
- BETWEEN 274
- EXISTS 276
- IN 276
- LIKE 275
- NULL 275
- OUTER JOIN 277
- quantified 274

PREFIX function 173
PREPARE statement 408
PRO_ARR_ESCAPE 175
PRO_ELEMENT function 175

Q

Quantified predicates 274
- query expressions in 274
QUARTER function 177

Query Expressions
- in basic predicates 274
- in IN predicate 276
- in quantified predicates 274

Query Expressions syntax in BNF 285

R

RADIANS function 178
RAND function 178

Relational operators 273
- in quantified predicates 274
REPEAT function 179
REPLACE function 179

Reserved words 201

REVOKE statement 86
- FROM PUBLIC 88
- FROM user_identifier 86
GRANT OPTION FOR 88
RESOURCE 86
RESTRICT | CASCADE 87

RIGHT function 180
ROLLBACK statement 89

ROUND function 181
ROWID function 182
RPAD function 183
RTRIM function 184

S

Search conditions syntax in BNF 287
SECOND function 185
SELECT statement 90

SET CATALOG statement 106
SET CONNECTION statement 410
SET PRO_CONNECT LOG statement 107, 108, 109
SET PRO_SERVER LOG statement 110

SET SCHEMA statement 113
SET TRANSACTION ISOLATION LEVEL Statement 114
SHOW CATALOGS statement 115
SHOW TENANT statement 118
SIGN function 185
SIN function 186

SQLCursor 317
SQLCursor.close 318
SQLCursor.fetch 319
SQLCursor.found 319
SQLCursorgetParam 320
SQLCursor.getValue 321
SQLCursor.makeNULL 323
SQLCursor.open 324
SQLCursor.registerOutParam 324
SQLCursor.rowCount 325
SQLCursor.getParam 326
SQLCursor.wasNULL 327
SQLGetInfo 357
SQLStatement 328
Index

SQLIStatement.execute 329
SQLIStatement.makeNULL 329
SQLIStatement.rowCount 330
SQLIStatement@param 331
SQLPStatement 332
SQLPStatement.execute 333
SQLPStatement.makeNULL 333
SQLPStatement.rowCount 334
SQLPStatement@param 335
SQLSTATE values 206
SQRT function 186
SUBSTR function 187
SUBSTRING function (ODBC Compatible) 188
SUFFIX function 190
SUM function 191
SYS_CHK_CONSTRS system table 248
SYS_CHKCOL_USAGE system table 247
SYS_KEYCOL_USAGE system table 248
SYS_REF_CONSTRS system table 249
SYS_TBL_CONSTRS system table 249
SYSCALCTABLE system table 230
SYSCOLAUTH system table 231
SYSCOLSTAT system table 231
SYSCOLSTAT system table 231
SYSCOLUMNS core system table 228
SYSCOLUMNS FULL system table 232
SYSDATATYPES system table 233
SYSDATE function 191
SYSDATESTAT system table 234
SYSTABAUTH system table 234
SYSFLOATSTAT system table 235
SYSIDXSTAT system table 235
SYSTABLES core system table 228
SYSTABLES FULL system table 241
SYSTBLSTAT system table 243
System Catalog Tables descriptions 226 overview 225
System limits 223
System tables and descriptions 226
SYSTIME function 191
SYSTIMESTAMP function 192
SYSTIMESTAMP system table 243
SYSTINYINTSTAT system table 244
SYSTRIGCOLS system table 244
SYSTRIGGER system table 245
SYSTSTZSTAT system table 245
SYSTSTZSTAT system table 246
SYSTSTZSTAT system table 246
SYSTABLES system table 247
SYSTIMESTAMP function 192
SYSTIMESTAMP system table 243
SYSTINYINTSTAT system table 244
SYSTRIGCOLS system table 244
SYSTRIGGER system table 245
SYSTSTZSTAT system table 245
SYSTSTZSTAT system table 246
SYSTSTZSTAT system table 246
SYSTABLES system table 247

T

TAN function 193
| Time formats 258 | UPDATE statement 124 |
| Time literals 271 | UPDATE STATISTICS statement 126 |
| Timestamp literals 272 | UPPER function 198 |
| TO_CHAR function 193 | USER function 198 |
| TO_DATE function 194 | WEEK function 199 |
| TO_NUMBER function 195 | WHENEVER statement 411 |
| TO_TIME function 195 | WHERE clause 98 |
| TO_TIMESTAMP function 196 | WITH clause 105 |
| TOP clause 95 | YEAR function 199 |
| TRANSLATE function 196 | |
| UCASE function 197 | |