# Table of Contents

## 1. SQL CONCEPTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 General Terms</td>
<td>1</td>
</tr>
<tr>
<td>1.2 DATA TYPES AND VALUES</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1 SQL Character Set</td>
<td>1</td>
</tr>
<tr>
<td>1.2.2 Expressions</td>
<td>2</td>
</tr>
<tr>
<td>1.3 TABLES</td>
<td>2</td>
</tr>
<tr>
<td>1.3.1 Attributes of Columns</td>
<td>2</td>
</tr>
<tr>
<td>1.3.2 Types of Table</td>
<td>2</td>
</tr>
<tr>
<td>1.3.3 Updatability of tables</td>
<td>3</td>
</tr>
<tr>
<td>1.3.4 Indexes</td>
<td>3</td>
</tr>
<tr>
<td>1.3.5 System Views</td>
<td>3</td>
</tr>
<tr>
<td>1.3.6 Integrity Constraints</td>
<td>3</td>
</tr>
<tr>
<td>1.4 DATABASE SYSTEM</td>
<td>4</td>
</tr>
<tr>
<td>1.4.1 Clients and Servers</td>
<td>4</td>
</tr>
<tr>
<td>Number and Location of Servers</td>
<td>4</td>
</tr>
<tr>
<td>Connections</td>
<td>4</td>
</tr>
<tr>
<td>1.4.2 Database Organization</td>
<td>5</td>
</tr>
<tr>
<td>Metadata and Data</td>
<td>5</td>
</tr>
<tr>
<td>Schema</td>
<td>5</td>
</tr>
<tr>
<td>Catalogs</td>
<td>5</td>
</tr>
<tr>
<td>Three-part Naming</td>
<td>5</td>
</tr>
<tr>
<td>1.5 USERS MANAGEMENT</td>
<td>6</td>
</tr>
<tr>
<td>1.5.1 RDBMS Users</td>
<td>6</td>
</tr>
<tr>
<td>1.5.2 Applications, Application Administrators, and Application Users</td>
<td>6</td>
</tr>
<tr>
<td>1.6 ACCESS CONTROL</td>
<td>7</td>
</tr>
<tr>
<td>1.6.1 Privileges</td>
<td>7</td>
</tr>
<tr>
<td>1.6.2 Authorizations</td>
<td>8</td>
</tr>
<tr>
<td>1.7 TRANSACTIONS</td>
<td>8</td>
</tr>
<tr>
<td>1.7.1 Transaction Attributes</td>
<td>9</td>
</tr>
<tr>
<td>1.7.2 Concurrent Transactions</td>
<td>9</td>
</tr>
<tr>
<td>Additional Guarantees</td>
<td>10</td>
</tr>
</tbody>
</table>

## 2. SQL COMMON ELEMENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 NOTATION</td>
<td>11</td>
</tr>
<tr>
<td>2.2 ERRORS AND WARNINGS</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1 Syntax Errors</td>
<td>11</td>
</tr>
<tr>
<td>2.3 LANGUAGE STRUCTURE</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 Separators and SQL Comments</td>
<td>11</td>
</tr>
<tr>
<td>2.3.2 Tokens</td>
<td>12</td>
</tr>
<tr>
<td>2.3.3 User-defined Names</td>
<td>12</td>
</tr>
<tr>
<td>Delimited User-defined Names</td>
<td>12</td>
</tr>
<tr>
<td>2.3.4 Types of User-defined Names</td>
<td>13</td>
</tr>
<tr>
<td>2.3.5 Format of Object Qualification</td>
<td>13</td>
</tr>
<tr>
<td>2.3.6 Default Qualification</td>
<td>14</td>
</tr>
<tr>
<td>2.3.7 Format of Column Qualification</td>
<td>14</td>
</tr>
<tr>
<td>2.4 KEYWORDS</td>
<td>14</td>
</tr>
<tr>
<td>2.5 GENERIC DATA TYPES</td>
<td>16</td>
</tr>
<tr>
<td>2.5.1 Bit String</td>
<td>17</td>
</tr>
<tr>
<td>2.5.2 Byte String</td>
<td>18</td>
</tr>
<tr>
<td>2.5.3 Character String</td>
<td>18</td>
</tr>
<tr>
<td>2.5.4 Numeric</td>
<td>18</td>
</tr>
<tr>
<td>2.5.5 Date/Time</td>
<td>19</td>
</tr>
</tbody>
</table>
# Table of Contents

3.1 DATA DEFINITION STATEMENTS ................................................................. 52
  3.1.1 General Diagnostics ............................................................................. 52
  3.1.2 ALTER TABLE ...................................................................................... 52
    ↓ FUNCTION .............................................................................................. 52
    ↓ SYNOPSIS ............................................................................................. 52
    ↓ DESCRIPTION ....................................................................................... 53
    ↓ ON DELETE Actions ............................................................................. 54
    ↓ APPLICATION USAGE ......................................................................... 55
    ↓ SECURITY .............................................................................................. 56
  3.1.3 CREATE CATALOG ................................................................................. 56
    ↓ FUNCTION .............................................................................................. 56
    ↓ SYNOPSIS ............................................................................................. 56
    ↓ DESCRIPTION ....................................................................................... 56
    ↓ SECURITY .............................................................................................. 56
  3.1.4 CREATE DATABASE ................................................................................ 57
    ↓ FUNCTION .............................................................................................. 57
    ↓ SYNOPSIS ............................................................................................. 57
    ↓ DESCRIPTION ....................................................................................... 57
    ↓ SECURITY .............................................................................................. 57
  3.1.5 CREATE INDEX ..................................................................................... 58
    ↓ FUNCTION .............................................................................................. 58
    ↓ SYNOPSIS ............................................................................................. 58
    ↓ DESCRIPTION ....................................................................................... 58
    ↓ SECURITY .............................................................................................. 58
  3.1.6 CREATE SCHEMA .................................................................................. 59
    ↓ FUNCTION .............................................................................................. 59
    ↓ SYNOPSIS ............................................................................................. 59
    ↓ DESCRIPTION ....................................................................................... 59
    SECURITY ................................................................................................. 59
  3.1.7 CREATE TABLE ...................................................................................... 60
    ↓ FUNCTION .............................................................................................. 60
    ↓ SYNOPSIS ............................................................................................. 60
    ↓ SECURITY .............................................................................................. 60
  3.1.8 CREATE VIEW ....................................................................................... 64
    ↓ FUNCTION .............................................................................................. 64
    ↓ SYNOPSIS ............................................................................................. 64
    ↓ DESCRIPTION ....................................................................................... 64
    ↓ SECURITY .............................................................................................. 64
  3.1.9 DROP CATALOG .................................................................................... 65
    ↓ FUNCTION .............................................................................................. 65
    ↓ SYNOPSIS ............................................................................................. 65
    ↓ DESCRIPTION ....................................................................................... 65
    ↓ SECURITY .............................................................................................. 66
  3.1.10 DROP INDEX ....................................................................................... 66
    ↓ FUNCTION .............................................................................................. 66
    ↓ SYNOPSIS ............................................................................................. 66
    ↓ DESCRIPTION ....................................................................................... 66
    ↓ SECURITY .............................................................................................. 66
  3.1.11 DROP SCHEMA .................................................................................... 67
    ↓ FUNCTION .............................................................................................. 67
    ↓ SYNOPSIS ............................................................................................. 67
    ↓ DESCRIPTION ....................................................................................... 67
    ↓ SECURITY .............................................................................................. 67
  3.1.12 DROP TABLE ........................................................................................ 68
    ↓ FUNCTION .............................................................................................. 68
8. RDBMS USER AND APPLICATION USER STATEMENTS ................................................................. 98

8.1 RDBMS USER STATEMENTS ................................................................................................. 98

8.1.1 AUTHENTICATE USER ........................................................................................................ 98

8.2 APPLICATION AND APPLICATION USER STATEMENTS ...................................................... 98

8.2.1 CREATE APPLICATION ....................................................................................................... 98

8.2.2 DROP APPLICATION .......................................................................................................... 99

8.2.3 ALTER APPLICATION ........................................................................................................ 99

8.2.4 ALTER SESSION SET APPLICATION .............................................................................. 100

8.2.5 CREATE APPLICATION_ADMIN ......................................................................................... 100

8.2.6 DROP APPLICATION_ADMIN ............................................................................................ 101

8.2.7 CREATE APPLICATION_USER .......................................................................................... 101

8.2.8 DROP APPLICATION_USER .............................................................................................. 101
1. SQL CONCEPTS

1.1 Introduction

This chapter defines terms and concepts of relational database systems. The remainder of the SQL Reference Guide relies on these definitions.

1.1.1 General Terms

A program means the host application program that uses SQL to access data.

Compilation means the process of converting a program to its executable form. When this manual asserts that a fact must be known at compilation time, the fact must be known before any pre-compilation begins.

A set is an unordered collection of distinct objects.

A multi-set is an unordered collection of objects that are not necessarily distinct.

A sequence is an ordered collection of objects that are not necessarily distinct.

The cardinality of a collection is the number of objects in that collection. Unless specified otherwise, any collection may be empty.

1.2 DATA TYPES AND VALUES

A value is a data item in the database. A single value is the smallest logical subdivision in the database. The physical representation of a value in a database is undefined.

A null value is distinct from all non-null values and represents a case where an actual value is unknown or not applicable. Each non-null value has a data type. A data type is a set of possible values. Page 16 defines six generic data types, bit string, character string, numeric, date/time, interval, security label. Within each generic data type, there are several named data types with specific attributes.

The null value is a valid value for all data types. It is distinct from all non-null values of that data type.

Each non-null value has a logical representation which is its display form in the syntax of SQL. Its logical representation is a literal (see page 40). The null value has no literal representation. In some SQL statements, the keyword NULL refers to a null value. Each non-null value is in the character string, date/time, interval or numeric class. Any two values can be compared unless they are in different classes.

1.2.1 SQL Character Set

TR provides a character set that contains at least the 26 uppercase letters, the 26 lowercase letters, the 10 digits specified in Appendix B, a blank character, a new-line indication, and the following symbols required by the SQL language:

, ( ) < > . : = * + - / ? _ % |
1.2.2 Expressions

The SQL language provides expressions that let database queries be based on computations (see page 47) and predicates that extract rows from tables based on whether they satisfy true/false assertions (see page 40). The host language may provide mechanisms to perform additional computations, whose results can be used to generate a query.

1.3 TABLES

A table is a collection of values that may vary over time. The table's values are arranged into rows and columns. The columns are ordered and at least one must be present. The rows are not ordered and a table can have zero rows. Thus, a table is a multi-set of rows; a row is a non-empty sequence of values.

A value is the smallest unit of data in a table that can be selected or updated. A row is the smallest unit of data than can be inserted into a table and deleted from a table. A table has a specification, which includes a specification of each of its columns. Every row of a table has the same cardinality and contains a value of every column of that table. The $i$th value in every row of a table is a value of the $i$th column of that table.

The degree of a table is its number of columns, which is the cardinality of each row. The cardinality of the table is its number of rows, which is the cardinality of each column.

1.3.1 Attributes of Columns

A column is a multi-set of values. The creator of a table specifies each column’s attributes and ordinal position within the table. A column’s attributes include its name, its data type, and whether it is constrained to contain only non-null values. Columns can also be added to a table after it is created.

Each column has a default value. This is the value to be given to the column when a program inserts a new row into the table without specifying a value for that column. A default value can be specified when the column is added to a table. If no default value is specified, the column’s default value is the null value.

The creator of a table can specify additional constraints on a table or on a column.

1.3.2 Types of Table

Each user-defined table is one of the following:

- **Base table**
  A base table is a named table defined by a CREATE TABLE statement. The specification of a base table includes its name.

- **Derived table**
  A derived table is a table derived directly or indirectly from one or more other tables by the evaluation of a query-expression. Page 47 describes the options the SQL program has for conceptually combining tables and for extracting desired information from the table(s).

Examples of derived tables include:

- **Viewed table**
  A viewed table (or a view) is a named derived table defined by a CREATE VIEW statement. The specification of a viewed table includes its name. (Derived tables other than views are not named.)
→ **Grouped table**
   When a query specification contains a GROUP BY or HAVING clause, evaluation involves arranging rows into groups so that, in each group, all values of specified columns (called the grouping columns) are equal. A grouped table is a derived table that contains such groups. In some contexts, set functions operate on the individual groups. For example, a computation can use the sum, average, or cardinality of a group.

→ **Grouped view**
   A grouped view is a viewed table derived from a grouped table.

In this document, table generally refers to viewed tables as well as base tables.

### 1.3.3 Updatability of tables

A table is either **updatable** or **read-only**. Updatability means whether it is valid to modify the table. The INSERT, UPDATE and DELETE statements are permitted for updatable tables and are not permitted for read-only tables.

All base tables are updatable. No grouped tables nor grouped views are updatable. A viewed table is updatable if it is derived from an updatable *query-expression*. The updatability of a *query-expression* is defined on page 48. It generally depends on the updatability of the constituent *query-specification(s)*, which is defined on page 45.

### 1.3.4 Indexes

An **index** can be thought of as a list of pointers to the rows of a table, ordered based on the values of one or more specified columns of the table. Existence of an index may enhance performance by obviating certain sort operations or by reducing the scanning of the table that is necessary to build a result set.

### 1.3.5 System Views

In addition to user-defined viewed tables, there are predefined viewed tables called **system views**. They are user-accessible, read-only viewed tables that contain information about the database. System views are described in more detail in the *Information Schema Guide*.

### 1.3.6 Integrity Constraints

Databases may have **integrity constraints**. They define a database's valid states by constraining the values in its base tables. The following integrity features are provided by TR:

→ **NOT NULL constraint**
   A table may prevent a column from containing null values.

→ **Unique constraint**
   A table may require that each row’s values for a column, or for a group of columns taken together, be unique. One set of unique columns may form a primary key for the table. (The indexes mentioned on page 3 can also be used to prevent two rows in a table from containing the same values in one or more columns.)

→ **Referential constraint**
   A table may require that each row's values for a column, or for a group of columns taken together (provided none are null), be present in a corresponding set of columns in some table in the database. These columns are called a **foreign key**.

© 2014 Infosystems Technology, Inc.
Check constraint
A table may limit the values that may be stored in a row.

Integrity constraints are checked at the end of the SQL statement.

A CREATE TABLE statement that creates a table with integrity constraints specifies the result of subsequent operations that would violate a constraint. For example, the subsequent statement may fail, it may have no effect, or it may make additional changes to the database to preserve the required integrity.

For more detailed information on Integrity Constraints, see the SQL-ODBC Tutorial.

1.4 Database System

1.4.1 Clients and Servers

The operation of a database system effectively involves two processors, a client and a server. The terms client and server in this document always mean SQL client and SQL server. Connection statements such as the CONNECT statement (see Connections) manage the associations between a client and one or more servers.

When a program is active, it is bound to a single client that processes the first implicit or explicit CONNECT statement. The client communicates with one or more servers, either directly or through other agents. The client processes the GET DIAGNOSTICS statement and has the primary role in processing CONNECT statements. The client may provide other functions independent of any server.

A server processes database requests from the client. Following the execution of such statements, diagnostic information is passed (in an undefined way) into the diagnostics area of the client.

Number and Location of Servers
An Open Group-compliant client lets the application connect to multiple servers. The application uses a single SQL programming paradigm to gain access to both local and remote servers.

Connections
A connection is an association between a client and a server. The following statements manage connections:

The CONNECT statement associates a client with a server.

The DISCONNECT statement ends this association.

→ The SET CONNECTION statement chooses a connection from among existing connections.

Connection Context
For the duration of a connection⁴ between a client and server, Trusted Rubix preserves at least the following context:

→ the transaction attributes described on page 9.

¹ The International Standard associates some of the context with a SQL session, which is a concept defined separately from the concept of a connection to a server. This document uses the term connection exclusively.
1.4.2 Database Organization

Metadata and Data

Each server provides a database, which consists of metadata and data.

Metadata comprises the definitions of all active base tables, viewed tables, indexes, privileges and user names. (An item of metadata is active if it has been defined and has not subsequently been dropped. User names are defined on page 12.)

Data comprises every value in every active table.

Schema

A schema is a collection of related objects. A schema may contain base tables, and contains any indexes that are defined on the base tables.

A schema has a separate namespace (that is, the name of every object in the schema must be unique) for each of the following:

- Tables
- Indexes
- Character sets
- Collations.

Catalogs

Catalogs are named collections of schema in an SQL environment. An SQL environment contains one or more catalogs and a catalog contains one or more schema. Every catalog contains a schema named INFO_SCHEMA that contains the views of the information schema, or system tables.

The creation and destruction of catalogs is implementation-defined. The set of catalogs that can be referenced in any SQL statement, during any particular transaction or during the course of an SQL session, is also implementation-defined.

Three-part Naming

An application can uniquely identify a table or index by qualifying the table or index identifier (preceding it with its schema name). In turn, a schema name may be further qualified by preceding it with a catalog name. A fully qualified table, index, or view qualifier takes the form of:

- catalog-name.schema-name.table-name
- catalog-name.schema-name.index-name
- catalog-name.schema-name.view-name

The total length of a three-part name is 150 characters, with each part equal to a maximum of 50 characters.

Certain ODBC metadata functions return result sets whose TABLE_CAT column indicates the catalog name of an object. For objects that do not have a catalog name, this column contains a zero-length string, except that if the implementation does not support catalog names, the column may instead be null.

The catalog-name and schema-name are each syntactically a user-defined-name. Other implementation-defined object naming systems may also be supported. In all cases, use of qualification is optional for the application program.
1.5 Users Management

Trusted RUBIX uses a two-tier user structure which consists of the RDBMS user and the optional application user. The RDBMS user connects directly to the database server and performs SQL based operations. An authenticated RDBMS user is required for every database session. The application user connects to a middleware application and performs operations as defined by the middleware application. The middleware application itself is executed by a RDBMS user. The purpose of the application user is to extend the mandatory access controls of Trusted RUBIX to second-tier users.

1.5.1 RDBMS Users

An authenticated RDBMS user is required for every database session. The RDBMS user must be a valid operating system (OS) user on the RDBMS server host platform. RDBMS users have an associated group which may be used for access control. Access control of the RDBMS user is managed through the Trusted RUBIX discretionary access control, multilevel security, attribute based access control, and role based access control mechanisms.

If the RDBMS client process executes on the RDBMS server host platform, authentication occurs by extracting the user and group ID’s from the client process. If the RDBMS client process executes on a remote host, then authentication occurs on the RDBMS server host platform using the operating system’s Pluggable Authentication Module (PAM).

1.5.2 Applications, Application Administrators, and Application Users

A Trusted RUBIX application corresponds to a RDBMS middleware application. An application typically services its own set of application users. On traditional database systems, these application users are unknown to the RDBMS and therefore not controlled by the security mechanisms of the RDBMS. However, Trusted RUBIX applications users are authenticated to the RDBMS and stored as part of the session’s security attributes. Application users have a string name that is unique within its associated application and an integer ID that is unique within the entire database.

An example of an application user is the typical user of an internet banking application. Application user’s access to the database is controlled using the Trusted RUBIX attribute based access control (ABAC) mechanism, the Security Policy Manager (SPM).

To configure the application user tier the following steps are followed:

1. The Database Administrator creates a named application (CREATE APPLICATION command). This application corresponds to a RDBMS middleware application.
2. The Security Administrator creates one or more application administrators (CREATE APPLICATION_ADMIN command). An application administrator is a RDBMS user that is allowed to execute the middleware application and set that application as current in the database session.
3. The application administrator executes the middleware application, connects to the database, and creates one or more application users (CREATE APPLICATION_USER command), as described next.

To service application users the following steps are followed:

1. The application administrator executes the middleware application and sets the application as being current (ALTER SESSION SET APPLICATION command).
2. The middleware application receives operations from application users.
3. The middleware application authenticates the application user to the RDBMS (AUTHENTICATE...
APPLICATION_USER command). The RDBMS sets the application user in the current database session.

4. The operations submitted by the application user to the middleware application are mapped to SQL operations and submitted to the database system.

5. The ABAC security policy of the SPM consults the current application user and ensures only operations permitted for that application user are allowed. It also ensures that rows created by the application user are labeled as being created by the application user.

The ABAC security policy of the SPM will ensure that the RDBMS user may not perform any operation restricted to an application user unless that application user is also currently authenticated. Thus, both the RDBMS user and application user must be authenticated to the database for any such operation to succeed. Both identities are usable for access controls using the SPM.

1.6 ACCESS CONTROL

A server distinguishes between different users. Each user is identified by a user name. Every program is executed on behalf of a user. The current user is the user on behalf of whom the current program is executed. The identity of the user is specified implicitly or explicitly in the CONNECT statement.

Trusted Rubix uses additional criteria to restrict database access. Specifically, in environments where such protections are appropriate, mandatory (label-based) protections can be implemented. See the Security Features User’s Guide for more detail.

1.6.1 Privileges

A privilege authorizes a user to perform a given category of action on a specified database object. All data manipulation statements report an error if the current user does not have the appropriate privilege for the operation. Different access privileges are applicable to different types of objects.

The NULL privilege is applicable to all types. It is a specific privilege that explicitly disallows access to the object for the specific user. It takes precedence over all other access controls.

The additional privileges applicable to database, catalog, and schema objects are READ, WRITE, and EXEC. These objects are similar in that they contain other objects (databases contain catalogs; catalogs contain schema; and schema contain relations and views). The READ privilege permits the subject to list the object names, WRITE privilege permits the subject to create and drop objects, and EXEC privilege permits the subject to access the contained objects.

The privileges applicable to table objects (relations and views) are:

- → SELECT(I),  → UPDATE(I),  → DELETE,
- → SELECT,    → UPDATE,    → CRVIEW,
- → INSERT(I),  → REFERENCES, → REFVIEW.
- → INSERT,    → REFERENCES(I),

The PRIVILEGE(I) form of these privileges permits the subject access PRIVILEGE to column I of a table. The PRIVILEGE form of these privileges implies PRIVILEGE(I) for all current or future columns I of that table. The SELECT, INSERT, and UPDATE privileges permit the subject to perform the corresponding operation on the table. The REFERENCES(I) privilege permits the subject to reference a specific column I of the table in an integrity constraint. The DELETE privilege permits the subject to delete tuples from the table. There is no corresponding DELETE(I) privilege because DELETE operates on entire tuples by
definition. The CRVIEW privilege permits the subject to create a view that references the table. Finally, in order for a subject to access a view, the subject must have REFVIEW privilege on all of the tables references in the view.

Each of the privileges listed above for all types of objects are also associated with an additional privilege, the GRANT privilege (WITH GRANT OPTION). In no case can any specified privilege, e.g., SELECT, be granted to another user unless that user also has the grant option for the specified privilege, i.e., SELECT.

### 1.6.2 Authorizations

Authorizations in TR can be granted to users, groups, and PUBLIC. Authorizations are either implicit or explicit. Implicit authorizations are granted when an object is created. When a database, catalog, schema, or stored relation is created, the creator is granted all applicable privileges (except for NULL) including the grant option for all privileges. When a view is created, the creator is granted the intersection of the privileges held by the creator on all the tables referenced in the view definition. No other implicit authorizations are granted upon object creation. Explicit authorizations are granted/revoked via explicit user action (i.e., by using the GRANT/REVOKE statement).

TRusted Rubix mediates all access by subjects to objects based on the subject's authorizations. TR uses the “most specific” rule to resolve conflicts between these authorizations. If a user is specifically granted or denied authorization to an object, this authorization takes precedence over any authorizations that are granted or denied to groups to which the user belongs. Similarly, group authorizations take precedence over authorizations to PUBLIC.

Since a TR subject can simultaneously possess multiple group memberships, the subjects effective group authorization is calculated by forming the union of the subject's individual group authorizations. In calculating this intersection, NULL privilege is ignored. That is, just because a subject is explicitly denied access by virtue of being a member of one group, it doesn't mean the subject cannot obtain access by virtue of being a member of another (authorized) group.

NOTE

For a detailed discussion of access control, please see the Security Features User's Guide.

### 1.7 TRANSACTIONS

A transaction is a sequence of executable SQL statements that is atomic with respect to recovery and concurrency. With one exception described below, changes that a transaction makes to a database can be perceived by that application but cannot be perceived by other applications unless and until the original transaction ends with a COMMIT statement.

A transaction starts (becomes active) when a program that does not already have an active transaction executes an SQL statement that operates on a database. The program can end the transaction by executing one of these statements:

- COMMIT enacts all changes made to the database during the transaction.
- ROLLBACK cancels all changes made to the database during the transaction.

If the program ceases execution without ending the transaction, then according to undefined criteria, either a COMMIT or ROLLBACK occurs.
1.7.1 Transaction Attributes

A transaction has the following attributes:

→ **Access Mode**
   
   A transaction’s access mode is either read-only or read-write. The **read-only** access mode applies only to tables. Even if the relevant tables and cursors are updatable, a DELETE, INSERT or UPDATE statement fails if it is within a transaction with the read-only access mode. Changes to metadata (data definition statements) are also disallowed in such transactions.

   In a transaction with the **read-write** access mode, the above restriction does not apply, but updates to metadata and data may be prevented by other restrictions such as a non-updatable cursor or table or lack of access rights.

→ **Isolation Level**
   
   The isolation level of a transaction is either READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ or SERIALizable. The isolation level specifies the degree to which the operations on metadata or data interact with the effects of concurrent transactions discussed in the next section.

In SQL, the application does not specify the attributes when starting a transaction. Instead, the `SET TRANSACTION` statement specifies the attributes to be used in the next transaction.

1.7.2 Concurrent Transactions

**Concurrent** transactions are transactions begun by different programs that gain access to the same metadata or data and that overlap in time. All concurrent transactions are by default guaranteed to be serializable; that is, any interaction between the transactions occurs only in ways that guarantee that the result is the same as some serial sequence of the same transactions. This corresponds to the SERIALizable isolation level defined in the International Standard. It is the highest isolation level.

At lower isolation levels, one or more of the following phenomena may occur during the execution of the following transactions:

→ **Phantoms**
   
   Selecting rows a second time with the same search conditions may retrieve a different set of rows. The second set may include “phantom” rows that another, concurrent transaction inserted after the first retrieval.

→ **Non-repeatable reads**
   
   Reading the same row a second time may produce a different result, if the row is updated or deleted by another concurrent transactions.

→ **Dirty reads**
   
   All transactions perceive changes to the database even before they are committed.

The various transaction isolation levels are defined below as guarantees that certain of the above phenomena are not possible:

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Read</th>
<th>Non-repeatable Read</th>
<th>Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIALizable</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
</tr>
</tbody>
</table>
When an application specifies an isolation level, **Trusted Rubix** prepares to conduct the next transaction so that the corresponding phenomena are not possible in that transaction, from the point of view of the application, based on database operations performed by other concurrent applications.

In **TR**, all four isolation levels are available to the application. However, specification of higher isolation levels may result in reduced concurrency.²

The concurrency control algorithm used in **Trusted Rubix** is a form of Multi Version Concurrency Control (MVCC) called Secure Multi-Version Timestamp Ordering (MVTO). As database rows are updated, this algorithm creates a new row version while maintaining old row versions. Transaction ID’s and timestamps are used to map operations to correct row versions. As row versions become obsolete they are efficiently garbage collected. The MVTO algorithm provides two major advantages. First, transaction conflicts between security domains (e.g., between MLS levels) are removed, thus removing any covert, illegal information flow. Secondly, read-only transactions are able to always proceed without conflicting with any read-write transaction.

**Additional Guarantees**

Regardless of the transaction isolation level, all transaction work is guaranteed to be atomic (which means that the results of the transaction’s execution are either all committed or all rolled back) and it is guaranteed that no updates are lost. In addition, it is guaranteed that none of the phenomena listed above occur (as in SERIALIZABLE isolation level) during the following operations:

→ the implied reading of metadata definitions that must be performed in the execution of an SQL statement;

→ the checking of integrity constraints; and,

→ the execution of referential actions associated with referential constraints.

**NOTE**

The **SQL-ODBC Tutorial** discusses in further detail the implications of using various concurrency schemes and how a user can specify which scheme to use.

---

² For example, a primitive but valid method of guaranteeing serializability would be to actually serialize concurrent transactions. Users and implementers may evaluate the consequences of each isolation level by performance measurement of the database.
2. SQL COMMON ELEMENTS

2.1 NOTATION

The following syntactic notation is used to define the SQL syntax, both in the SYNONYMS section and in Appendix A:

→ Words shown in uppercase, such as COUNT, are literal syntactic elements of SQL and must appear as shown.

→ Words in italics, including hyphens (for example, search-condition), are descriptive terms for syntactic elements. The term may appear in the text to refer to the element. When two elements with the same syntactic form have different semantic rules, they may have separate names such as field-name-1 and field-name-2.

→ Square brackets [ ] enclose syntax that is optional.

→ Braces { } delimit a range of syntax to which vertical bars or ellipses apply (see below).

→ Vertical bars separate mutually exclusive options.

→ An ellipsis (...) means that the syntax enclosed by the immediately preceding pair of braces or brackets may be repeated.

→ All other symbols (such as parentheses) are literal syntactic elements of SQL and must appear as shown.

2.2 ERRORS AND WARNINGS

Any SQL statement can result in unqualified success or can result in a variety of warning or error outcomes. The preferred way for an application to test the outcome of an SQL statement is to test the SQLSTATE status variable.

Anywhere that this manual contains a statement imposing a requirement on the application for proper invocation of an SQL statement, it means that Trusted Rubix checks that it was called properly. The manual also indicates the SQLSTATE value by which TR reports a violation of this rule.

2.2.1 Syntax Errors

Violation of any syntactic assertion in this manual that does not specify a SQLSTATE value is a Syntax error (’42000’). The general rule is that this SQLSTATE applies to the syntax rules in the SYNONYMS and DESCRIPTION sections of SQL statement descriptions.

2.3 LANGUAGE STRUCTURE

The SQL language combines characters from the system's character set to form tokens and separators.

2.3.1 Separators and SQL Comments

A separator is a space, a newline indication, or an SQL comment. A separator may be followed by another separator or a token. If the newline indication is a character, it cannot be used to form a token.
An SQL comment begins with two or more consecutive hyphens and runs up to the next newline indication. A comment must not exist on the same line as an SQL statement. It must exist on separate line(s).

### 2.3.2 Tokens

A token is either a non-delimiter token or a delimiter token. A non-delimiter token must be followed by a separator or, if allowed by the syntax, a delimiter token. A non-delimiter token is a user-defined name, a host identifier, a keyword or a numeric-literal. A delimiter token is a character-string-literal or one of the characters:

```
, ( ) <> . : = * + - ? and /
```

or one of the character-pairs:

```
< > >= <= and ||
```

A user-defined name or a keyword may contain lowercase letters. In these tokens, a lowercase letter is equivalent to the corresponding uppercase letter. Literals are described on page 24.

### 2.3.3 User-defined Names

A user-defined name is a character string that must be supplied by the user to satisfy the format in which that name appears. It may consist of upper- and lowercase letters, digits and underscore characters, except that the first character must be a letter. All the characters in a user-defined name are significant, except that case is not significant. User-defined names are represented in uppercase in the system views, in the diagnostics area, and in SQL descriptor areas.

The maximum length of a user-defined name is 50 characters. A user-defined name must not be one of the reserved words listed on page 14.

#### Delimited User-defined Names

User-defined names may be enclosed in double quotes (“”). These are called delimited user-defined names. The text within the double quotes is called the body of the user-defined name. To specify a double quote as a component character of the user-defined name, two consecutive double quotes appear in the body.

Delimited user-defined names are special in three respects:

- They can contain characters that the syntax of SQL would not permit or would interpret specially if they occurred without delimitation, including blanks in any or all positions.
- They can be identical to a reserved keyword.
- They are tested in a case sensitive manner, allowing the application to specify uppercase and lowercase letters.

The value of a delimited user-defined name is the body (defined above) of the name, after converting each instance of two consecutive double quotes to a single double quote. (The value is visible to applications through the system views in the Information Schema Guide. The length limit of 50 characters specified for user-defined names applies to this value.)

---

1 Consecutive hyphens inside a delimited user-defined name or character string literal do not begin an SQL statement.
2.3.4 Types of User-defined Names

The following syntactic elements are user-defined names:

- **unqualified-base-table-name**
  - The name of a base table. It can be qualified (see page 13).

- **unqualified-column-name**
  - The name of a column, which must be unique within its table. It can be qualified (see page 14).

- **correlation-name**
  - An alias for a *table-name*. It is specified by being paired with the *table-name* in a FROM clause. Its function is described on page 49. It is used in *table-references*, which are defined on page 49.

- **cursor-name**
  - The name of a cursor. It must be unique within a host program.

- **unqualified-index-name**
  - The name of an index. It can be qualified (see page 13).

- **product-name**
  - A name that identifies *Trusted Rubix*’s SQL version.

- **statement-name**
  - A name that identifies an SQL statement that is prepared for execution.

- **user-name**
  - The name of a user (not necessarily the login name). It must be unique within the names of authorized users of the database.

- **vendor-name**
  - A name that identifies the supplier of *TR*.

- **unqualified-viewed-table-name**
  - The name of a viewed table. It can be qualified.

2.3.5 Format of Object Qualification

An *unqualified-base-table-name* is the defined name of a base table. An *unqualified-viewed-table-name* is the defined name of a viewed table. An *unqualified-index-name* is the defined name of an index.

Any of these objects may be uniquely identified (qualified) by preceding its unqualified name with an object-qualifier, which specifies the schema in which it resides. An object-qualifier has the following syntax:

```
  schema-name
```

The *schema-name* is syntactically a user-defined-name. In all cases, use of the object-qualifier is an application option.

To indicate a case where the SQL syntax allows the application to qualify an identifier, this specification...
6.0 REVISION 9

introduces the following additional terms:

→ The construct `[object-qualifier.]unqualified-base-table-name` is referred to as a `base-table-name`.
→ The construct `[object-qualifier.]unqualified-viewed-table-name` is referred to as a `viewed-table-name`.
→ The construct `[object-qualifier.]unqualified-index-name` is referred to as an `index-name`.

A `table-name` is either a `base-table-name` or a `viewed-table-name`.

### 2.3.6 Default Qualification

For each connection, there is a default catalog name, and a default schema name. In any identifier that could be qualified, if there is no catalog name, then the default catalog is implicit; if there is no schema name, then the default schema is implicit.

The application can change the default catalog name by using the `SET CATALOG` statement described on page 84. The application can change the default schema name by using the `SET SCHEMA` statement described on page 85.

The initial value of the default catalog name is “default_catalog”; the default schema name is “default_schema”.

### 2.3.7 Format of Column Qualification

A `column-identifier` names a column and must be unique within the names of the columns of the associated table. A `column-identifier` may be uniquely identified by being qualified by a `table-name` or `correlation-name`.

The construct `[ { table-name | correlation-name }.] unqualified-column-name` is referred to as a `column-name`.

### 2.4 KEYWORDS

Keywords are predefined words that are required literally in some situations in SQL syntax. The syntax definitions in this specification show keywords in uppercase letters. The following keywords are unavailable for use as `user-defined names`:

<table>
<thead>
<tr>
<th>ABSOLUTE</th>
<th>DECLARE</th>
<th>GO</th>
<th>NULL</th>
<th>SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>DEFAULT</td>
<td>GOTO</td>
<td>NULLIF</td>
<td>SQL</td>
</tr>
<tr>
<td>ADD</td>
<td>DEFERRABLE</td>
<td>GRANT</td>
<td>NUMERIC</td>
<td>SQLCA</td>
</tr>
<tr>
<td>ALL</td>
<td>DEFERRED</td>
<td>GROUP</td>
<td>OCTET_LENGTH</td>
<td>SQLCODE</td>
</tr>
<tr>
<td>ALLOCATE</td>
<td>DELETE</td>
<td>HAVING</td>
<td>OF</td>
<td>SQLERROR</td>
</tr>
<tr>
<td>ALTER</td>
<td>DESC</td>
<td>HOUR</td>
<td>ON</td>
<td>SQLSTATE</td>
</tr>
<tr>
<td>AND</td>
<td>DESCRIBE</td>
<td>IDENTITY</td>
<td>ONLY</td>
<td>SQLWARNING</td>
</tr>
<tr>
<td>ANY</td>
<td>DESCRIPTOR</td>
<td>IMMEDIATE</td>
<td>OPEN</td>
<td>SUBSTRING</td>
</tr>
<tr>
<td>ARE</td>
<td>DIAGNOSTICS</td>
<td>IN</td>
<td>OPTION</td>
<td>SUM</td>
</tr>
<tr>
<td>AS</td>
<td>DISCONNECT</td>
<td>INCLUDE</td>
<td>OR</td>
<td>SYSTEM_USER</td>
</tr>
<tr>
<td>ASC</td>
<td>DISTINCT</td>
<td>INDEX</td>
<td>ORDER</td>
<td>TABLE</td>
</tr>
<tr>
<td>ASSERTION</td>
<td>DOMAIN</td>
<td>INDICATOR</td>
<td>OUTER</td>
<td>TEMPORARY</td>
</tr>
<tr>
<td>ASYNC</td>
<td>DOUBLE</td>
<td>INITIALLY</td>
<td>OUTPUT</td>
<td>TEST</td>
</tr>
<tr>
<td>AT</td>
<td>DROP</td>
<td>INNER</td>
<td>OVERLAPS</td>
<td>THEN</td>
</tr>
<tr>
<td>AUTHORIZATION</td>
<td>ELSE</td>
<td>INPUT</td>
<td>PAD</td>
<td>TIME</td>
</tr>
<tr>
<td>Keyword</td>
<td>Symbol</td>
<td>Keyword</td>
<td>Symbol</td>
<td>Keyword</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
<td>----------------------</td>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>AVG</td>
<td>END</td>
<td>):</td>
<td>PARTIAL</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>BEGIN</td>
<td>END_EXEC</td>
<td>DEFAULT</td>
<td>POSITION</td>
<td>TIMEZONE_HOUR</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>ESCAPE</td>
<td>INTERSECT</td>
<td>PREERVE</td>
<td>TIMEZONE_MINUTE</td>
</tr>
<tr>
<td>BIT</td>
<td>EXCEPT</td>
<td>INTERCEPT</td>
<td>PRESERVE</td>
<td>TO</td>
</tr>
<tr>
<td>BIT_LENGTH</td>
<td>EXCEPTION</td>
<td>INTERVAL</td>
<td>PRIMARY</td>
<td>TRANSACTION</td>
</tr>
<tr>
<td>BOTH</td>
<td>EXEC</td>
<td>INTO</td>
<td>PRIOR</td>
<td>TRANSLATE</td>
</tr>
<tr>
<td>BY</td>
<td>EXECUTE</td>
<td>IS</td>
<td>PRIVILEGES</td>
<td>TRUE</td>
</tr>
<tr>
<td>CASCADE</td>
<td>EXISTS</td>
<td>ISOLATION</td>
<td>PROCEDURE</td>
<td>TRUE</td>
</tr>
<tr>
<td>CASCADED</td>
<td>EXTERNAL</td>
<td>JOIN</td>
<td>PUBLIC</td>
<td>TRUE</td>
</tr>
<tr>
<td>CASE</td>
<td>EXTRACT</td>
<td>LANGUAGE</td>
<td>READ</td>
<td>TRUE</td>
</tr>
<tr>
<td>CAST</td>
<td>FALSE</td>
<td>LEFT</td>
<td>REPEATABLE</td>
<td>TRUE</td>
</tr>
<tr>
<td>CATALOG</td>
<td>FETCH</td>
<td>LEVEL</td>
<td>UPDATE</td>
<td>TRUE</td>
</tr>
<tr>
<td>CHAR</td>
<td>FIRST</td>
<td>LIKE</td>
<td>USER</td>
<td>TRUE</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>FLOAT</td>
<td>MAX</td>
<td>VARCHAR</td>
<td>TRUE</td>
</tr>
<tr>
<td>CHARACTER_LENGTH</td>
<td>FOR</td>
<td>MIN</td>
<td>VIEW</td>
<td>TRUE</td>
</tr>
<tr>
<td>CHAR_LENGTH</td>
<td>FOREIGN</td>
<td>MINUe</td>
<td>TRANSACTION</td>
<td>TRUE</td>
</tr>
<tr>
<td>CHECK</td>
<td>FOUND</td>
<td>MODULE</td>
<td>WHEN</td>
<td>TRUE</td>
</tr>
<tr>
<td>CLOSE</td>
<td>FROM</td>
<td>MONTH</td>
<td>WHENCevere</td>
<td>TRUE</td>
</tr>
<tr>
<td>COALESCE</td>
<td>FULL</td>
<td>NCHAR</td>
<td>WORK</td>
<td>TRUE</td>
</tr>
<tr>
<td>COLLATE</td>
<td>GET</td>
<td>NATURAL</td>
<td>WRITE</td>
<td>TRUE</td>
</tr>
<tr>
<td>COLLATION</td>
<td>GLOBAL</td>
<td>NATURAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLUMN</td>
<td>CROSS</td>
<td>SCHEMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td>CURRENT</td>
<td>SCROLL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMITTED</td>
<td>CURRENT_DATE</td>
<td>SECOND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td>CURRENT_TIME</td>
<td>SECOw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECTION</td>
<td>CURRENT_TIMESTAMP</td>
<td>SESSION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSTRAINT</td>
<td>CURRENT_USER</td>
<td>SERIALIZEABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td>CURSOR</td>
<td>SESSION_USER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>DATE</td>
<td>SESSION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONVERT</td>
<td>DAY</td>
<td>SET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORRESPONDING</td>
<td>DEALLOCATE</td>
<td>SIZE</td>
<td>tal</td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td>DEC</td>
<td>SMALLINT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE</td>
<td>DECIMAL</td>
<td>SOME</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following keywords are defined in Trusted Rubix SQL but are not reserved words. Applications may use them as user-defined names:

ADA
CATALOG_NAME
CC
CHARACTER_SET_CATALOG
CHARACTER_SET_NAME
CHARACTER_SET_SCHEMA
CLASS_ORIGIN
COBOL
COLLATION_NAME
COLLATION_SCHEMA
COLUMN_NAME
CONDITION_NUMBER
CONNECTION_NAME
CONSTRAINT_CATALOG
CONSTRAINT_NAME
CONSTRAINT_SCHEMA

CURSOR_NAME
DATA
DATETIME_INTERVAL_CODE
DATETIME_INTERVAL_PRECISION
DYNAMIC_FUNCTION
ENVIRONMENT_NAME
FORTRAN
HIGH
LENGTH
LOW
MESSAGE_LENGTH
MESSAGE_OCTET_LENGTH
MESSAGE_TEXT
MORE
MUMPS

NAME
NULLABLE
PASCAL
PLI
RETURNED_LENGTH
RETURNED_OCTET_LENGTH
RETURNED_SQLSTATE
ROW_COUNT
SCALE
SCHEMA_NAME
SERVER_NAME
SUBCLASS_ORIGIN
TABLE_NAME
TYPE
UNNAMED

© 2014 Infosystems Technology, Inc.
The following names are reserved words for future use:

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFTER</td>
<td>LEAVE</td>
<td>PENDANT</td>
</tr>
<tr>
<td>ALIAS</td>
<td>LESS</td>
<td>PREORDER</td>
</tr>
<tr>
<td>BEFORE</td>
<td>LIMIT</td>
<td>PRIVATE</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>LOOP</td>
<td>PROTECTED</td>
</tr>
<tr>
<td>BREADTH</td>
<td>MODIFY</td>
<td>RECURSIVE</td>
</tr>
<tr>
<td>CALL</td>
<td>NEW</td>
<td>REF</td>
</tr>
<tr>
<td>COMPLETION</td>
<td>NO</td>
<td>REFERENCING</td>
</tr>
<tr>
<td>CYCLE</td>
<td>NONE</td>
<td>REPLACE</td>
</tr>
<tr>
<td>DEPTH</td>
<td>OBJECT</td>
<td>RESIGNAL</td>
</tr>
<tr>
<td>DICTIONARY</td>
<td>OFF</td>
<td>RETURN</td>
</tr>
<tr>
<td>EACH</td>
<td>OID</td>
<td>RETURNS</td>
</tr>
<tr>
<td>ELSEIF</td>
<td>OLD</td>
<td>ROLE</td>
</tr>
<tr>
<td>EQUALS</td>
<td>OPERATION</td>
<td>ROUTINE</td>
</tr>
<tr>
<td>GENERAL</td>
<td>OPERATORS</td>
<td>ROW</td>
</tr>
<tr>
<td>IF</td>
<td>OTHERS</td>
<td>SEARCH</td>
</tr>
<tr>
<td>IGNORE</td>
<td>PARAMETERS</td>
<td>SENSITIVE</td>
</tr>
</tbody>
</table>

The following are **Trusted Rubix** reserved words:

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>INDEX</td>
<td>POLYINSTANTIATION</td>
</tr>
<tr>
<td>CLOB</td>
<td>LABEL</td>
<td>ROWID</td>
</tr>
<tr>
<td>BLOB</td>
<td>LONG</td>
<td>ROWIN</td>
</tr>
<tr>
<td>DATABASE</td>
<td>MICRO</td>
<td>ROWLABEL</td>
</tr>
<tr>
<td>HISTORICAL</td>
<td>NONE</td>
<td>ROWOUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROWREADLABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROWREADTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STOL</td>
</tr>
</tbody>
</table>

## 2.5 GENERIC DATA TYPES

A data type is a set of representable values. A data type restricts the contents or representation of a value.

There are six generic data types; bit string, character string, date/time, interval, numeric and security label. Non-null values within a generic data type can be assigned to one another and compared to one another. Generally, values cannot be assigned to or compared with values of another generic data type. (The null value is treated specially in comparisons and assignment; see page 30.) However, the application may explicitly convert values from one generic data type to another using the `CAST` function (see page 38).

Within each generic data type, there are a variety of named data types. A named data type has specific characteristics and SQL syntax. The application identifies a named data type using an SQL keyword and, for some named data types, may specify additional attributes such as length and precision. Named data types are classified as shown below, and are defined in the following sections. **TR** SQL supports the named data types listed in the following table:

<table>
<thead>
<tr>
<th>Generic Data Type</th>
<th>Subclassification</th>
<th>Named Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit/Byte String</td>
<td>Fixed Length</td>
<td>BIT</td>
</tr>
<tr>
<td>(see page 17)</td>
<td>Variable Length</td>
<td>BIT VARYING</td>
</tr>
</tbody>
</table>
<pre><code>                                      | --------------------------------|
</code></pre>
2.5.1 Bit String

A bit string is a sequence of bits, each having the value of 0 or 1. A bit string has a length, which is the number of bits in the string. The length is 0 or a positive integer.

The data type bit string is specified by BIT. Bits in a bit string are numbered beginning with 1. If VARYING is not specified, then the length in bits of the bit string is fixed at n. If VARYING is specified, then the length in bits of the string is variable, with a minimum length of 0 and a maximum length of n.

A bit string is either a fixed length or variable length bit string. The following named bit string data types

<table>
<thead>
<tr>
<th>Generic Data Type</th>
<th>Subclassification</th>
<th>Named Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte String</td>
<td>Variable Length</td>
<td>BLOB</td>
</tr>
<tr>
<td>(see page 18)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character String</td>
<td>Fixed Length</td>
<td>CHARACTER (CHAR)</td>
</tr>
<tr>
<td>(see page 18)</td>
<td>Variable Length</td>
<td>CHARACTER VARYING (VARCHAR)</td>
</tr>
<tr>
<td></td>
<td>Variable Length</td>
<td>CLOB</td>
</tr>
<tr>
<td>Numeric</td>
<td>Exact Numeric</td>
<td>DECIMAL (DEC)</td>
</tr>
<tr>
<td>(See page 18)</td>
<td></td>
<td>INTEGER (INT)</td>
</tr>
<tr>
<td></td>
<td>Approximate Numeric</td>
<td>NUMERIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Date/Time</td>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td>(see page 19)</td>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Interval</td>
<td>Long Interval</td>
<td>INTERVAL YEAR</td>
</tr>
<tr>
<td>(see page 21)</td>
<td></td>
<td>INTERVAL MONTH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL YEAR TO MONTH</td>
</tr>
<tr>
<td></td>
<td>Short Interval</td>
<td>INTERVAL DAY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL HOUR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL MINUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL SECOND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL DAY TO HOUR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL DAY TO MINUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL DAY TO SECOND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL HOUR TO MINUTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL HOUR TO SECOND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL MINUTE TO SECOND</td>
</tr>
<tr>
<td>Security</td>
<td>Fixed Length</td>
<td>SHORT LABEL</td>
</tr>
<tr>
<td>(see page 23)</td>
<td>Variable Length</td>
<td>LONG LABEL</td>
</tr>
</tbody>
</table>
are defined.

<table>
<thead>
<tr>
<th>Named Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT((n)) 1 &lt; (n) &lt; 25,165,824</td>
<td>Bit string of fixed length (n) bits.</td>
</tr>
<tr>
<td>BIT VARYING((n)) 1 &lt; (n) &lt; 25,165,824</td>
<td>Variable length bit string with a maximum string length (n) bits.</td>
</tr>
</tbody>
</table>

### 2.5.2 Byte String

The following named bit string data types are defined.

<table>
<thead>
<tr>
<th>Named Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>Variable length of bytes with a maximum length of 2,147,483,647 bytes.</td>
</tr>
</tbody>
</table>

### 2.5.3 Character String

A character string is a sequence of characters taken from a specific character set. A character string has a length, which is the number of characters in the sequence. Any two strings that use the same character set can be compared, even if their lengths differ.

A character string is either a fixed length or variable length character string. For variable length strings, the length may be zero. The following named character string data types are defined:

<table>
<thead>
<tr>
<th>Named Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER((n)) 1 &lt; (n) &lt; 3,145,728</td>
<td>Character string of fixed length (n) characters.</td>
</tr>
<tr>
<td>CHARACTER VARYING((n)) 1 &lt; (n) &lt; 3,145,728</td>
<td>Variable length character string with a maximum string length (n) characters.</td>
</tr>
<tr>
<td>CLOB</td>
<td>Variable length character string with a maximum length of 2,147,483,647 characters.</td>
</tr>
</tbody>
</table>

The application may omit the parenthesized length specification \((n)\) following CHARACTER. The default length is 1.

**Trusted Rubix** SQL also allows CHAR as a synonym for CHARACTER and VARCHAR as a synonym for CHARACTER VARYING.

### 2.5.4 Numeric

Each numeric value is either exact or approximate.

An exact numeric value has a precision and scale. The precision is a positive integer that determines the total number of significant decimal digits. The scale is a non-negative integer representing the number of
significant decimal digits to the right of the decimal point.

An approximate numeric value consists of a mantissa and an exponent. The mantissa is a signed numeric value and the exponent is a signed integer value that specifies the magnitude of the mantissa. An approximate numeric value has a precision, which represents the number of significant bits in the mantissa.

The following named numeric data types are defined:

<table>
<thead>
<tr>
<th>Named Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECIMAL(p,s) 1 &lt;= p &lt;= 50; 0 &lt; s &lt; p</td>
<td>Exact numeric, signed, precision p, scale s.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>Approximate numeric, signed, mantissa precision 47 bits; value zero or absolute value 10^{-38} to 10^{38}.</td>
</tr>
<tr>
<td>FLOAT(p) 1 &lt; p &lt; 47</td>
<td>Exact numeric, signed, mantissa precision at least p; value zero or absolute value 10^{-38} to 10^{38}.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Exact numeric, signed, precision 10 digits, scale zero; value from -2,147,483,648 through 2,147,483,647.</td>
</tr>
<tr>
<td>NUMERIC(p,s) 1 &lt;= p &lt;= 50; 0 &lt; s &lt; p</td>
<td>Exact numeric, signed, precision p, scale s.</td>
</tr>
<tr>
<td>REAL</td>
<td>Approximate numeric, signed, mantissa precision 21 bits; value zero or absolute value 10^{-38} to 10^{38}.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Exact numeric, signed, precision 5 digits, scale zero; value from -32,768 through 32,767.</td>
</tr>
</tbody>
</table>

Following DECIMAL and NUMERIC, the application may omit the scale, writing (p) as a synonym for (p,0), or omit the precision and scale specifications entirely. The application may likewise omit the precision specification following FLOAT. The default scale is 0. Default precisions for DECIMAL, NUMERIC, and FLOAT data are discussed on page 104.

For both DECIMAL and NUMERIC, the value all nines (9) is reserved. Thus, depending on how many decimal digits are defined, the value is represented by 1 to 50 nines.

**Trusted Rubix** allows DEC as a synonym for DECIMAL and INT as a synonym for INTEGER.

### 2.5.5 Date/Time

A date/time value contains some or all of the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND. These fields always occur in the order listed here, which is from most significant to least significant. Each field is an integer except that the SECOND field can have an integer fractional seconds component. Date/time values are not character strings and cannot be used interchangeably with character strings in comparison or assignment unless explicitly converted by use of the CAST function.

The three classes of date/time data type are as follows:

- **DATE**
  
  Contains the YEAR, MONTH and DAY fields;

- **TIME**
  
  Contains the HOUR, MINUTE and SECOND fields; and,

- **TIMESTAMP**
  
  Contains the YEAR, MONTH, DAY, HOUR, MINUTE and SECOND fields.

A date/time data item may have the null value, but if a date/time data item is non-null, then all fields required by the item’s class must be non-null.
Each field that is present in a date/time value has a non-negative value that denotes a date (using the Gregorian calendar) and/or a time (using a 24-hour clock). The limit on the value of each field is that normally imposed by the Gregorian calendar and the 24-hour time system:

- \( 0001 \leq \text{YEAR} \leq 9999 \)
- \( 01 \leq \text{MONTH} \leq 12 \)
- \( 01 \leq \text{DAY} \leq 31 \), constrained by MONTH and YEAR
- \( 00 \leq \text{HOUR} \leq 23 \)
- \( 00 \leq \text{MINUTE} \leq 59 \)
- \( 00 \leq \text{SECOND} \leq 61 \)

A date/time is only meaningful in conjunction with a time zone. However, date/time values in **Trusted Rubix SQL** do not specify a time zone. Neither clients nor servers are required to account for different time zones or convert date/time values from one time zone to another. If the application or the user cannot infer a time zone (for example, from the context of the user’s session) then the date/time value is ambiguous.

The following named date/time data types are defined:

<table>
<thead>
<tr>
<th>Named Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>Date/time value describing a date. Its length is 10 and it contains a YEAR, MONTH and DAY field (constrained as described on page 19) in the format: YYYY-MM-DD</td>
</tr>
<tr>
<td>TIME(p)</td>
<td>Date/time value describing the time in an unspecified day, with precision ( p ) (( 0 \leq p \leq 6 )). Its length is 8 (or ( 9+p ) if ( p &gt; 0 )) and it contains an HOUR, MINUTE, and SECOND field (constrained as described on page 19) in the format: HH:MM:SS [.F] where F is the fractional part of the SECOND field.</td>
</tr>
<tr>
<td>TIMESTAMP(p)</td>
<td>Date/time value describing both a date and a time, with precision ( p ) (( 0 \leq p \leq 6 )). Its length is 19 (or ( 20+p ) if ( p &gt; 0 )) and it contains a YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND field (constrained as described on page 19) in the format: YYYY-MM-DD HH:MM:SS [.F] where F is the fractional part of the second field.</td>
</tr>
</tbody>
</table>

---

3 A seconds value of 60 or 61 may be necessary during situations that require leap second adjustment.

4 Time zones are political divisions of the earth’s surface within which the time is the same and approximates the time as it would be measured based on the sun. However, the law may specify different time offsets within a time zone based on political subdivisions or varying over the course of the year.

5 Applications that have to reconcile date/time values that represent times in different time zones should associate time zone information with each date/time value. However, the application should do so in a way that can accommodate database products in which date/time values contain zone information.
The application may omit the parenthesized specification of seconds precision (p) following TIME and TIMESTAMP. The default seconds precision is 6 for TIME and 6 for TIMESTAMP which represents microseconds.

2.5.6 Interval

An interval value is one of the following:

→ one or both of the fields YEAR and MONTH; or,
→ one or more consecutive fields from the set: DAY, HOUR, MINUTE and SECOND.

Interval Precision

Named interval data types (which are enumerated on page 21) have an interval precision, which is a list of all the fields that values of that data type may contain. For example, in a value of type INTERVAL HOUR TO SECOND, the date/time precision is HOUR, MINUTE, SECOND.

Leading Precision

Conceptually, an interval is a signed numeric quantity comprising a specific set of fields. The fields have the same ordering, relative significance, and Gregorian calendar constraints as described above for date/time values except as described below.

In all intervals, the high-order field is not constrained by the Gregorian calendar. Named interval data types have a leading precision, which is the number of decimal digits that the high-order field can accommodate.

For example, in an INTERVAL HOUR TO MINUTE, the MINUTE field is constrained as described above but the HOUR field is constrained only by the leading precision. If interval arithmetic generates an intermediate value of 01:61, Trusted Rubix converts it to 02:01 before making it visible to the application.

Seconds Precision

Any interval data type that has a SECOND field has a seconds precision. This is the number of decimal digits allowed in the fractional part of the SECOND value.

In the named data type INTERVAL SECOND, the SECOND field is the only field and has both a leading precision and a seconds precision.

Interval Qualifier

The syntactic element interval-qualifier follows every use of the keyword INTERVAL to specify the interval precision (the set of available fields). Valid interval-qualifiers are listed on page 21. The interval-qualifier is used in certain other cases as well.

An interval-qualifier specifies or implies the interval precision, leading precision, and seconds precision (if applicable) of the value.

Named Interval Data Types

A named interval data type consists of the word INTERVAL followed by one of the following interval-qualifiers:
### Description of Named Data Type

<table>
<thead>
<tr>
<th>interval-qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR(p)</td>
<td>Interval class describing a number of years, with leading precision p. It contains a YEAR field in the format: YYYY</td>
</tr>
<tr>
<td>MONTH(p)</td>
<td>Interval class describing a number of months, with leading precision p. It contains a MONTH field in the format: MM</td>
</tr>
<tr>
<td>YEAR(p) TO MONTH</td>
<td>Interval class describing a number of years and months, with leading precision p. Its format is: YYYY-MM</td>
</tr>
<tr>
<td>DAY(p)</td>
<td>Interval class describing a number of days. Its leading precision is p. It contains a DAY field in the format: DD</td>
</tr>
<tr>
<td>HOUR(p)</td>
<td>Interval class describing a number of hours, with leading precision p. It contains an HOUR field in the format: HH</td>
</tr>
<tr>
<td>MINUTE(p)</td>
<td>Interval class describing a number of minutes, with leading precision p. It contains a MINUTE field in the format: MM</td>
</tr>
<tr>
<td>SECOND(p,s)</td>
<td>Interval class describing a number of seconds, with leading precision p and seconds precision s. It contains a SECOND field in the format: SS [.F]</td>
</tr>
<tr>
<td>DAY(p) TO HOUR</td>
<td>Interval class describing a number of days and hours, with leading precision p. Its format is: DD HH</td>
</tr>
<tr>
<td>DAY(p) TO MINUTE</td>
<td>Interval class describing a number of days, hours and minutes, with leading precision p. Its format is: DD HH:MM</td>
</tr>
<tr>
<td>DAY(p) TO SECOND(s)</td>
<td>Interval class describing a number of days, hours, minutes and seconds, with leading precision p and seconds precision s. Its format is DD HH:MM:SS [.F]</td>
</tr>
<tr>
<td>HOUR(p) TO MINUTE</td>
<td>Interval class describing a number of hours and minutes, with leading precision p. Its format is: HH:MM</td>
</tr>
<tr>
<td>HOUR(p) TO SECOND(s)</td>
<td>Interval class describing a number of hours, minutes and seconds, with leading precision p and seconds precision s. Its format is: HH:MM:SS [.F]</td>
</tr>
<tr>
<td>MINUTE(p) TO SECOND(s)</td>
<td>Interval class describing a number of minutes and seconds, with leading precision p and seconds precision s. Its format is: MM:SS [.F]</td>
</tr>
</tbody>
</table>

The application may omit the parenthesized specification of leading precision (p). The default leading precision is 2, even if the high-order field is YEAR. The application may omit the parenthesized specification of seconds precision (s). The default seconds precision is Trusted Rubix-defined. In the case of INTERVAL SECOND (p, s), the application may omit the seconds precision, writing (p) and implying the TR default seconds precision. The application can also omit both precision specifications.

### Length of an Interval

The three types of precision determine the maximum length of any interval value. Length is expressed in positions. This is the same as a number of characters of the string representation of the interval (using a
The rules for computing an interval’s maximum length are as follows:

- Allow two positions for every field in the interval that is not the high-order field.
- For the high-order field, allow the number of positions that is the leading precision.\(^6\)
- Add one position for each separator between fields.
- If the seconds precision is nonzero, add that number of positions, plus one for the decimal point.

Applying these rules to each named interval data type produces the following lengths, where \(p\) is the leading precision and \(s\) is the seconds precision (specified or implied):

<table>
<thead>
<tr>
<th>(\text{interval-qualifier})</th>
<th>Length of Named Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>(p)</td>
</tr>
<tr>
<td>MONTH</td>
<td>(p)</td>
</tr>
<tr>
<td>YEAR TO MONTH</td>
<td>(3 + p)</td>
</tr>
<tr>
<td>DAY</td>
<td>(p)</td>
</tr>
<tr>
<td>HOUR</td>
<td>(p)</td>
</tr>
<tr>
<td>MINUTE</td>
<td>(p)</td>
</tr>
<tr>
<td>SECOND</td>
<td>(p) (or (p + s + 1) if (s &gt; 0))</td>
</tr>
<tr>
<td>DAY TO HOUR</td>
<td>(3 + p)</td>
</tr>
<tr>
<td>DAY TO MINUTE</td>
<td>(6 + p)</td>
</tr>
<tr>
<td>DAY TO SECOND</td>
<td>(9 + p) (or (10 + p + s) if (s &gt; 0))</td>
</tr>
<tr>
<td>HOUR TO MINUTE</td>
<td>(3 + p)</td>
</tr>
<tr>
<td>HOUR TO SECOND</td>
<td>(6 + p) (or (7 + p + s) if (s &gt; 0))</td>
</tr>
<tr>
<td>MINUTE TO SECOND</td>
<td>(3 + p) (or (4 + p + s) if (s &gt; 0))</td>
</tr>
</tbody>
</table>

### 2.5.7 Security

The following named security data type is defined:

<table>
<thead>
<tr>
<th>Named Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABEL</td>
<td>Sensitivity label string of variable length.</td>
</tr>
</tbody>
</table>

### 2.5.8 Rules For Determining Data Types

#### Data Types of Table Columns

Each column of a base table is given a named data type when the column is created (that is, when the table is created or when the column is added to the table). The data types of a derived table’s columns may be generic data types or named data types:

- Columns of derived tables, defined by a `query-specification`, derive their data types from the `expressions` in the `query-specification` that define columns.
- Columns of a viewed table inherit the data types of the corresponding result columns of the `query-specification` that defines the view.
- Columns defined by a cursor specification inherit the data types of the corresponding result columns of the first `query-specification` in the cursor specification.

---

\(^6\) This algorithm, derived from the International Standard, does not allow for the minus sign that is present in the string representation of negative intervals. Application writers are advised that such a string representation could be one character longer than the maximum length.
Data Types of Expressions

If an expression (see page 31) contains a single operand, the data type of the expression is the data type of that operand. The data type of the result of an arithmetic operation depends on the data type of the two operands and is determined as follows:

→ If the data type of either operand is approximate numeric, the data type of the result is approximate numeric with Trusted Rubix precision and range of magnitude.

→ If the data type of the result is approximate numeric, the result must be within the TR range of magnitude.

→ If the data type of both operands is exact numeric, the data type of the result is exact numeric with TR precision and range, and scale dependent on the operation as follows:
  ↑ If the operation is addition or subtraction, the scale is the larger of the scales of the two operands.
  ↑ If the operation is multiplication, the scale is the sum of the scales of the two operands.
  ↑ If the operation is division, the scale is Trusted Rubix-defined.

If the data type of the result is exact numeric and the operator is not division, the result must be exactly representable within its data type. If the data type of the result is exact numeric and the operator is division, the result must be representable within its data type without losing any leading significant digits.

The data type of a result of a set function depends on the function and on the type of its argument. (See page 36.)

The data type of a result of a date/time or interval computation is specified on page 32.

The data type of a result of the various string operations is specified on page 33.

The data type of an expression can be explicitly converted to another data type using the CAST function.

2.6 LITERALS

A literal is a sequence of characters representing a value. The classification of literals is according to the values that they represent.

2.6.1 Bit String Literals

Written either as a sequence of 0s and 1s enclosed in single quotes and preceded by the letter B or as a sequence of hexadecimal digits enclosed in single quotes and preceded by the letter X.

  e.g.: B'11000001'  B'0101'  B'0'
  X'C1'  X'5'  X'fed'

NOTE

The hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 A (or a), B (or b), C (or c), D (or d), E (or e), and F (or f). They represent the bit strings 0000 through 1111 in the usual way. Thus, the 1st and 4th of the examples above represent the same value, as do the 2nd and 5th.

The data type of a “bit-string-literal” is fixed length bit string.
2.6.2 Character String Literals

A `character-string-literal` represents a character string and consists of a sequence of characters delimited at each end by the single quote character. That is, it has the following format:

```
' {character} . . '
```

Within the delimiters, a single quote character is represented by two consecutive single quote characters. The value of the `literal` is the value of the sequence of characters within the delimiters. The data type of a `character-string-literal` is fixed length character string.

2.6.3 Numeric Literals

A `numeric-literal` represents a number and consists of a character string whose characters are selected from the digits 0 through 9, the plus sign, the minus sign, the decimal point and the character `E`.

A `numeric-literal` is either an `exact-numeric-literal` representing an exact numeric value or an `approximate-numeric-literal` representing an approximate numeric value. An `exact-numeric-literal` has the following format:

```
[ + | - ] {unsigned-integer} [. unsigned-integer]
```

where `unsigned-integer` is defined as:

```
{digit} . .
```

The data type of an `exact-numeric-literal` is exact numeric, its precision is the number of digits that it contains, and its scale is the number of digits to the right of the decimal point. The value of an `exact-numeric-literal` is derived from the normal mathematical interpretation of the specified notation.

An `approximate-numeric-literal` has the format `mantissa`E`exponent`, where the `mantissa` is an `exact-numeric-literal` and the `exponent` has the form:

```
[ + | - ] unsigned-integer
```

The data type of an `approximate-numeric-literal` is approximate numeric and its precision is the precision of its `mantissa`.

The value of an `approximate-numeric-literal` is the product of the value represented by the `mantissa` with the number obtained by raising the number 10 to the power represented by the `exponent`.

2.6.4 Date/Time Literals

A date/time literal represents a date/time value and consists of one of the keywords shown below, followed by text delimited at each end by a single quote character. It can have the following formats:

```
→ DATE 'date-value'
→ TIME 'time-value'
→ TIMESTAMP 'date-value space time-value'
```

The quoted text contains the fields specified on page 19 for the respective date/time subtype, in the order
and with the limits described there, using the delimiters shown on page 19 (‘2007’).

A date-value has the following format:

\[\text{year-value-month-value-day-value}\]

A time-value has the following format:

\[\text{hour-value : minute-value : second-value}\]

Where year-value, month-value, day-value, hour-value and minute-value are unsigned-integers and second-value has the following format:

\[\text{unsigned-integer[.unsigned-integer]}\]

and-, , and space above represent single characters determined by the character set in use.

### 2.6.5 Interval Literals

An interval literal represents a date/time value and consists of the INTERVAL keyword followed by text delimited at each end by a single quote character. It can have the following formats:

\[\text{INTERVAL} \ [+ | - ] \ ‘\text{interval-value}’ \ ‘\text{interval-qualifier}’\]

The text of interval-value must be a valid representation of a value of the named interval data type specified by interval-qualifier. The text contains a decimal value for every field implied by interval-qualifier.

If the interval precision includes the fields YEAR and MONTH, values of these fields are separated by the minus sign.

→ If the interval precision includes the fields DAY and HOUR, values of these fields are separated by a space.

→ If the interval precision includes HOUR and lower-order fields, values of these fields are separated by the colon.

→ No field value can be more than two digits long except that:

↑ The value of a YEAR field can be 4 digits long;

↑ The value of the high-order field can be as long as the leading precision specified in interval-qualifier; and,

↑ The SECOND field can have a fractional part whose maximum length is specified or implied by interval-qualifier.

### 2.6.6 Pseudo-literals

A pseudo-literal is a syntactic element that can be used as a literal. Rather than a constant value, it returns variable information obtained from Trusted Rubix.

**Current Date and Time**

The following keywords denote date/time values: 7

→ CURRENT_DATE

7 The International Standard classifies these three pseudo-literals as date/time value functions. In Trusted RUBIX SQL, they are valid anywhere other date/time literals are, with one exception noted for column constraints.
A DATE value denoting the current date.

→ CURRENT_TIME
   A TIME value denoting the current time.

→ CURRENT_TIMESTAMP
   A TIMESTAMP value denoting the current date and time.

If a single SQL statement references several instances of CURRENT_DATE, CURRENT_TIME and CURRENT_TIMESTAMP, they all return values that denote the same date and time.

Current User and Group
The following keywords denote user/group values:

→ CURRENT_USER
   A CHAR value denoting the current user name.

→ CURRENT_USER_ID
   An INTEGER value denoting the current user ID.

→ CURRENT_GROUP
   A CHAR value denoting the current group name.

→ CURRENT_GROUP_ID
   An INTEGER value denoting the current group ID.

Current Session Label
The following keywords denote session label values:

→ SESSION_LABEL
   A LABEL value denoting the current session label.

Current Application
The following keywords denote application values:

→ CURRENT_APPLICATION
   A CHAR value denoting the current application.

Current Application User
The following keywords denote application user name and ID values:

→ CURRENT_APPLICATION_USER
   A CHAR value denoting the current application user name.

→ CURRENT_APPLICATION_USER_ID
   An INTEGER value denoting the current application user ID.

2.7 ASSIGNMENT

2.7.1 Bit String Assignments
When non-null bit values are retrieved from the database (e.g., via SELECT) and the target object is fixed length, the source string expression is conceptually truncated on the right or padded on the right with 0-bits (as necessary) to make it the same length as the target before the assignment is performed. When non-
null bit values are stored in the database (e.g., via UPDATE), truncation is not allowed.

If the target object is varying length, its current actual length is set to the lesser of its declared maximum length and the length of the source expression, and the assignment is then performed as in the fixed length case.

2.7.2 Character String Assignments

The following rules apply when assigning a non-null character string of length $S$ (source) to a table column or host variable whose data type is fixed-length character string of length $D$ (destination) or variable length character string of maximum length $D$.

1. If the assignment is to a table column and $S$ is larger than $D$, then the rightmost $S-D$ characters of the source string must all be blanks (‘22001’) and only the leftmost $D$ characters of the source string are assigned.

2. If the assignment is to a host variable and $S$ is larger than $D$, the leftmost $D$ characters of the character string are assigned (‘01004’).  

3. If $S$ is smaller than $D$, the character string is assigned to the leftmost $S$ character positions of the destination field. If the destination's data type is fixed-length character string, the remaining $D-S$ positions are blank-filled.

4. If $S$ is equal to $D$, the character string is assigned to the destination field.

2.7.3 Character String Storage for C

For C, a character string of length $D$ requires an array of $D + 1$ elements. The elements are numbered 0 through $D$. The application must place a null byte (‘\0’) in the position following the last one that contains character data, in order to ensure correct interpretation by the database (‘22024’). When assigning a value to such a host variable, the database places a null byte in the $D$th position.

2.7.4 Numeric Assignments

When assigning a non-null numeric value to an exact numeric table column or host variable, there must be a representation of the numeric value in the data type of the destination field that does not cause the whole part of the number (that is, the leading significant digits) to be truncated (‘22003’). The fractional part of the number (that is, the trailing significant digits) may be truncated as necessary and warning flags or indicator variables set.

When assigning a non-null numeric value to an approximate numeric table column or host variable, the numeric value must be within the range of magnitude of the destination field (‘22003’). The result is an approximation of the source numeric value that has the precision of the destination field.

2.7.5 Date/Time and Interval Assignment

When assigning a non-null date/time value to a column, the data type of the target location must also be

---

8 Truncation of non-blank data on assignment to the database is an error (and the SQL statement fails), whereas truncation on extracting data from the database is just a warning, whether or not the truncated data is blank.

Indicator variables can detect truncation, as described in section 2.7.4.
date/time.

Its subtype (DATE, TIME or TIMESTAMP) must be the same as that of the source (‘2007’).

When assigning a non-null value to a date/time table column, the leading precision of the target must be sufficient to represent the value of the source (‘2008’).

When assigning a non-null value to an interval table column, the leading precision of the target must be sufficient to represent the value of the source (‘2015’).

Truncation of a fraction in a SECOND field is subject to the rules stated above for *Numeric Assignments*.

### 2.8 COMPARISON

#### 2.8.1 Bit

Bit strings are compared bit by bit from left to right. The individual pair-wise comparisons are performed in accordance with the convention that a 0-bit is less than a 1-bit. If every pair-wise comparison yields “equal,” the bit strings are equal; otherwise, the first pair-wise comparison that does not yield “equal” determines the overall result.

If two bit strings of different length are to be compared, the shorter is conceptually padded at the end with 0-bits to make it the same length as the other string before the comparison is carried out. Two bit strings may compare equal even if they are of different length.

#### 2.8.2 Character

All character strings taken from the same character set are comparable and are compared from the start to the end of the string with respect to the collating sequence of the character set they are taken from.

If two character strings of different lengths are to be compared, the shorter is conceptually padded at the end with blank characters to make it the same length as the other string before the comparison is carried out. Two character strings may compare equal even if they are of different length or contain different sequences of characters. If this is the case in the MAX and MIN functions, in references to a grouping column, or in the operations specified by the DISTINCT and UNION keywords, one of the equal values is chosen, using undefined criteria.

#### 2.8.3 Numeric

All numbers are comparable and are compared according to their algebraic value.

#### 2.8.4 Date/Time

Two date/time values are comparable if their subtype (DATE, TIME or TIMESTAMP) is identical. When comparing two TIME or TIMESTAMP values, *Trusted Rubix* extends the seconds precision of the value with the lower precision by adding trailing zeros.

#### 2.8.5 Interval

Two interval values are comparable only if they have some fields in common. Interval types with YEAR or
MONTH fields are not comparable to interval types with DAY, HOUR, MINUTE or SECOND interval types. Interval types with different interval precision are conceptually converted to the same precision before comparison by adding fields as required. 9

2.8.6 Large Objects (LOBs)

LOBs (BLOBs and CLOBs) may not be used in any comparison operation.

2.9 NULL VALUES

Every data type includes the null value, which is distinct from all non-null values. The null value represents a value that is unknown or not applicable.

There is no literal that represents the null value. However, in some situations, SQL syntax uses the keyword NULL to represent the null value.

A table may constrain certain columns (using NOT NULL) to contain only non-null values. SQL inserts null values into tables in several situations when the application fails to specify or imply non-null values for insertion into the table. In these cases, the destination columns must not be constrained by NOT NULL.

The application and TR can communicate null values in embedded host variables by using an associated indicator-variable.

2.9.1 Null Values in Computations

If the value of any operand in an expression is the null value then the value of the expression is the null value. (In particular, subtracting a null value from a null value evaluates to null and not to zero.)

If any operand in an expression is allowed to contain the null value, the result of that expression is conceptually allowed to contain the null value.

Assigning Null Values

In a TR operation that would assign a null value to a table column, the column must be allowed to contain null values (that is, NOT NULL must not have been specified).

In a TR operation that would assign a null value to a host variable, the application must provide an associated indicator variable (‘22002’). The content of the host variable is undefined.

2.9.2 Comparing Null Values Explicitly

Generally, comparing a null value with any value, even another null value, evaluates to the unknown truth value. The only way to test for the presence or absence of null values is the IS [NOT] NULL predicate (see page 43).

---

9 Added low-order fields have the value zero. Added high-order fields have the value zero unless the previous high-order field exceeds the limit imposed by the Gregorian calendar; in this case its value is normalized, resulting in a nonzero value in the new field. For example, to compare an INTERVAL MINUTE containing the valid value 75 to an INTERVAL HOUR TO MINUTE, the invalid value 0:75 is converted to the valid value 1:15.
2.9.3 Implicit Comparisons

Certain SQL comparisons and predicates imply comparison of values. The following table describes how these comparisons handle null values:

<table>
<thead>
<tr>
<th>Context</th>
<th>Treatment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOREIGN KEY</td>
<td>If any column of a FOREIGN KEY is null, the referential constraint is met.</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>Null grouping column values are treated as equal.</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Null values are treated as equal and are all greater than non-null values.</td>
</tr>
<tr>
<td>PRIMARY KEY</td>
<td>Null values are not allowed.</td>
</tr>
<tr>
<td>SELECT DISTINCT</td>
<td>Null values are treated as equal.</td>
</tr>
<tr>
<td>Set functions</td>
<td>Null values are eliminated regardless of whether DISTINCT is specified except that COUNT( *) includes all null values.</td>
</tr>
<tr>
<td>UNIQUE columns</td>
<td>Null values are treated as not equal, so multiple rows with null values in the same columns are allowed.</td>
</tr>
<tr>
<td>UNIQUE INDEX</td>
<td>Null values are treated as equal, so the table cannot have multiple rows with null values in the column.</td>
</tr>
</tbody>
</table>

2.10 EXPRESSIONS

An expression represents a single value. It consists of one of the following:

- a host-variable-reference;
- a literal (see page 24), including pseudo-literals;
- a column-name;
- a set-function-reference (see page 36);
- a dynamic-parameter (see page 38);
- one of the scalar functions CHAR_LENGTH, CHARACTER_LENGTH, EXTRACT, OCTET_LENGTH, POSITION, SUBSTRING or TRIM (see page 33); or the CAST function (see page 38);
- any valid combination of these primary components connected by operators.

2.10.1 Arithmetic Operators

Arithmetic operators combine numeric expressions and yield a numeric value. The following arithmetic operators are allowed (in descending order of precedence):

- +, -   unary plus and minus (negation)
- *, /   multiplication and division
- +, -   addition and subtraction
The operand following a unary plus or minus must not be a signed numeric literal.

Operations with the same precedence are executed from left to right. Parentheses may be used to depart from the above precedence order since, when used in an expression, they determine the binding of expression components to arithmetic operators.

Arithmetic errors occur in evaluating an expression (that does not involve date/time or interval values) on a violation of the following rules:

→ No computed value, as a final or intermediate result, may exceed the range of the data type ('2003').
→ Division by zero may not be performed ('2012').

2.10.2 Date/Time and Interval Arithmetic

The following table lists the only valid arithmetic operations involving values whose generic data type is date/time or interval:

<table>
<thead>
<tr>
<th>Operand 1</th>
<th>Operator</th>
<th>Operand 2</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date/Time</td>
<td>-</td>
<td>Date/Time</td>
<td>Interval</td>
</tr>
<tr>
<td>Date/Time</td>
<td>+ or -</td>
<td>Interval</td>
<td>Date/Time</td>
</tr>
<tr>
<td>Interval</td>
<td>+</td>
<td>Date/Time</td>
<td>Date/Time</td>
</tr>
<tr>
<td>Interval</td>
<td>+ or -</td>
<td>Interval</td>
<td>Interval</td>
</tr>
<tr>
<td>Interval</td>
<td>* or /</td>
<td>Numeric</td>
<td>Interval</td>
</tr>
<tr>
<td>Numeric</td>
<td>*</td>
<td>Interval</td>
<td>Interval</td>
</tr>
</tbody>
</table>

Operands cannot be combined arithmetically unless their data types are comparable (as defined on page 31). If either operand is the null value, then the result of any arithmetic operation is the null value.

If arithmetic involves two date/time or interval values with a defined scale, then the scale of the result is the larger of the scales of the two operands.

Interval arithmetic that involves date ranges that span discontinuities in calendars produces Trusted Rubix-defined results.

Arithmetic errors occur in date/time and interval arithmetic, on a violation of the following rules:

→ No computed date/time value, as a final or intermediate result, can exceed the leading precision of the data type ('2008').
→ No computed interval value, as a final or intermediate result, can exceed the leading precision of the data type ('2015').
→ Division by zero may not be performed ('2012').

Subtracting Two Date/Times

One date/time value may be subtracted from another to produce an interval that is the signed difference.

10 Discontinuities are decisions to add units to, or remove units from, a calendar at a specified point in time. The International Standard lists leap seconds as an example of discontinuity. Another example is the addition of 10 days to the Gregorian calendar in the 18th century. The acceptance and timing of a discontinuity depends on the locale.
between the stated dates or times. However, the application program must specify a named interval data type for the result by using an interval-qualifier. The required syntax is:

\[(\text{date-time-1} \ | \ \text{date-time-2}) \text{ interval-qualifier}\]

### 2.10.3 EXTRACT Function

The **EXTRACT** function extracts a single numeric field from a date/time or interval value. It has the form:

\[\text{EXTRACT} (\text{field-name FROM expression})\]

where \text{field-name} is one of \text{YEAR}, \text{MONTH}, \text{DAY}, \text{HOUR}, \text{MINUTE} or \text{SECOND}. The \text{expression} must be of type date/time or interval and must contain the field specified by \text{field-name}.

The data type of the result is exact numeric. The precision and scale are \text{Trusted Rubix} -defined except that, for \text{field-names} other than \text{SECOND}, the scale is 0.

If \text{expression} is null, then the result is the null value. Extracting any field from a negative interval produces a negative value. All other uses of \text{EXTRACT} produce a non-negative value.

### 2.10.4 String Operations

The following table lists the valid operations on character strings. In the table, \(s1\) and \(s2\) stand for \text{character-string} operands and \(i\) and \(j\) stand for \text{arithmetic-expressions}. Additional valid syntax and details of the respective operations follow the table. String operations may not be used with the CLOB data type.

<table>
<thead>
<tr>
<th>Simplified Syntax</th>
<th>Result Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s1</td>
<td></td>
</tr>
<tr>
<td>\text{CHAR_LENGTH}(s1)</td>
<td>An exact numeric indicating the length of (s1).</td>
</tr>
<tr>
<td>\text{CHARACTER_LENGTH}(s1)</td>
<td>Same as \text{CHAR_LENGTH}.</td>
</tr>
<tr>
<td>\text{LOWER}(s)</td>
<td>A copy of (s) with any uppercase letters converted to lowercase.</td>
</tr>
<tr>
<td>\text{OCTET_LENGTH}(s1)</td>
<td>Same as \text{CHAR_LENGTH}.</td>
</tr>
<tr>
<td>\text{POSITION}(s1 \ \text{IN} \ s2)</td>
<td>An exact numeric indicating the character position of the first occurrence of (s1) in (s2).</td>
</tr>
<tr>
<td>\text{SUBSTRING}(s1 \ \text{FROM} \ i \ [\text{FOR} \ j])</td>
<td>A string extracted from (s1), starting at character position (i) and continuing for (j) characters.</td>
</tr>
<tr>
<td>\text{TRIM}(s1)</td>
<td>A string consisting of (s1) with leading and/or trailing pad characters removed.</td>
</tr>
<tr>
<td>\text{UPPER}(s)</td>
<td>A copy of (s) with any lowercase letters converted to uppercase.</td>
</tr>
</tbody>
</table>

### Long Strings

An exact numeric return value is specified above for \text{CHAR\_LENGTH}, \text{CHARACTER\_LENGTH}, \text{OCTET\_LENGTH} and \text{POSITION}. Applications can use integer values to retrieve these return values, and elsewhere to represent the length of, or position within, a string. However, if it is possible that the string
length could exceed the capacity of an integer variable, C applications should do one of the following:

- use `long` and be prepared to handle overflow exceptions; or,
- use `double` and be prepared for the result to be inexact without notification.

**Concatenation**

The result of the concatenation operation is the value of the string operand preceding the `||` operator, followed by the value of the string operand following the `||` operator.

If the data type of both operands is a fixed length string, then the data type of the result is a fixed length string. If the data type of either operand is a variable length string, then the data type of the result is a variable length string. If the value of either operand is null, the result is the null value.

The length of the result is the sum of the lengths of the operands except when this is excessive:

- If the data type of both operands is a fixed length string and if the length of the resulting string exceeds the **Trusted Rubix** maximum for fixed length strings, the operation fails (`'42000'`).
- If the data type of either operand is a variable length string and if the length of the resulting string exceeds the **TR** maximum for variable length strings, then:
  - if all the characters in the positions beyond the maximum length are the space character, the result is truncated on the right to the maximum length (without warning);
  - otherwise, the operation fails (`'22001'`).

**Capitalization Functions**

The `LOWER` function returns a copy of its string argument, in which any uppercase letter is replaced with the corresponding lowercase equivalent. The `UPPER` function returns a copy of its string argument, in which any lowercase letter is replaced with the corresponding uppercase equivalent.

Uppercase letters, lowercase letters, and the correspondence referred to above are defined by the character set associated with the string argument. The string result is in the same character set. The expressions `UPPER(LOWER(s))` and `LOWER(UPPER(s))` do not necessarily yield `s` in all cases.

**BIT_LENGTH**

Returns the length of a string in bits. The string in question is specified by means of an arbitrary string expression.

**CHAR_LENGTH**

The `CHAR_LENGTH` function returns the length of the value (the number of characters) of its single character string operand.

- If the value of the operand is null, then the function returns the null value.
- The keyword `CHARACTER_LENGTH` can be used instead of `CHAR_LENGTH`. For single byte character sets, the keyword `OCTET_LENGTH` can be used instead of `CHAR_LENGTH`.

**POSITION**

The `POSITION` function returns an integer indicating the first position at which the value of one character string occurs within another. The first position in a character string is position number 1. `POSITION` returns the value 0 in several cases specified below where there is no such occurrence. The function has the form:
POSITION (search-string IN source-string)

where search-string and source-string are both string-expressions.

The function performs the following tests in sequence:

→ If the value of either operand is null, the function returns the null value.
→ If source-string has a length of zero, the function returns zero.
→ If the characters of search-string occur consecutively within source-string, then the function returns the character position of the start of the first such occurrence.
→ Otherwise (if search-string does not occur within source-string), the function returns zero.

All comparisons of characters within the two operands are case insensitive.

NOTE
POSITION does not apply to bit strings.

SUBSTRING
The SUBSTRING function returns a character string which is a portion (zero-length, partial, or complete) of its character string operand. The function has the form:

SUBSTRING (source-string, FROM start-position [FOR string-length])

where source-string is a string-expression and start-position and string-length are both arithmetic expressions.

The data type of the result is variable length character string, with the same maximum length as source-string (or as the fixed length of source-string, if it is a fixed-length string). The data type of both numeric operands is exact numeric with a scale of zero.

If string-length is omitted, then its value is assumed to be:

CHAR_LENGTH (source-string) + 1 – start-position

This denotes the entire remainder of source-string at and beyond start-position.

The SUBSTRING function performs the following steps in sequence:

→ If the value of any operand is null, the function returns the null value.
→ If string-length is negative, or if start-position is greater than the number of characters in source-string, the function fails (‘22011’).
→ Otherwise, the function returns a character string containing string-length characters of source-string, starting at the character specified by start-position and in the same sequence that the characters appear in source-string. (To the extent that some of these positions are before the start or beyond the end of source-string, then no character is returned. If all the characters specified are thus, then the result is the null string.)

TRIM
The TRIM function returns a character string that is the value of a string operand with certain pad
characters removed. The complete syntax of TRIM is as follows:

\[
\text{TRIM} \left( \left[ \left[ \text{LEADING} \mid \text{TRAILING} \mid \text{BOTH} \right] \left[ \text{trim-character} \right] \text{FROM} \right] \text{source-string} \right)
\]

where \text{trim-character} and \text{source-string} are both \text{string-expressions}.

The brackets above indicate optional syntax. However, if the word \text{FROM} is specified, then some valid syntax must appear between the opening parenthesis and the word \text{FROM}. If neither \text{LEADING}, \text{TRAILING} nor \text{BOTH} is specified, then \text{BOTH} is assumed.

The value of \text{trim-character} is the trim character; the single character that is to be removed from the start and/or end of \text{source-string}. If \text{trim-character} is omitted, then the space character is the trim character.

The data type of the result is variable-length character string, with the same maximum length as \text{source-string} (or as the fixed length of \text{source-string}, if it is a fixed-length string).

The \text{TRIM} function performs the following steps in sequence:

1. If the value of any operand is null, the function returns the null value.
2. If the length of the value of \text{trim-character} is not 1, the operation fails ('22027').
3. Otherwise, the function returns the value of \text{source-string} after trimming characters from it as follows:
   - If \text{LEADING} is specified, any consecutive occurrences of the trim character at the start of \text{source-string} are removed;
   - If \text{TRAILING} is specified, any consecutive occurrences of the trim character at the end of \text{source-string} are removed; and,
   - If \text{BOTH} is specified or assumed, both leading and trailing occurrences of the trim character are removed.

\textbf{String operations and CLOBs}

String functions may not be used with the CLOB data type.

\section*{2.10.5 Set Functions}

A set function can appear in an \textit{expression} in two contexts within a \textit{query-specification} (see page 44):

1. Within a \textit{select-sublist}, the set function analyses the entire virtual table.
2. Within the \textit{search-condition} in a \textit{HAVING} clause, the \textit{query-specification} specifies a grouped table and the set function analyses the current group.

A set function returns a single value as a result from a group (collection) of rows.

A \textit{set-function-reference} requires an argument in parentheses. The rules for a valid argument depend on the form used and are discussed below. The general format of a \textit{set-function-reference} is:

\[
\text{set-function-name} \left( \text{argument} \right)
\]

For the \texttt{COUNT} function, Trusted Rubix supports these two forms:

\begin{itemize}
  \item \texttt{COUNT (*)}
  \item \texttt{COUNT ( argument )}
\end{itemize}
The set-function-name may be any of those shown in the following table:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Data Type of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>Average of the values in the column.</td>
<td>Same generic data type as the argument; TR defined attributes.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Total number of values in (cardinality of) the column.</td>
<td>Exact numeric, scale 0, TR defined precision and range.</td>
</tr>
<tr>
<td>MAX</td>
<td>Largest value in the column.</td>
<td>Same as the argument.</td>
</tr>
<tr>
<td>MIN</td>
<td>Smallest value in the column.</td>
<td>Same as the argument.</td>
</tr>
<tr>
<td>SUM</td>
<td>Sum of the values in the column.</td>
<td>Exact numeric argument: Exact numeric result; same scale as argument; TR defined precision and range. Approximate numeric argument: Approximate numeric result; TR defined precision and range of magnitude.</td>
</tr>
</tbody>
</table>

Functions AVG and SUM can be applied only to numeric columns.

If DISTINCT is specified, the set function calculates the result after excluding duplicate values of the argument. In this case, argument must be a simple column-name and the set function cannot be combined with other terms using binary arithmetic operators. The keyword DISTINCT cannot be used in a query or sub-query which itself uses the form SELECT DISTINCT (see page 4-63).

If ALL is specified (or in the case of COUNT(*)), duplicate values are retained. In this case, the argument may be an expression but it must contain at least one column-name and may not contain any set-function-references.

If neither ALL nor DISTINCT is specified, ALL is assumed.

In both cases, any null argument values are eliminated before the function is applied. (If any null values are eliminated, SQLSTATE is set to (‘01003’).) However, COUNT (*) counts all values, even null values.

A set-function-reference may be specified only in an expression in a SELECT clause or in an expression that is contained within the search-condition of a HAVING clause. If it appears in a sub-query of a HAVING clause, its argument must be a correlated reference.

An expression directly contained within the search-condition of a WHERE clause must not reference a column derived from a set function.

A column-name specified in the argument of a set-function-reference must not reference a column

4 COUNT(*) does not refer to columns but returns the cardinality of the entire virtual table (or, if it occurs in a HAVING clause, the cardinality of the current group).
derived from a set function. If such a column-name is a correlated reference, then all of the following must be true:

→ The argument must consist solely of that column-name.
→ The set-function-reference must be contained in a sub-query of a HAVING clause.
→ The correlated reference must be to a table-reference contained in the FROM clause of the outer query that contains the HAVING clause.

Set Function Exceptions
The sum of the column values on which a call to AVG or SUM operates must be within the range of the data type of the result (‘22003’).

All set functions return the null value if applied to an empty set, except that COUNT (*) returns the value 0.

2.10.6 Dynamic Parameters
A dynamic parameter is represented by a question mark (?) and identifies a parameter in a dynamically prepared SQL statement.

If the context in which the dynamic parameter is used requires an interval, then the question mark must be followed by a suitable interval-qualifier to specify the interval precision. There are two exceptions, in which no interval-qualifier is required:

→ a quantity added to, or subtracted from, a DATE is assumed to be of type INTERVAL YEAR(p) to MONTH;
→ a quantity added to, or subtracted from, a TIME or TIMESTAMP is assumed to be of type INTERVAL YEAR (p) TO SECOND(s);
→ where p and s above are the Trusted Rubix defined maxima for the respective precision.

2.10.7 CAST Function
The CAST function explicitly converts data of one data type to another data type. The CAST function has the form:

```
CAST ( ( { expression | NULL } ) AS data-type )
```

where data-type is one of the named data types specified on page 16.

The resulting data type of the CAST function is data-type. Applying the CAST function to NULL, or to an expression that has the null value, yields the null value of the target data type.

Casting an expression from a source data type (specified by the data type of expression) to a target data type (specified as data-type) is subject to the rules set out below. Conversions not specifically permitted below are invalid.

Source and Target Data Types Identical
Use of the CAST function is essentially an assignment, subject to the rules on page 28.
**Character Source**
The value of *expression* must conform to the natural limits imposed on intervals (‘2006’) and on date/time values (‘2007’) by the Gregorian calendar (see page 19); and must in other respects be a valid literal representation of a value of the target data type (‘2018’).

**Character Target**
There must exist a representation of a literal with the value of *expression* in the character set defined by TR (‘2018’).

For conversion to the character class, the length of the converted value must not exceed the length of the target (for CHARACTER) or the maximum string length (for CHARACTER VARYING) (‘2001’).

For conversions to the CHARACTER data type, the value of the source expression is padded on the right with spaces if the length of the converted value is less than the length of the target data type. For conversions to the CHARACTER VARYING data type, this blank padding does not take place.

**Numeric Source**
If the source data type is exact numeric, any conversion to an interval data type must not result in the loss of leading significant digits (‘2015’).

**Date/Time Source**
The CAST function can be used to cast an expression of one date/time data type to another with the following effects:

- When casting a DATE to a TIMESTAMP, the HOUR, MINUTE and SECOND fields of the target are set to zero. Other fields are set to the corresponding values in the source expression.
- When casting a TIME to a TIMESTAMP, the YEAR, MONTH and DAY fields of the target are set to the respective values obtained by evaluating CURRENT_DATE. Other fields are set to the corresponding values in the source expression.
- When casting a TIMESTAMP to a DATE or TIME, the fields of the target are set to the corresponding values in the source expression. Some fields of the source expression are not present in the target.

**Interval Source**
The value of the source expression must be representable as an exact numeric value without the loss of leading significant digits (‘2003’).

## 2.11 HIDDEN COLUMN

In addition to the user-defined column in relations, **Trusted Rubix** also allows users to access one “hidden” column. This hidden column can be used in queries for display and comparisons. It cannot be updated by any SQL construction. It is “hidden” in the sense that when a user uses the “*” construct to select all columns in a relation, the “hidden” column does not get selected. The TR hidden field and the information it stores is listed below.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWLABEL</td>
<td>label</td>
<td>The row label, i.e., the label of the subject inserting the row.</td>
</tr>
</tbody>
</table>
2.12 SEARCH CONDITIONS AND PREDICATES

A predicate is an assertion about a relationship between values. The valid SQL predicates are described in the remainder of this section. These predicates specify or imply comparisons between two values. In any such comparison, the data types of these values must be comparable (as defined on page 31). A search condition is a combination of one or more predicates using the logical operators AND, OR and NOT.

Search conditions, predicates and logical operators in SQL use tri-state logic, dealing with the truth values true, false and unknown. The unknown truth value often results from comparisons with a null value.

2.12.1 Uses of Search Conditions

In query specifications and in sub-queries (see page 48), the WHERE clause uses a search-condition to qualify its selection of rows. The HAVING clause uses a search-condition to qualify its selection of groups. The CHECK clause in the CREATE TABLE statement uses a search-condition to restrict the values that rows of that table may contain. The predicates in a search-condition are evaluated once for each row or group selected. Qualifying rows or groups in a SELECT are those for which the search-condition evaluates to true.

A column-name or expression specified in a search-condition is directly contained in that search-condition if the column-name or expression is not specified within a set-function-reference or a sub-query of that search-condition.

2.12.2 Search Conditions and Large Objects (LOBs)

Large objects (BLOBs and CLOBs) may not be used within a search condition.

2.12.3 Logical Operators

The logical operators AND, OR and NOT operate on the truth values true (T), false (F) and unknown (U), as follows:

<table>
<thead>
<tr>
<th>AND</th>
<th>T</th>
<th>F</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>F</td>
<td>U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>T</th>
<th>F</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th>T</th>
<th>F</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

The order of precedence among the logical operators is NOT (highest), followed by AND, followed by OR. The order of evaluation at the same precedence level is from left to right. The application may use parentheses to change this order since, when used in a search-condition, they determine the binding of predicates to logical operators.

2.12.4 Comparison Predicate

A comparison predicate compares two values and has the form:

expression-1 comparison-operator {expression-2 | (subquery)}

where comparison-operator may be any of the following:

= equal to
Within the context of a comparison predicate, a *sub-query* must result in either a single value or an empty set (‘21000’). If the result of the *sub-query* is an empty set, the result of the predicate is unknown. If the result of the *sub-query* is a single value, the same rules apply as for expression-2.

If either `expression-1` or `expression-2` evaluates to the null value, the result of the predicate is unknown. Otherwise, the result is true or false, depending on the outcome of the comparison.

### 2.12.5 Quantified Comparison Predicate

A quantified comparison predicate compares a value with a number of derived values and has the form:

```
expression comparison-operator ALL | ANY | SOME (sub-query)
```

where `comparison-operator` is as described above.

#### 2.12.6 If ALL is Specified

The result of the predicate is true if the `sub-query` results in an empty set or if the comparison is true for every value returned by the `sub-query`. The result of the predicate is false if the comparison is false for at least one value returned by the `sub-query`. Otherwise, the result of the predicate is unknown.

#### 2.12.7 If SOME or ANY is Specified

The result of the predicate is true if the comparison is true for at least one value returned by the `sub-query`. The result of the predicate is false if the `sub-query` results in an empty set or if the comparison is false for every value returned by the `sub-query`. Otherwise, the result of the predicate is unknown.

#### 2.12.8 BETWEEN Predicate

A BETWEEN predicate tests whether a value is within a range of values and has the form:

```
expression-1 NOT BETWEEN expression-2 AND expression-3
```

The predicate (without NOT) is equivalent to:

```
expression-1 >= expression-2 AND expression-1 <= expression-3
```

Using the keyword NOT negates the result in the manner of the NOT logical operator.

#### 2.12.9 IN Predicate

An IN predicate compares a value with a list of values or with a number of derived values and has the form:

```
expression NOT IN ( (value [ , value] . . .) | (sub-query) )
```
where value is a dynamic-parameter, a literal or a host-variable-reference.  

The operator IN is equivalent to the operator = ANY. Thus, when NOT is not specified, the result of the predicate is true if the implied equality comparison is true for at least one specified or derived comparison value; false if the implied equality comparison is false for every specified or derived comparison value or if the sub-query results in an empty set; and unknown otherwise. Using the keyword NOT negates the result in the manner of the NOT logical operator.

### 2.12.10 LIKE Predicate

A LIKE predicate compares a column value with a pattern and has the form:

```sql
    column-name [ NOT ] LIKE pattern-value
    [ ESCAPE escape-character]
```

where `pattern-value` and `escape-character` are each a character-string-literal, a host-variable-reference, or the keyword USER.

The column `column-name` must reference a character string column. If `host-variable-reference` is specified, it must reference a character string variable.

If `escape-character`, `pattern-value`, or the value of the column referenced by `column-name` is the null value, the result of the predicate is unknown. If NOT is not specified, the result of the predicate is true or false depending on whether or not the value of the column referenced by `column-name` conforms to the specified pattern. Using the keyword NOT negates the result in the manner of the NOT logical operator.

#### Pattern Syntax

Within the character string represented by `pattern-value`, characters are interpreted as follows:

- The underscore character (`_`) stands for any single character.
- The percent character (`%`) stands for any sequence of zero or more characters.
- All other characters stand for themselves.

For example, LIKE `_%A_%` is true for any column value that contains the character A. LIKE `B ___` is true for any column value that is three characters long and starts with the character B.

#### Escape Character

The ESCAPE clause introduces a single character as the escape character for `pattern-value`. The escape character is not treated as part of the pattern, and causes the following character in `pattern-value` to be taken literally (without the special effects for `_` and `%` described above). For example, LIKE `e%%` ESCAPE `e` is true for any column value that begins with a percent sign.

To include the `escape-character` literally within `pattern-value`, it must appear twice in succession.

#### Diagnostics

The length of `escape-character` must be exactly 1 character (`'2019'`).

The character following an occurrence of `escape-character` in `pattern-value` must be a character whose effect can be defeated by `escape-character`; namely, the underscore character, the percent character, or another `escape-character` (`'2025'`).

---

5 Any character-string pseudo-literal (for example, USER) is also valid here, but not useful.
2.12.11 NULL Predicate

A NULL predicate compares a value against the null value and has the form:

\[ column\-NAME \text{ IS [ NOT] NULL} \]

The result of the IS NULL predicate is true if the value of the column referenced by \textit{column-name} is the null value, and false otherwise. Using the keyword NOT negates the result in the manner of the NOT logical operator.

This predicate is the only way to test for the presence or absence of null values. An application cannot identify null values by comparing them with an embedded host variable whose \textit{indicator-variable} identifies it as a null value because the comparison always returns the unknown (U) truth value.

2.12.12 EXISTS Predicate

An EXISTS predicate tests for the existence of a row satisfying some condition and has the form:

\[ \text{EXISTS (sub-query)} \]

The result of the predicate is true if the sub-query does not result in an empty set, otherwise the result of the predicate is false.

OVERLAPS Predicate

An OVERLAPS predicate tests two ranges of dates and times to see if the ranges overlap. It has the form:

\[ row\-value\-constructor\-1 \text{ OVERLAPS row\-value\-constructor\-2} \]

Row value constructors are defined on page 50.

The degree of both \textit{row-value-constructors} must be two. Therefore, another way to illustrate the OVERLAPS predicate is:

\[ (a1, b1) \text{ OVERLAPS (a2, b2)} \]

The values of \textit{a1} and \textit{a2} must be date/time values whose data type is comparable (as defined on page 29). Each parenthesized pair of values defines a range of dates or times. The values \textit{a} and \textit{b} in a pair must relate to one another in one of the following ways:

- if \textit{a} and \textit{b} are date/time values whose data type is comparable, then the pair defines the range of dates or times between \textit{a} and \textit{b}, (either of which may be the earlier); or,
- if \textit{a} is a date/time value and \textit{b} is an interval value that could be added to it (as defined on page 29), then the pair defines the range between \textit{a} and \textit{a} + \textit{b}. (This sum is earlier or later than \textit{a} depending on the sign of \textit{b}.)

The result of the OVERLAPS predicate is true if the two ranges have any date or time in common, including either endpoint of the ranges. If the two ranges do not overlap, the result is false. If any of the values is null, then the result is undefined.

2.13 QUERIES

A \textit{query-specification} is the basic data retrieval construct. It derives a table by selecting specific fields from one or more tables subject to specified criteria. Columns of derived tables, defined by \textit{query-}
A sub-query is a limited form of \textit{query-specification} that derives values for use within a predicate.

\section*{2.13.1 Query Specification}

\subsection*{\textbf{FUNCTION}}
Specify a derived table.

\subsection*{\textbf{SYNOPSIS}}
A query-specification has the form:

\begin{verbatim}
SELECT [ ALL | DISTINCT] select-list
    [WHERE search-condition]
    [GROUP BY column-name [ , column-name] . . . ]
    [HAVING search-condition]
    [ORDER BY select-column-ref [ASC | DESC] [ , select-column-ref [ASC | DESC]] . . .]
    [LIMIT limit-value] [OFFSET offset-value]
\end{verbatim}

where \textit{select-list} is defined as:

\begin{verbatim}
* | select-sublist [ , select-sublist]. . .
\end{verbatim}

and \textit{select-sublist} is defined as:

\begin{verbatim}
expression [AS column-identifier] . . . | \{table-name | correlation-name\} . *
\end{verbatim}

where \textit{select-column-ref} is defined as:

the output name of a column in \textit{select-list} | ordinal number of column in \textit{select-list}

\subsection*{\textbf{DESCRIPTION}}
The result of evaluating a query-specification can be explained in terms of a multi-step algorithm. The order of steps in this algorithm follows the mandatory order of the clauses (FROM, WHERE, and so on) of the \texttt{SELECT} statement:

\begin{itemize}
  \item \textbf{Step 1} For each \textit{table-reference} that is a \textit{joined-table}, conceptually join the tables as specified to form a single table, as described on page 46.
  \item \textbf{Step 2} Form a Cartesian product of all the \textit{table-references}, so that each row of the result is a concatenation of one row from each of the tables in the order specified, and there is one row in the result for each combination of rows of the referenced tables.
  \item \textbf{Step 3} Eliminate all rows that do not satisfy the \textit{search-condition} in the \texttt{WHERE} clause.
  \item \textbf{Step 4} Arrange the \textit{resulting} rows into groups.
    \begin{itemize}
      \item If there is no \texttt{GROUP BY} clause, there is a single group and there are no grouping columns.
      \item If there is a \texttt{GROUP BY} clause specifying grouping columns, then form groups so that all rows within each group have the same values for the grouping columns.
      \item If a grouping column contains any null values, these values are considered equal and give rise to only a single group.
    \end{itemize}
  \item \textbf{Step 5} If there is a \texttt{HAVING} clause, eliminate all groups that do not satisfy its \textit{search-condition}. The \textit{expressions} in the \texttt{HAVING} clause must be single-valued per group. Thus each \textit{column-name} directly contained within the \textit{search-condition} must reference a grouping column or be a
correlated reference, and if a sub-query contained in the search-condition contains a correlated reference, it must be to a grouping column or must be specified as the argument of a set function reference.

→ **Step 6** Generate result rows based on the result columns specified by the select-list.

The select-list “*” is equivalent to a sequence of expressions in which each expression is a column-name that references a column resulting from Step 2. The columns are referenced once each, in the order of their ordinal position at the end of Step 2.

If the select-sublist “{table-name | correlation-name}.*” is specified, table-name or correlation-name must denote a table referenced by the FROM clause. The form select-sublist “{table-name | correlation-name}.*” is equivalent to a sequence of expressions in which each expression is a column-name that references a column of the table denoted by table-name or correlation-name. The columns are referenced once each, in the order of their ordinal position in the denoted table.

Each implicit or explicit expression in the select-list represents a result column.

If GROUP BY was specified, each group generates one result row and the result columns must be single-valued per group. Therefore, any column referenced by the select-list must either be a grouping column or be referenced within the argument of a set-function-reference. If there are no groups, the result of the query-specification is an empty set.

All result columns have names, derived as follows:

↑ If an AS clause is specified for the column, then its column-identifier specifies the name of the result column.

↑ Otherwise, if expression for the column consists only of a column-name, then the column-name specifies the name of the result column.

↑ Otherwise, the name of the result column is un-defined, except that it is guaranteed to be different from the name of any column, other than itself, of a table referenced by a table-reference in the SQL statement. If GROUP BY was not specified, each row generates one result row unless a result column is derived from a set function, in which case the query-specification results in only a single row and any column referenced by the select-list must be referenced within the argument of a set-function-reference.

→ **Step 7** In case of SELECT DISTINCT, eliminate duplicate rows from the result.

→ **Step 8** In case of ORDER BY, sort rows according to specified ordering.

→ **Step 9** If LIMIT is specified, at most limit-value rows from the result set are returned. If OFFSET is specified, the first offset-value rows are not returned.

**Updatability**

A query-specification is updatable if and only if the following conditions are satisfied:

→ The FROM clause contains a single table-reference, either to a base table or to a derived table that is updatable.

→ Neither the GROUP BY clause nor the HAVING clause is present.

→ DISTINCT is not specified.
→ All the result columns are derived from column-names and no column-name appears more than once. That is, either the select-list contains no explicit expressions or every explicit expression it contains consists of a single distinct column-name.

→ If there is a WHERE clause and it contains a sub-query, then the sub-query must not reference any table referenced in the FROM clause (including indirect references).

2.13.2 Joined Tables

The JOIN syntax provides methods of combining information in tables. Varieties of combinations, defined below, are the natural join, the inner join, the left outer join, and the right outer join. (Specifying two or more tables in the FROM clause of a query specification also conceptually joins the tables, in the same manner as an inner join.)

↓ FUNCTION

Specify combinations of tables within a query-specification.

↓ SYNOPSIS

A joined-table is defined as:

qualified-join | (joined-table)

and qualified-join is defined as:

table-reference-1

[CROSS] | [[ NATURAL ] [ INNER ] [ LEFT [ OUTER ] ] [ RIGHT [ OUTER ] ] [ FULL [ OUTER ]]]

JOIN table-reference-2

{ ON search-condition | USING (column-name \ [ , column-name … ] ) }

↓ DESCRIPTION

Every use of qualified-join (that is, every case of combining two tables using the JOIN operator) must specify exactly one of NATURAL, ON, and USING to indicate one of the following operations:

→ NATURAL

NATURAL JOIN produces a non-null table when the two tables being joined have common columns (columns with the same name). The operation is meaningful when common columns contain identical information in the case where the respective rows are related. For example, two tables may contain different sets of information on persons, but a common column called NAME indicates in both tables the person to which the information in that row pertains. In this example, if the NAME column contains the same value in rows of two tables, it means those two rows are related.

The result set of a NATURAL JOIN contains one row for each case where all common columns in the two tables are equal. The result set contains all the columns from both tables, except that the matching common columns appear only once in the result set.

Performing a NATURAL JOIN on the two tables of personal information discussed in the example above produces a result set with one composite row for each person, based on the value of the common NAME column.

Every common column in the two tables must have a data type that permits the values in the two tables to be compared.

→ USING
USING introduces a list of column-names. The operation is conceptually the same as NATURAL JOIN, but JOIN USING takes the specified columns, instead of all common columns, as the basis for joining the rows.

Specifying the columns explicitly instead of using the entire set of common columns is useful when both tables contain common columns (for example, a REMARKS column) which might not be identical even though the rows are related. Specifying the columns explicitly also guards against undesired rejection of rows in cases where columns have identical names by accident.

Every specified column in the two tables must have a data type that permits the values in the two tables to be compared.

→ **ON**

ON introduces a search-condition. Form a Cartesian product of the two tables (defined on page 44), the result set contains only those rows for which the search-condition is true. The search-condition cannot reference common columns unless it qualifies them, such as by table name.

**OUTER JOIN**
The above discussion describes the effect if inner is specified, or if neither inner, left outer, or right outer is specified. (The keyword outer is always optional.) The inner join includes rows in the result set based on the respective criteria listed above. left outer join includes in the result table additional rows from the table specified to the left of the word join (that is, table-reference-1). right outer join includes additional rows from the table to the right (table-reference-2):

→ In LEFT OUTER JOIN, rows from table-reference-1 that were not the basis for any row of the INNER JOIN appear at the end of the result set. All information from table-reference-1 thus is present in at least one row of the result set.

→ In RIGHT OUTER JOIN, rows from table-reference-2 that were not the basis for any row of the OUTER JOIN appear at the end of the result set. All information from table-reference-2 thus is present in at least one row of the result set.

Both OUTER JOINS add rows to the result set based on a row of only one of the joined tables. In each row, the columns that are defined only in the other table contain the null value. These rows are combined with the result of the INNER JOIN as in the UNION ALL operator discussed on page 47.

### 2.13.3 Query Expressions

A query-expression is one or a combination of query-specifications defined as follows:

\[
\text{query-expression} \quad \text{UNION} \quad [\text{ALL}] \quad \text{query-expression} \\
\quad \mid (\text{query-expression}) \\
\quad \mid \text{query-specification}
\]

Parentheses control the order of evaluation of any UNION operators.

### 2.13.4 UNION of Tables

The UNION or UNION ALL operator forms a union of two derived tables specified as the operands. The union is a derived table that contains all the rows of the two operands. In the UNION ALL form, duplicate rows are retained; if ALL does not follow UNION, then duplicate rows are eliminated from the union. The two operands must have the same number of columns.

Corresponding columns in a UNION or UNION ALL operation must have compatible data types. The
following list shows all valid combinations of operand columns and, for each combination, shows the data type of the resulting column:

→ If the data type of one operand is a character string, then the data type of the other operand must be character string.

   If either of the operand data types is variable-length character string, then the result data type is variable-length character string whose maximum length is the largest maximum length of either operand.

   Otherwise, the result data type is fixed-length character string whose length is the largest length of either of the operands.

→ If both of the operand data types are exact numeric, then the result data type is exact numeric, with trusted rubix defined precision, and whose scale is the largest scale of either of the operands.

→ If the data type of one operand is approximate numeric, then the data type of the other operand must be numeric. The result data type is approximate numeric with TR defined precision.

→ If the data type of one operand is date/time, then the data type of the other operand must be date/time and the subtypes must be identical. The result data type is date/time with this same subtype.

→ If the data type of one operand is interval, then the data type of the other operand must also be interval. Either both operands’ interval precisions must specify YEAR or MONTH, or both operands’ interval precisions must specify DAY, HOUR, MINUTE or SECOND. The result data type is interval and its interval precision is selected to accommodate the date/time fields of either operand. For example, the union of INTERVAL DAY with INTERVAL MINUTE TO SECOND produces a result of data type INTERVAL DAY TO SECOND.

The result column allows null values if and only if either operand column allows them.

When these corresponding columns have the same name, the result column takes that name. Otherwise, the name of the result column is undefined, except that it is guaranteed to be different from the name of any column, other than itself, of a table referenced by a table-reference in the SQL statement.

UNION may be specified several times; the application can use parentheses to indicate the order in which the union operations are performed. If two or more UNION operations are specified without parentheses, the operations are performed from left to right. The result set of the cursor is the result of the final UNION.

**Updatability**

A query-expression that contains the keyword UNION is not updatable. A query-expression that is a single query-specification is updatable depending on whether the query-specification is updatable. This is defined on page 45.

### 2.13.5 Sub-queries

**FUNCTION**

Provide a multi-set of values or rows within a predicate.

**SYNOPSIS**

A sub-query has the same syntax as a query-specification (see page 44) but is enclosed in parentheses.
DESCRIPTION

A sub-query is a limited form of query-specification. A sub-query can be used only on the right hand side of a comparison predicate, quantified predicate, or IN predicate, or as the subject of an EXISTS predicate. A sub-query can be nested in expressions within another sub-query. The result of the sub-query is substituted into the predicate of the outer query.

A sub-query used in a context where each result row that is compared to a scalar value (in the quantified comparison and IN predicates) must have a single result column. A sub-query used in a comparison predicate must have a single result row and column.

A sub-query in an EXISTS predicate is used to derive a multi-set of rows, but it is the existence of one or more result rows, rather than the values in these rows, that is significant; therefore, in this case, the result columns can only be specified using a single asterisk (the normal case), a single asterisk qualified by a table-name or correlation-name, or a single column-name.

A sub-query is subject to the same restriction as for query-specification regarding multiple use of the keyword DISTINCT.

DIAGNOSTICS

When a sub-query is used in a position that requires at most a single value, its evaluation must produce a result of no more than one row (’21000’).

2.13.6 Correlation

The column-name element has been defined as:

\[ \{ \text{table-name} | \text{correlation-name} \}. \text{unqualified-column-name} \]

A qualifier is added to an unqualified-column-name to show which table, and which reference to that table, the column-name relates to. The reasons for using a qualifier are:

1. to override the scope rules, given below, that assume a default table reference; and
2. to differentiate among simultaneous references to the same table in cases described below.

In addition, labeling references to tables with correlation-names may increase the readability of the application program even in cases where their use is not strictly required.

In any table-reference in a FROM clause, the table-name can be followed by a correlation-name. If a correlation-name appears, it is the name that specifies this table reference. If it does not appear, the table-name itself specifies this table reference. This name of the table reference is the name that can be used to qualify an unqualified-column-name.

2.13.7 Scope of Table Reference Names

The scope of a name that denotes a table reference is the entire innermost sub-query, query-specification or SELECT statement that contains the FROM clause.

Within a FROM clause, table-names that denote table references must be unique, and correlation-names must be unique and distinct from the unqualified table-names that denote table references.

The scope of a table-name in an UPDATE or searched DELETE statement is the entire statement.
If a column-name contains a qualifier, the column-name must be within the scope of that qualifier. If the column-name is within the scope of more than one qualifier with the specified name, the name refers to the qualifier with the most local scope. The specified table reference must be to a table that contains a column with the specified unqualified-column-name.

If a column-name does not include a qualifier, the column-name must be within the scope of, and is assumed to be qualified by, a qualifier whose associated table contains a column with the specified unqualified-column-name. If there is more than one possible qualifier, there must be only one with the most local scope and that one is assumed.

If the table reference represents a JOIN operand, then the scope is the join expression that immediately contains that JOIN excluding any select-expression or join expression that is nested anywhere within the original join expression in which another correlation_name is introduced with the same name.

2.13.8 Multiple References to the Same Table

There are two notable cases:

→ A FROM clause references the same table twice.
  A typical example of this is joining a table to itself (joining rows of the same table). Since there are two simultaneous distinct references to the same table, two distinct, explicit qualifiers are needed to distinguish between the two references. Thus, at least one of the table-references must contain a correlation-name.

→ A sub-query and an outer query reference the same table.
  If the sub-query refers to both references - the table reference within the sub-query and the reference to the same table in the outer query - then the references are simultaneous, and qualifiers must be used to distinguish between the two occurrences. Thus, at least one of the table-references must contain a correlation-name.

In this example, only the reference in the sub-query to the occurrence in the outer query needs an explicit qualifier for unique identification. The reference to the occurrence within the sub-query could be implicit because of the rule that the reference with the most local scope is assumed.

If the value of a sub-query depends on the value of a column in a row of an outer query, the sub-query is a “correlated sub-query” and the reference to the column of the outer query is a “correlated reference”. A correlated sub-query has to be evaluated once for each row of the outer query.

A correlated reference need not be explicitly qualified unless required for uniqueness of reference. However, it may improve clarity to associate a correlation-name with the relevant table-name in the FROM clause of the outer query, and to qualify the correlated reference with this name.

If the sub-query refers only to its own occurrence, then this reference and the reference in the outer query are not simultaneous and do not need to be distinguished. This case is not a correlated reference.

2.13.9 Row Value Constructor

FUNCTION
Specify the value of each column of a table row.
SYNOPSIS
A row-value-constructor has the form:

(column-spec [ , column-spec ] ...)

where each column-spec can be an expression, NULL, or DEFAULT.

DESCRIPTION
There are two cases in Trusted Rubix SQL syntax where the application uses a row-value-constructor to specify a complete row of a table:

→ One form of the INSERT statement, which inserts a row into a table, lets the application specify values of each column in the new row.

→ The OVERLAPS predicate, which compares two ranges of dates or times, deals with each range as a row containing two columns of type date/time or interval.

These valid contexts for a row-value-constructor impose restrictions on the number and values of the column-specs.

The degree of a row-value-constructor is the number of column-specs it contains.
3. Data Definition Statements

3.1 DATA DEFINITION STATEMENTS

A data definition statement modifies the metadata of the database. Data definition statements define databases, catalogs, schemas, tables, indexes, columns, and views; data manipulation statements store and retrieve data from them.

For the data manipulation statements to work correctly, the definitions of the tables they reference must exist before the data manipulation statements are executed. For example, the CREATE TABLE or CREATE VIEW statements that define specific tables must have been executed before executing a data manipulation statement that inserts data into those tables.

Applications should not mix data definition statements and data manipulation statements within transactions.

**NOTE**

In addition to the Multilevel Security (MLS) Mandatory Access Control (MAC) policy, **TR** supports the Type Enforcement (TE) MAC policy of SELinux and a proprietary Attribute Based Accessed Control (ABAC) MAC policy of the Security Policy Manager (SPM). In general, all configured MAC policies must permit an operation for it to succeed. For more information on TE and SELinux please see the Trusted RUBIX SELinux Guide and for more information on ABAC and the SPM please see the Trusted RUBIX Security Policy Manager Reference Guide and Tutorial.

3.1.1 General Diagnostics

Only the creator of a schema can alter the definition of the objects within the schema. Therefore, the current user must be the creator of the schema that contains the object referenced by ALTER TABLE, CREATE INDEX, DROP INDEX, DROP TABLE or DROP VIEW; or the creator of the schema dropped by DROP SCHEMA (‘42000’). (The CREATE SCHEMA statement lets a user create and populate a schema to be used by another user.)

3.1.2 ALTER TABLE

**FUNCTION**

Add or destroy a column or constraint in an existing base table.

**SYNOPSIS**

```sql
ALTER TABLE [catalog-name.][schema-name.]base-table-name
    { { ADD [COLUMN] column-definition } |
    { DROP [COLUMN] identifier { CASCADE | RESTRICT } } |
    { ADD CONSTRAINT base-table.constraint-definition }
    { DROP CONSTRAINT constraint { CASCADE | RESTRICT } } }
```

where base-table.constraint-definition is defined as:
UNIQUE (unqualified-column-name [ , unqualified-column-name] . . .)
| CHECK (search-condition)
| FOREIGN KEY referencing-columns REFERENCES [catalog-name.[schema-name.]base-table-name-
  2 [(referenced-columns)] [ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT} ]

↓ DESCRIPTION

The ALTER TABLE statement adds a new column to, drops an existing column from, adds a new constraint to, or drops an existing user named constraint from base-table-name.¹

Adding a Column

The ADD clause specifies the column name and data type of a new column, and may also specify a default value and column constraints, using the same syntax as in the CREATE TABLE statement. The new column appears at the right-hand side of the table, so that a new value is added at the end of every existing row of the table. The column’s initial value is the default value. (If ALTER TABLE does not specify a default value, the default value is the null value.)

Adding a column to a table has no effect on any existing query-specification contained in a view definition, since any implicit column references are resolved when the view definition is originally executed.

Dropping a Column

In an ALTER TABLE statement that contains the DROP clause, the application must use the keyword CASCADE or RESTRICT. Doing so specifies the effect if the dropped column is referenced in the query-specification of a view definition or in the search-condition of a constraint:

→ CASCADE
  All such views and constraints are also dropped.

→ RESTRICT
  The operation fails if there are any such views or constraints (RD004).

A column cannot be dropped when it is the only one in the table.

Adding a Constraint

The ADD constraint definition permits a new constraint to be defined for an existing base table (the new constraint does not replace any existing constraints for the base table in question, but is logically ANDed on to them.

Dropping a Constraint

DROP permits an existing base table constraint to be removed. Although CASCADE or RESTRICT must be specified, they only have meaning if the constraint is a candidate key definition. An attempt to drop a constraint key definition will fail if any foreign key references that candidate key, unless CASCADE is specified. In that case, all such foreign key definitions are also dropped.

Table Constraints

A base-table-constraint-definition may contain UNIQUE, CHECK or FOREIGN KEY:

→ If UNIQUE appears, then each row is constrained to contain a different value in the specified column (or specified columns taken together).

¹ A former alternative syntax for the ALTER statement that specified several columns in parentheses is no longer supported. To add more than one column to an existing base table use multiple ALTER TABLE statements.
If CHECK appears, then each row is constrained to contain values that satisfy the search-condition.

If FOREIGN KEY appears, then each row is constrained in that the referencing-columns must contain values found in base-table-name-2. The valid values are contained in the referenced-columns or, if this is omitted, in the primary key of base-table-name-2.

Each unqualified-column-name specified in UNIQUE or referencing-columns must identify a column of base-table-name-1. In each case, if a list is specified, it must not identify the same column more than once.

### ON DELETE Actions

As described above, specifying FOREIGN KEY for one or more referencing-columns in the table being defined restricts their valid values to those existing in certain columns of base-table-name-2. If those values in base-table-name-2 were deleted, all referencing-columns that contained the deleted values would immediately violate the FOREIGN KEY constraint.2

When creating a table with a FOREIGN KEY, the application can specify what happens in this case by using an ON DELETE clause. This specifies the effect of certain positioned DELETE or searched DELETE statements applied to base-table-name-2. ON DELETE is followed by exactly one of the following:

- **NO ACTION**
  The positioned DELETE or searched DELETE fails with 23000 and makes no change to the database.

- **CASCADE**
  All rows where any referencing-columns are affected are also deleted. If some columns of a deleted row are themselves the object of FOREIGN KEY references, the deletion of the row may cause additional rows to be deleted, possibly including rows in other tables.

- **SET NULL**
  Affected referencing columns are set to the null value.3

- **SET DEFAULT**
  Each affected column of referencing-columns is set to its respective default value.

If the ALTER TABLE statement contains a FOREIGN KEY clause that does not contain an ON DELETE clause, ON DELETE NO ACTION is assumed.

### Column Constraints

A column-constraint definition is a distinct syntactic form that achieves the same purpose as a table-constraint-definition but applies to a single column. This form may contain CHECK, NOT NULL, REFERENCES or UNIQUE.

- The NOT NULL constraint constrains the column to not contain null values. NOT NULL may not occur more than once in a column-definition.

- The UNIQUE constraints have the same effect as described above for table constraints. In the case

---

2 The table being defined could also violate this constraint if the values in base-table-name-2 were modified. The International Standard includes an ON UPDATE clause, which works in the same manner as the ON DELETE clause specified here, to cover this case. X/Open does not require support of the ON UPDATE clause.

3 Doing so may itself violate an integrity constraint, such as NOT NULL or CHECK constraint. In this case, the positioned DELETE or searched DELETE fails as described for the NO ACTION case.
of column constraints, the application cannot specify both PRIMARY KEY and UNIQUE.\textsuperscript{4}

→ The REFERENCES column constraint is equivalent to a FOREIGN KEY table constraint. It constrains each row to contain only values contained in a specified base table.

→ If CHECK appears, then each row is constrained to contain a value that satisfies the search-condition. The search-condition may only contain literals and references to the column being defined.

The DEFAULT clause specifies a value that is given to the column if an INSERT statement does not furnish a value. All columns have a default-value; if the application does not use the DEFAULT clause, the column’s default value is the null value.

The specified or implied default value can be NULL, even if the column is constrained by NOT NULL, but it is then a constraint violation to insert a row without overriding the default value.

Any specified default value is subject to the following rules:

→ If the data type of the column is character string, default-value must be a character string literal or pseudo-literal and its length must not exceed the column’s length or maximum length; blanks are appended to a literal if necessary to make it the length of a fixed-length column.

→ If default value is a pseudo-literal such as CURRENT_TIMESTAMP or CURRENT_USER, the columns default value reflects the value the pseudo-literal has at the time of the particular insertion, not the value it had at the time of the CREATE TABLE statement.

→ If the data type of the column is exact numeric, default-value must be a numeric literal for which there is a valid representation in the column that does not lose any significant digits.

→ If the data type of the column is approximate numeric, default value must be a numeric literal.

If the check clause appears, the search-condition may not contain a set function reference, a subquery, current_date, current_time or current_timestamp.

FOREIGN KEY lets one table reference itself or another table. The references keyword can appear in a column-constraint-definition to apply to one column or to a set of columns taken together. Referencing and referenced columns must be equal in number and in data type. The sequence of referenced-columns must be the same as that in a unique or primary key list in the create table statement that defines base-table-name-2.

\textbf{↓ APPLICATION USAGE}

The ALTER TABLE ADD COLUMN statement, which sets the new column to its default value in all rows of the table, fails due to a constraint violation in the following cases:

→ If the new column is constrained to be NOT NULL, the default is null and the table is not empty, then an immediate violation of the NOT NULL constraint occurs because the statement would extend all existing rows to contain the new column with a null value.

→ If the new column is constrained to be UNIQUE, the default value is non-null and the table has two or more rows, then an immediate violation of the UNIQUE constraint occurs because the statement

\textsuperscript{4} UNIQue columns may not specify NOT NULL, and PRIMARY KEY columns imply NOT NULL. In X/Open SQL, a UNIQUE index (which is functionally similar to a unique constraint) may be created including columns which are defined to allow null values. This means X/Open-compliant implementations cannot implement unique constraints using UNIQUE indexes unless they enforce the NOT NULL constraint separately.
would extend all existing rows to contain the new column with the same value.

**SECURITY**

MAC

Session sensitivity label $\geq$ parent catalog sensitivity label (3D000).

Session sensitivity label $\geq$ parent schema sensitivity label (3F000)

Session sensitivity label = table sensitivity label (OA514, if session sensitivity label $>$ table sensitivity label; RR005, otherwise.)

Session sensitivity label = table sensitivity label of any table referenced by a foreign key (RI010).

For DROP, the dropped column must not have any views that reference it where view sensitivity label $>$ session sensitivity label (RD004)

DAC

The user must have the `rubix.dac.rxcreate.dbname` authorization for ADD or the `rubix.dac.rxdrop.dbname` authorization for DROP or have all of the following privileges:

- EXEC on database (RA001)
- EXEC on catalog (RA008)
- EXEC, WRITE on schema (RA002)
- REFERENCES(I) on any column referenced by a foreign key (RI012)

AUDIT (sql_rel_alter) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, relation label, catalog name, schema name, relation name, relation structure, operation type, [new relation structure]

---

**NOTE**

To ensure error-free operation, make sure that all cursors are closed prior to issuing an ALTER command.

### 3.1.3 CREATE CATALOG

**FUNCTION**

Create a catalog.

**SYNOPSIS**

```
CREATE CATALOG catalog-name [catalog-element [catalog-element] . . .]
```

where each `catalog-element` is a CREATE SCHEMA command.

**DESCRIPTION**

The CREATE CATALOG statement creates a catalog, a unit that can contain schemas.

*Object-qualifier* is the name of the catalog being created. The system creates a default catalog called “default_catalog” in which all user and system specified objects are placed.

- By finishing the CREATE CATALOG statement with one or more `catalog-elements`, the user can specify the initial contents of the catalog. This can be the CREATE SCHEMA statement.

A CREATE CATALOG statement need not specify any `catalog-element`. Objects can be created by data
definition statements executed subsequently.

**SECURITY**

MAC  Session sensitivity label >= database sensitivity label (08001).

All objects created are labeled with the session sensitivity label.

DAC  The user must have the *rubix.dac.rxcreate.dbname* authorization or have all of the following privileges:

  - EXEC, WRITE on database (RA001)

AUDIT (sql_catalog_create) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, catalog label, catalog name.

### 3.1.4 CREATE DATABASE

**FUNCTION**

Create a database

**SYNOPSIS**

CREATE DATABASE *database-name* [[no logging]]

 [CREATE CATALOG *catalog-name*]
 [CREATE SCHEMA *schema-name*]

**DESCRIPTION**

The CREATE DATABASE statement creates a new database on the server. The operation will fail if a database with the same name already exists on the same host. When a database is created, it can be created with or without logging. The default is to create the database with logging. It is recommended that users create important databases with logging enabled. The default storage locations for logging files, table data, and LOB data may be specified in the *rxconfig* file prior to creating the database. To move storage directories after database creation see the *rxdb* and *rxlogs* commands. For more information please see the [Trusted Rubix Trusted Facility Manual](#).

All instantiations of Trusted Rubix SQL must be started while connected to a database. The CREATE DATABASE statement can only be used when connected to the *master* database.

**SECURITY**

MAC  The database is sensitivity labeled with the session sensitivity label. All created objects (such as *default_schema*) are created at this sensitivity label as well.

DAC  There are no DAC requirements to create a database. By default, only the creator can connect to the created database. To allow other users access requires using the GRANT SQL statement to effect DAC changes.

AUDIT (sql_db_create) event name, user ID, group ID, currently open database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, new database label, new database name
3.1.5 CREATE INDEX

**FUNCTION**
Create an index on a base table. Ensures uniqueness of the values in the specified columns if the index is UNIQUE, or is the PRIMARY KEY.

**SYNOPSIS**

```sql
CREATE [UNIQUE] INDEX index-name ON
[catalog-name.] [schema-name.]
base-table-name (unqualified-column-name
```

It is not an error that a referenced object does not exist if either of the following rules applies:

→ The reference occurs in a CREATE statement that is bringing the object into existence; or,
→ The reference occurs in a statement that is enclosed in a CREATE SCHEMA statement and the referenced object is defined somewhere in the same CREATE SCHEMA statement.

**DESCRIPTION**
The CREATE INDEX statement creates an index named `index-name` on the existing base table `base-table-name`. The index is created in the schema in which the base table resides.

The index key is constructed of columns from the specified base table in the given order of significance. When `ASC` is specified, the order of the referenced column is ascending. When `DESC` is specified, the order of the referenced column is descending. The default order is ascending.

`UNIQUE` indicates that at most one row is allowed in the table for each combination of values in the specified columns. For the purpose of this clause two null values are considered equal. The constraint is enforced when rows are inserted or updated and checked during execution of the CREATE INDEX statement.

A composite index (also called a concatenated index) is an index that you can create on multiple columns in a table. Columns in a composite index can appear in any order and need not be adjacent in the table.

**SECURITY**

**MAC**

Session sensitivity label >= parent catalog sensitivity label (3D000).

Session sensitivity label >= parent schema sensitivity label (3F000)

Session sensitivity label = table sensitivity label (OA514, if session sensitivity label > table sensitivity label; RR005, otherwise).

All objects created are labeled with the session sensitivity label.

**DAC**
The user must have the `rubix.dac.rxcreate.dbname` authorization or have all of the following privileges:

→ EXEC on database (RA001)
→ EXEC on catalog (RA008)
EXEC, WRITE on schema (RA002)

AUDIT (sql_idx_create) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, index label, catalog name, schema name, relation name, index name, index definition

3.1.6 CREATE SCHEMA

FUNCTION
Create a schema.

SYNOPSIS
CREATE SCHEMA [catalog-name.]schema-name
[DEFAULT CHARACTER SET character-set-name] [schema-element] [schema-element . . . ]

where each schema-element is a CREATE TABLE, CREATE VIEW, CREATE INDEX or GRANT statement.

DESCRIPTION
The CREATE SCHEMA statement creates a schema, an object that can contain tables, indexes and views.

If the DEFAULT CHARACTER SET clause is present, then character-set-name is the name of the character set that all data within the schema will use, unless contrary declarations are made on a more local level, such as by the CREATE TABLE statement for specific columns of a table. Currently, Trusted Rubix only supports the ASCII character set which is the default set for the schema if the clause is omitted.

By finishing the CREATE SCHEMA statement with one or more schema-elements, the user can specify the initial contents of the schema. This can be some sequence of the following statements:

→ the CREATE INDEX statement
→ the CREATE TABLE statement
→ the CREATE VIEW statement and,
→ the GRANT statement

The statements can appear in any order; in fact, the definition of a view or index can precede the definition of the underlying base table, and tables can reference columns in tables that appear later in the CREATE SCHEMA statement.

If schema-elements are specified, these operations can fail with diagnostics specified in the respective section. If any error occurs executing a schema-element, then CREATE SCHEMA returns the associated diagnostics and does not create the schema.

A CREATE SCHEMA statement need not specify any schema-element. Objects can be created and privileges granted by data definition statements executed subsequently.

SECURITY
MAC session sensitivity label >= parent catalog sensitivity label (3D000).

All objects created are labeled with the session sensitivity label.
DAC  The user must have the *rubix.dac.rxcreate.dbname* authorization or have all of the following privileges:

↑ EXEC on database (RA001)

↑ EXEC, WRITE on catalog (RA008)

AUDIT (sql_schema_create) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, schema label, catalog name, schema name

### 3.1.7 CREATE TABLE

#### FUNCTION

Create a base table.

#### SYNOPSIS

```
CREATE [temporary-definition] TABLE [catalog-name.] [schema-name.]
    base-table-name-1 (column-element [, column-element] . . . ) [on-commit-behavior]
```

where *temporary-definition* is defined as:

- LOCAL TEMPORARY
- | GLOBAL TEMPORARY

where *on-commit-behavior* is defined by:

- ON COMMIT DELETE ROWS
- | ON COMMIT PRESERVE ROWS

where *column-element* is defined as:

- *column-definition* | *table-constraint-definition*

and *column-definition* is defined as:

- unqualified-column-name data-type [DEFAULT default-value]
- [column-constraint-definition [, column-constraint-definition] . . .]

and *column-constraint-definition* is defined as:

- CHECK (search-condition)
- | NOT NULL
- | PRIMARY KEY
- | REFERENCES [catalog-name.] [schema-name.] base-table-name-2 [ (unqualified-column-name) ]
- | UNIQUE

and *default-value* is defined as:

- literal | NULL

and *table-constraint-definition* is defined as:

- UNIQUE (unqualified-column-name [, unqualified-column-name] . . .)
- | PRIMARY KEY (unqualified-column-name [, unqualified-column-name] . . .)
| CHECK (search-condition) |
| FOREIGN KEY referencing-columns REFERENCES [catalog-name][schema-name].base-table-name-2 [(referenced-columns)] [ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ] |

and referencing-columns and referenced-columns are defined as:

(unqualified-column-name [, unqualified-column-name ] . . .)

and where no two column-definitions begin with the same unqualified-column-name.

**Temporary Table Definition**

An optional temporary-definition may contain:

→ LOCAL TEMPORARY: Unsupported.

→ GLOBAL TEMPORARY: The temporary table will be accessible by SQL operations executed by other database sessions, according to DAC and MAC permissions and transactional consistency. The table is automatically dropped during session termination.

If temporary-definition is defined, an optional on-commit-behavior may contain:

→ ON COMMIT DELETE ROWS: All table rows are automatically deleted during transaction commit. If on-commit-behavior is not specified, ON COMMIT DELETE ROWS is the default behavior or a temporary table.

→ ON COMMIT PRESERVE ROWS: All rows within the table are preserved after the transaction commit.

**Table Constraints**

A table-constraint-definition may contain UNIQUE, PRIMARY KEY, CHECK or FOREIGN KEY:

→ If UNIQUE appears, then each row is constrained to contain a different value in the specified column (or specified columns taken together).

→ PRIMARY KEY has the same meaning as UNIQUE, but also declares that the column names are the ones implicitly referenced in any FOREIGN KEY constraints (see below) that reference base-table-name-1. PRIMARY KEY may not occur more than once in a CREATE TABLE statement. Designating a column as the PRIMARY KEY also implicitly constrains the column to be NOT NULL.

→ If CHECK appears, then each row is constrained to contain values that satisfy the search-condition.

→ If FOREIGN KEY appears, then each row is constrained in that the referencing-columns must contain values found in base-table-name-2. The valid values are contained in the referenced-columns or, if this is omitted, in the primary key of base-table-name-2.

Each unqualified-column-name specified in UNIQUE, PRIMARY KEY or referencing-columns must identify a column of base-table-name-1. In each case, if a list is specified, it must not identify the same column more than once.

Multiple unique (UNIQUE or PRIMARY KEY) clauses cannot apply to the same set of columns.

**ON DELETE Actions**

As described above, specifying FOREIGN KEY for one or more referencing-columns in the table being defined restricts their valid values to those existing in certain columns of a table base-table-name-2. If those values in base-table-name-2 were deleted, all referencing-columns that contained the deleted values
would immediately violate the FOREIGN KEY constraint.\(^5\)

When creating a table with a FOREIGN KEY, the application can specify what happens in this case by using an ON DELETE clause. This specifies the effect of certain positioned DELETE or searched DELETE statements applied to base-table-name-2. ON DELETE is followed by exactly one of the following:

- **NO ACTION**
  The positioned DELETE or searched DELETE fails with '23000' and makes no change to the database.

- **CASCADE**
  All rows where any referencing-columns are affected are also deleted. If some columns of a deleted row are themselves the object of FOREIGN KEY references, the deletion of the row may cause additional rows to be deleted, possibly including rows in other tables.

- **SET NULL**
  Affected referencing columns are set to the null value.\(^6\)

- **SET DEFAULT**
  Each affected column of referencing-columns is set to its respective default value.

If the CREATE TABLE statement contains a FOREIGN KEY clause that does not contain an ON DELETE clause, ON DELETE NO ACTION is assumed.

**Column Constraints**

A column-constraint-definition is a distinct syntactic form that achieves the same purpose as a table-constraint-definition but applies to a single column. This form may contain CHECK, NOT NULL, PRIMARY KEY, REFERENCES or UNIQUE.

- **The NOT NULL constraint** constrains the column to not contain null values. NOT NULL may not occur more than once in a column-definition.

- **The PRIMARY KEY and UNIQUE constraints** have the same effect as described above for table constraints. In the case of column constraints, the application cannot specify both PRIMARY KEY and UNIQUE.\(^7\)

- **The REFERENCES column constraint** is equivalent to a FOREIGN KEY table constraint (see above). It constrains each row to contain only values contained in a specified base table.

- **If CHECK appears**, then each row is constrained to contain a value that satisfies the search-condition. The search-condition may only contain literals and references to the column being defined.

The DEFAULT clause specifies a value that is given to the column if an INSERT statement does not furnish a value. All columns have a default-value; if the application does not use the DEFAULT clause, the column’s default value is null.

---

\(^5\) The table being defined could also violate this constraint if the values in base-table-name-2 were modified. The International Standard includes an ON UPDATE clause, which works in the same manner as the ON DELETE clause specified here, to cover this case. X/Open does not require support of the ON UPDATE clause.

\(^6\) Doing so may itself violate an integrity constraint, such as a NOT NULL or CHECK constraint. In this case, the positioned DELETE or searched DELETE fails as described for the NO ACTION case.

\(^7\) UNIQUE columns may specify NOT NULL, and PRIMARY KEY columns imply NOT NULL. In X/Open SQL, a UNIQUE index (which is functionally similar to a unique constraint) may be created including columns which are defined to allow null values. This means X/Open-compliant implementations cannot implement unique constraints using UNIQUE indexes unless they enforce the NOT NULL constraint separately.
The specified or implied default value can be NULL, even if the column is constrained by NOT NULL, but it is then a constraint violation to insert a row without overriding the default value.

Any specified default value is subject to the following rules:

- If the data type of the column is character string, default-value must be a character string literal or pseudo-literal and its length must not exceed the column’s length or maximum length; blanks are appended to a literal if necessary to make it the length of a fixed-length column.

- If default-value is a pseudo-literal such as CURRENT_TIMESTAMP or CURRENT_USER, the column’s default value reflects the value the pseudo-literal has at the time of the particular insertion, not the value it had at the time of the CREATE TABLE statement.

- If the data type of the column is exact numeric, default-value must be a numeric literal for which there is a valid representation in the column that does not lose any significant digits.

- If the data type of the column is approximate numeric, default-value must be a numeric literal.

In the CHECK clause, the search-condition may not contain a set function reference, a subquery, CURRENT_DATE, CURRENT_TIME or CURRENT_TIMESTAMP.

FOREIGN KEY lets one table reference itself or another table. The REFERENCES keyword can appear in a column-constraint-definition to apply to the column being defined, or in a table-constraint-definition to apply to one column or to a set of columns taken together. Referencing and referenced columns must be equal in number and in data type. The sequence of referenced-columns must be the same as that in a UNIQUE or PRIMARY KEY list in the CREATE TABLE statement that defines base-table-name-2.

**SECURITY**

MAC  
Session sensitivity label >= parent catalog sensitivity label (3D000).  
Session sensitivity label >= parent schema sensitivity label (3F000).  
Session sensitivity label = table sensitivity label of any table referenced by a foreign key (RI010).  
Foreign key constraints are enforced only when the row sensitivity label equals the row sensitivity label of the referenced column’s row. For more information see the INSERT command.  
ON DELETE behavior for foreign key constraints, that may alter the contents of other rows (SET NULL, SET DEFAULT, CASCADE), only have effect when the row sensitivity label equals the row sensitivity label of the referenced column’s row. For more information see the DELETE command.  
All objects created are labeled with the session sensitivity label.

DAC  
The user must have the rubix.dac.rxcreate.dbname authorization or have all of the following privileges:

- EXEC on database (RA001)
- EXEC on catalog where the table is created (RA008)
- EXEC, WRITE on schema where the table is created (RA002)
- REFERENCES on all columns of tables referenced by foreign keys (RI012); for these tables the user must have EXEC on their catalog(s) (RA008) and EXEC on their schema(s) (RA002)

AUDIT  
(sql_rel_create) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, relation label, catalog name, schema name, relation name, relation structure
3.1.8 CREATE VIEW

**FUNCTION**
Create a viewed table.

**SYNOPSIS**
```
CREATE VIEW [catalog-name.][schema-name.]viewed-table-name [ (unqualified-column-name
[ , unqualified-column-name ] . . . ) ] AS query-expression [WITH CHECK OPTION]
```

where a given `unqualified-column-name` appears only once.

**DESCRIPTION**
The `CREATE VIEW` statement creates a viewed table `viewed-table-name` that is the result of evaluating `query-specification`.

The `query-specification` defines a derived table. The resulting viewed table is updatable if this derived table is updatable. The data types of the viewed table's columns are the same as those of the corresponding columns of the derived table. The viewed table's column names are the names of the corresponding columns of the derived table, unless the `CREATE VIEW` statement contains `unqualified-column-names`. (If the derived table contains any columns with undefined or duplicate names, then the `CREATE VIEW` statement must contain `unqualified-column-names`.) Performing an `INSERT`, `UPDATE`, `DELETE`, or `SELECT` command on a defined view implicitly executes the view's `query-specification` definition statement.

The creator of the view is given an initial set of DAC privileges that is calculated from those he holds on the underlying table(s). If the viewed table is not updatable then the creator will not be given `INSERT`, `UPDATE`, or `DELETE` privileges on any columns regardless of what the user holds on the underlying table(s). Noting that exception, the creator is given the `INSERT`, `UPDATE`, `DELETE` and `SELECT` on a column in the view only if the user has that column privilege on the underlying table. If the user is given any column privilege on the view he is always given WITH GRANT OPTION on that privilege and the GRANTNULL privilege on that column.

If any `query-specification` in `query-expression` contains a GROUP BY clause, `viewed-table-name` identifies a grouped view.

No `query-specification` in `query-expression` may reference any host variables, contain any dynamic parameters, or reference the viewed table being created.

**With Check Option**
The search conditions in the `query-specification` in the `CREATE VIEW` statement, combined with those in any underlying `CREATE VIEW` statement, are called the **view criteria**.

If WITH CHECK OPTION is specified, then any insert or update performed on the viewed table, or on any table derived from the viewed table, fails (setting SQLSTATE to `44000`) if the resulting row violates the view criteria. That is, the insert or update fails if the effect of the change would be that the resulting row is no longer present in `viewed-table-name`.

If WITH CHECK OPTION is not specified, then `query-expression` determines the initial definition of the viewed table, but does not regulate the changes that can be made to the viewed table. In this case, the application may insert or update rows in the viewed table that it is subsequently unable to retrieve through that viewed table.
## SECURITY

### MAC

- session sensitivity label $\geq$ parent catalog sensitivity label (3D000).
- session sensitivity label $\geq$ parent schema sensitivity label (3F000).
- session sensitivity label $\geq$ referenced table/view sensitivity label (VT002).
- session sensitivity label = view sensitivity label of any view referenced by a foreign key (RI010).

All objects created are labeled with the session sensitivity label.

### DAC

The user must have the `rubix.dac.rxcreate.dbname` authorization or have all of the following privileges:

- $\uparrow$ EXEC on database (RA001)
- $\uparrow$ EXEC on catalog where the view is created (RA008)
- $\uparrow$ EXEC, WRITE on schema where the view is created (RA002)
- $\uparrow$ CRVIEW on all the columns of tables/views directly listed in the query expression (RC003); for these tables or views, the user must have EXEC on their catalog(s) (RA008) and EXEC on their schema(s) (RA002)
- $\uparrow$ REFVIEW on all columns of the base tables as calculated in the query expression (RA003); for these base tables, the user must have EXEC on their catalog(s) (RA008) and EXEC on their schema(s) (RA002)

The creator of the view is given an initial set of DAC privileges that is calculated from those he/she holds on the underlying table(s). If the viewed table is not updateable then the creator will not be given INSERT, UPDATE, or DELETE privileges on any columns regardless of what the user holds on the underlying table(s). Noting that exception, the creator is given the INSERT, UPDATE, DELETE, and SELECT on a column in the view only if the user has that column privilege on the underlying table. If the user is given any column privilege on the view he is always given the WITH GRANT OPTION on that privilege and the GRANTNULL privilege on that column.

### AUDIT

The `sql_view_create` event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, view label, catalog name, schema name, table name, view structure, view definition

### 3.1.9 DROP CATALOG

#### FUNCTION

Destroy a catalog.

#### SYNOPSIS

```
DROP CATALOG catalog-name \{CASCADE | RESTRICT\}
```

#### DESCRIPTION

The `DROP CATALOG` statement destroys the catalog specified by `catalog-name`. 

---

See the **Security Features User Guide** for a discussion of security issues pertaining to `CREATE VIEW`. 

---
The application must use the keyword CASCADE or RESTRICT. Doing so specifies the effect if the catalog contains any schema:

- → CASCADE
  All such objects are also destroyed.

- → RESTRICT
  The statement fails if there is any schema other than the default _schema (3D505) or if the default_schema is not empty (3F505).

**SECURITY**

**MAC**
Session sensitivity label = catalog sensitivity label (3D506, if session sensitivity label > catalog sensitivity label; 3D000, otherwise).
Session sensitivity label = all objects (schema, table, view) inside the catalog (3D505).
Session sensitivity label = all views that reference a table or view inside the catalog (RD004).

**DAC**
The user must have the `rubix.dac.rxdrop.dbname` authorization or have all of the following privileges:

- ↑ EXEC, WRITE on database (RA001)

**AUDIT**
(sql_catalog_drop) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, catalog label, catalog name.

**NOTE**
To ensure error-free operation, make sure that all cursors are closed prior to issuing a DROP command.

**3.1.10 DROP INDEX**

**FUNCTION**
Destroy an index.

**SYNOPSIS**
Drop INDEX `index-name`

**DESCRIPTION**
The DROP INDEX statement removes the index `index-name` from the database.

**SECURITY**

**MAC**
Session sensitivity label >= parent catalog sensitivity label (3D000).
Session sensitivity label >= parent schema sensitivity label (3F000).
Session sensitivity label = index sensitivity label (XX007, if session sensitivity label > index sensitivity label; RR005, otherwise).

**DAC**
The user must have the `rubix.dac.rxdrop.dbname` authorization or have all of the following privileges:

- ↑ EXEC on database (RA001)
- ↑ EXEC on catalog (RA008)
- ↑ EXEC, WRITE on schema (RA002)
3.1.11 DROP SCHEMA

FUNCTION
Destroy a schema.

SYNOPSIS
DROP SCHEMA [catalog-name.]schema-name {CASCADE | RESTRICT}

DESCRIPTION
The DROP SCHEMA statement destroys the schema specified by schema-name.

The application must use the keyword CASCADE or RESTRICT. Doing so specifies the effect if the schema contains any object (table, index, or view):

→ CASCADE
   All such objects are also destroyed.
→ RESTRICT
   The statement fails if any such objects exist

SECURITY
MAC  Session sensitivity label >= parent catalog sensitivity label (3D000).
     Session sensitivity label = schema sensitivity label (3F506, if session sensitivity label > schema sensitivity label; 3F000, otherwise).
     Session sensitivity label = all objects (table, view) inside the schema (3F505).
     Session sensitivity label = all views that reference a table or view inside the schema (RD004).

DAC  The user must have the rubix.dac.rxdrop.dbname authorization or have all of the following privileges:
     ↑ EXEC on database (RA001)
     ↑ EXEC, WRITE on catalog (RA008)

AUDIT  (sql_schema_drop) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, schema label, schema name

NOTE
To ensure error-free operation, make sure that all cursors are closed prior to issuing a DROP command.
3.1.12 DROP TABLE

FUNCTION
Destroy a base table.

SYNOPSIS
DROP TABLE [catalog-name.][schema-name.]base-table-name {CASCADE | RESTRICT} {DEALLOCATE}

DESCRIPTION
The DROP TABLE statement removes base-table-name from the database. It also drops any indexes based on that base table, and revokes any privileges that were granted on that base table.

The application may use the keyword CASCADE or RESTRICT. If no keyword is specified then RESTRICT is the default. Doing so specifies the effect if FOREIGN KEY or CHECK constraints reference base-table-name or if viewed tables are based on base-table-name:

- CASCADE
  All such constraints and viewed tables are also dropped.
- RESTRICT (default)
  The operation fails if any such object exist.

If the application uses the DEALLOCATE keyword then all physical resources associated with base-table-name will be permanently deallocated during the corresponding transaction’s COMMIT operation. If the transaction is aborted then no deallocation occurs and base-table-name remains in the database.

NOTE
To ensure error-free operation, if the DEALLOCATE keyword is used, ensure that no active transactions access the dropped table after the transaction commits.

If the DEALLOCATE keyword is not used and the transaction commits then one of the following will occur during the transaction COMMIT:

- base-table-name is physically deallocated (as if the DEALLOCATE keyword were used) if there are no active transactions that may potentially access the table.
- base-table-name is logically removed from the database if there are active transactions that may potentially access the table. No physical resources associated with the table are removed. The physical resources may be removed at a future time by the Trusted RUBIX Maintenance Server. Please see the Trusted Facility Manual for more information about the Maintenance Server.

SECURITY
MAC
Session sensitivity label >= parent catalog sensitivity label (3D000).
Session sensitivity label >= schema sensitivity label (3F000).
Session sensitivity label = table sensitivity label (0A514, if session sensitivity label > table sensitivity label; RR005, otherwise).
Session sensitivity label = all views that reference the table (RD004).

DAC
The user must have the rubix.dac.rxdrop.dbname authorization or have all of the following privileges:
3.1.13 DROP VIEW

**FUNCTION**
Destroy a view.

**SYNOPSIS**

```
DROP VIEW [catalog-name.][schema-name.]viewed-table-name {CASCADE | RESTRICT}
```

**DESCRIPTION**

The DROP VIEW statement removes `viewed-table-name` from the database. It also revokes any privileges granted on the viewed table.

The application must use the keyword CASCADE or RESTRICT. Doing so specifies the effect if other viewed tables are based on `viewed-table-name`:

- → CASCADE
  All such other viewed tables are also dropped.
- → RESTRICT
  The operation fails if any such objects exist.

**SECURITY**

**MAC**

Session sensitivity label >= parent catalog sensitivity label (3D000).
Session sensitivity label >= schema sensitivity label (3F000).
Session sensitivity label = view sensitivity label (0A514, if session sensitivity label > view sensitivity label).
Session sensitivity label = all views that reference the table (RD004).

**DAC**

The user must have the `rubix.dac.rxdrop.dbname` authorization or have all of the following privileges:

- ↑ EXEC on database (RA001)
- ↑ EXEC on catalog (RA008)
- ↑ EXEC, WRITE on schema (RA002)

**Audit**

To ensure error-free operation, make sure that all cursors are closed prior to issuing a DROP command.
view name.

NOTE

To ensure error-free operation, make sure that all cursors are closed prior to issuing a DROP command.

3.1.14 GRANT

FUNCTION

Grant DAC privileges on a database, catalog, schema, table, view or column. When this command is invoked by a user holding the *rubix.dac.rxgrant.dbname* authorization (nominally the default Database Administrator role), it functions as an administrative tool useful for managing DAC privileges.

SYNOPSIS

```
GRANT [GRANT OPTION FOR] [ ALL PRIVILEGES | privilege [ , privilege ] . . . ]
  ON { DATABASE | CATALOG catalog-name | SCHEMA [catalog-name.]schema-name ]
  TABLE [catalog-name.]|[schema-name.|table-name ]
  VIEW [catalog-name.|[schema-name.|view-name ] TO { user-name [ , user-name ] . . .
  GROUP group-name [ , group-name ] . . PUBLIC ] [WITH GRANT OPTION] [ ASSUME
  { user-name [ , user-name ] . . . GROUP group-name [ , group-name ] . . PUBLIC ] ]
```

where *privilege* is one of the following:

- SELECT [( column-identifier [ ,column-identifier ] . . . ) ]
- INSERT [( column-identifier [ ,column-identifier ] . . . ) ]
- UPDATE [( column-identifier [ ,column-identifier ] . . . ) ]
- REFERENCES [ column-identifier [ ,column-identifier ] . . . ) ]
- CRVIEW [ ( column-identifier [ ,column-identifier ] . . ) ]
- REFVIEW [ ( column-identifier [ ,column-identifier ] . . ) ]
- NULL [ ( column-identifier [ ,column-identifier ] . . ) ]
- GRANTNULL [ ( column-identifier [ ,column-identifier ] . . ) ]
- DELETE
- READ
- WRITE
- EXEC

DESCRIPTION

The GRANT statement grants privileges on database objects, to one or more users or groups (grantee), provided the current user (grantor) has the appropriate grant privileges on that object.

The ability to further grant any privilege is governed by the WITH GRANT OPTION. Only a user who had previously been granted a privilege that included the WITH GRANT OPTION may further propagate it. If the WITH GRANT OPTION is specified as part of the GRANT command for a NULL privilege, it is ignored. To grant the NULL privilege to someone the GRANTNULL privilege is required. The GRANTNULL privilege has an associated WITH GRANT OPTION.

If the object is a database, the current database is used. Specifying a READ grants the right to retrieve the list of schemas in the database. WRITE grants the right to create and drop catalogs. EXEC grants the right to use any fully specified catalog. To perform these functions, the user must connect to the database which requires EXEC privilege. Specifying a NULL specifically disallows any operation on the object. Specifying
a GRANTNULL grants the right to give the NULL privilege to other users.

If the object is a schema or a catalog, the user is granted the privileges for schema-name or catalog-name. Specifying a READ grants the right to retrieve the names of the objects in the schema or catalog. WRITE grants the right to create and drop objects. EXEC grants the right to use the objects. Specifying a NULL specifically disallows any operation on the object. Specifying a GRANTNULL grants the right to give the NULL privilege to other users.

If the object is a table, the user is granted the privileges for columns of table-name. Specifying SELECT grants the right to retrieve column-identifiers. DELETE grants the right to delete rows. INSERT grants the right to insert column-identifiers. UPDATE grants the right to update column-identifiers. REFERENCES grants the right to reference column-identifiers from a FOREIGN KEY. CRVIEW grants the right to create views on column-identifiers. REFVIEW grants the right to access a view that is defined on the column-identifiers. NULL specifically disallows any operation on column-identifiers. GRANTNULL grants the right to grant the NULL privilege on column-identifiers. Specifying any privilege without column-identifiers implicitly grants the privilege on all grantable columns contained in table-name.

If the object is a view, user is granted privileges for columns of view-name. The privileges of the view override the privileges of all underlying objects. However, this can only take place if the user has REFVIEW privilege on the underlying objects. SELECT grants the right to retrieve column-identifiers. DELETE grants the right to delete rows. INSERT grants the right to insert column-identifiers. UPDATE grants the right to update column-identifiers. NULL specifically disallows any operation on column-identifiers. GRANTNULL grants the right to grant the NULL privilege on column-identifiers. Specifying any privilege without column-identifiers implicitly grants the privilege on all grantable columns contained in view-name.

Specifying GRANT ALL PRIVILEGES is equivalent to specifying all the grantable privileges that the current user has on that object. The application specifies PUBLIC to grant the specified privilege to all present and future users, GROUP to grant the specified privilege to all present and future users of the group-name, or identifies specific users by user-name.

The application specifies WITH GRANT OPTION to allow any users affected by this GRANT statement to further grant the privileges to other users. A user may only receive a privilege through a single thread of control. If a privilege is repeatedly granted, all grants other than the first one will generate an error message. To take control of a particular privilege that a user possesses by some previous grant, first revoke it then grant it again.

A special form of the GRANT command, known as GRANT ASSUME may be used by the database administrator (those with the rubix.dac.rxgrant.* authorization) to change the grantor of pre-existing privileges. The anticipated use of this command to allow the database administrator to assign a user (the grantee) the new responsibility of managing a set of DAC privileges (belonging to the assumee) with regard to a particular object. The GRANT ASSUME form of the GRANT command will cause the grantee (subject specified by the “TO” clause) to become the grantor of pre-existing privileges that belong to the assumee (subject of the “ASSUME” clause). In this case the grantee of the GRANT command may already have the privilege. The grantee of the GRANT ASSUME command must be a user-name and not any of the following:

- a group-name,  → a list of users or groups, or  → PUBLIC.

Additionally, one of the following must be true of the grantee:

1. The grantee of a GRANT ASSUME must not already have the privilege or
2. The grantee must have a grantor of RUBIX (root of DAC privilege tree).
3. The grantee must have the same grantor as the assumee (same parents).
4. The grantee must be the grantor of the grantor of the assumee (grantee is the grandparent of the assumee).

A GRANT ASSUME results in the grantee having privileges (from the privilege list) with the WITH GRANT OPTION. The WITH GRANT OPTION clause should not be used with GRANT ASSUME.

If any privilege explicitly requested to be granted (e.g., GRANT EXEC, READ…) may not be granted because of insufficient user privilege an error code is returned and no privileges are granted. If a group of privileges are implicitly requested to be granted (e.g., GRANT ALL…) then those privileges that the user has the privilege to grant are granted and the remaining privileges are not. In each case where any number of privileges are granted the number of granted privileges is displayed (ISQL). If an error occurs while processing the command no privileges are granted.

**SECURITY**

**MAC** Session sensitivity label >= parent catalog sensitivity label, if applicable (3D000).
Session sensitivity label >= schema sensitivity label, if applicable (3F000).
Session sensitivity label = object sensitivity label (PA005)
All created ACL rows are labeled with the session sensitivity label.

**DAC** If the ASSUME clause is not specified the user must have the `rubix.dac.rxgrant.dbname` authorization or have all of the following privileges:

- The privilege being granted with the GRANT OPTION (RA001 for databases, RA002 for schemas,
- RA003 for table/view columns, RA008 for catalogs)
- EXEC on database (RA001) for catalogs, schemas, relations, and views
- EXEC on catalog (RA008) for schemas, table columns, and view columns
- EXEC on schema (RA002) for table columns and view columns

If the ASSUME clause is specified the user must have the `rubix.dac.rxgrant.dbname` authorization.

**AUDIT** (sql_acl_modify) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, ACL label, operation type, new ACL bitmap, grantor, object name

### 3.1.15 REVOKE

**FUNCTION**

Revoke DAC privileges on a database, catalog, schema, table, view or column. When this command is invoked by a user holding the `rubix.dac.rxgrant.dbname` authorization (nominally the default Database Administrator role), it functions as an administrative tool useful for managing DAC privileges.

**SYNOPSIS**

```
REVOKE [GRANT OPTION FOR] { ALL PRIVILEGES | privilege [ , privilege ] ... } 
ON { DATABASE | CATALOG catalog-name | SCHEMA [catalog-name.]schema-name | TABLE [catalog-name.]schema-name.]table-name | VIEW [catalog-name.]schema-name.]view-name | ALL }
```
FROM { user-name [ , user-name ] . . .
GROUP group-name [ , group-name ] . . . PUBLIC }
{ CASCADE | RESTRICT | ASSIGN user-name }

where privilege is one of the following:

SELECT [ ( column-identifier [ , column-identifier ] . . . ) ]
INSERT [ ( column-identifier [ , column-identifier ] . . . ) ]
UPDATE [ ( column-identifier [ , column-identifier ] . . . ) ]
REFERENCES [ ( column-identifier [ , column-identifier ] . . . ) ]
CRVIEW [ ( column-identifier [ , column-identifier ] . . . ) ]
REFVIEW [ ( column-identifier [ , column-identifier ] . . . ) ]
NULL [ ( column-identifier [ , column-identifier ] . . . ) ]
GRANTNULL [ ( column-identifier [ , column-identifier ] . . . ) ]
DELETE
READ
WRITE
EXEC

where column-identifier is the name assigned to the table column

DESCRIPTION

The REVOKE statement revokes privileges on a database, catalog, schema, table columns, or view columns to one or more users, provided the current user has the appropriate privilege on that object.

If the object is a database, the current database is used. Specifying a READ revokes the right to retrieve the list of catalogs in the database. Specifying a WRITE revokes the right to create and drop catalogs. Specifying a EXEC revokes the right to use the database. Specifying a NULL removes the explicit denial of access to the database. Specifying a GRANTNULL revokes the right to GRANT the NULL privilege on the database.

If the object is a schema or a catalog, the user is revoked the privileges for schema-name or catalog-name. Specifying a READ revokes the right to retrieve the names of the objects in the schema or catalog. Specifying a WRITE revokes the right to create and drop objects. Specifying a EXEC revokes the right to use the objects. Specifying a NULL removes the explicit denial of access to the object. Specifying a GRANTNULL revokes the right to GRANT the NULL privilege on the object.

If the object is a table, the user privileges for columns of table-name are revoked. Specifying SELECT revokes the right to retrieve column-identifiers. DELETE revokes the right to delete rows. INSERT revokes the right to insert column-identifiers. UPDATE revokes the right to update column-identifiers. REFERENCES revokes the right to reference column-identifiers from a FOREIGN KEY. CRVIEW revokes the right to create views on column-identifiers. REFVIEW revokes the right to access a view that is defined on the column-identifiers. NULL revokes the specific denial of any operation on column-identifiers. GRANTNULL revokes the right to grant the NULL privilege on column-identifiers. Specifying any privilege without column-identifiers implicitly revokes the privilege on all grantable columns contained in table-name.

If the object is a view, the user privileges for columns of view-name are revoked. SELECT revokes the right to retrieve column-identifiers. DELETE revokes the right to delete rows. INSERT revokes the right to insert column-identifiers. UPDATE revokes the right to update column-identifiers. CRVIEW revokes the right to create views on column-identifiers. NULL revokes the specific denial of any operation on column-identifiers. GRANTNULL revokes the right to grant the NULL privilege on column-identifiers. Specifying any privilege without column-identifiers implicitly revokes the privilege on all grantable columns contained in view-name.
The FROM clause either identifies one or more users by their *user-names*, specifies GROUP with *group-names* or specifies PUBLIC to revoke privileges formerly extended to all users using GRANT ... PUBLIC.

The application must use either the keyword CASCADE or RESTRICT. Doing so specifies the effect on propagated privileges. If a privilege would be abandoned from its grantor, the cascade option revokes it and the restrict option causes the revoke to fail.

→ CASCADE
   Revokes all privileges which have been granted by users who, in turn, have received the grant option being revoked.

→ RESTRICT
   If any privilege has been propagated to more users, this revoke will fail.

A special form of the REVOKE command, known as REVOKE ASSIGN, is available to the database administrator (with the *rubix.dac.rxrevoke.* authorization). The REVOKE ASSIGN command is an alternative to the CASCADE and RESTRICT options. The anticipated use of the REVOKE ASSIGN command is to allow the transference of object DAC management responsibilities to a new user when a user is removed. The REVOKE ASSIGN performs the following actions:

1. The revokee’s (subject of the FROM clause) specified privilege(s) is revoked.
2. The assignee (subject of the ASSIGN clause) is granted the same privilege(s) revoked along with the WITH GRANT OPTION.
3. The assignee replaces the revokee as the grantor of all specified privileges previously granted by the revokee.

The assignee and the revokee of the REVOKE ASSIGN must be a user name and not be any of the following → a group name, → or a list of users or groups, → or PUBLIC.

Additionally, one of the following must be true of the assignee:

→ The assignee must not already have the privilege.
→ The assignee must have a grantor of RUBIX (root of DAC privilege tree).
→ The assignee must be the grantor of the revokee (assignee direct parent of revokee).
→ The assignee must be the grantee of the revokee (assignee direct child of revokee).

If any privilege explicitly requested to be granted (e.g., GRANT EXEC, READ…) may not be granted because of insufficient user privilege an error code is returned and no privileges are granted. If a group of privileges are implicitly requested to be granted (e.g., GRANT ALL…) then those privileges that the user has the privilege to grant are granted and the remaining privileges are not. In each case where any number of privileges are granted the number of granted privileges is displayed (ISQL). If an error occurs while processing the command no privileges are granted.

 adultes Security

MAC
   Session sensitivity label &gt;= parent catalog sensitivity label, if applicable (3D000).
   Session sensitivity label &gt;= schema sensitivity label, if applicable (3F000).
   Session sensitivity label = object sensitivity label (PA005)

DAC
   If the ASSIGN clause is not specified, the user must have the *rubix.dac.rxrevoke.dbname*
   authorization or have all of the following privileges:

   ↑ The privilege being revoked with the GRANT OPTION (RA001 for databases, RA002 for schemas, RA003 for table/view columns, RA008 for catalogs) EXEC on database (RA001)
for catalogs, schemas, relations and views

↑ EXEC on catalog (RA008) for schemas, table columns, and view columns

↑ EXEC on schema (RA002) for table columns and view columns

If the ASSIGN clause is specified, the user must have the `rubix.dac.rxrevoke.dbname` authorization.

AUDIT (sql_acl_modify) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, ACL label, operation type, new ACL bitmap, revokee, object name
4. Data Manipulation Statements

4.1 DATA MANIPULATION STATEMENTS

A data manipulation statement operates on the data contents of the database or controls the state of a cursor.

INSERT and UPDATE statements assign values to columns in tables.

Applications should not mix data definition statements and data manipulation statements within transactions.

### NOTE

In addition to the Multilevel Security (MLS) Mandatory Access Control (MAC) policy, TR supports the Type Enforcement (TE) MAC policy of SELinux and a proprietary Attribute Based Accessed Control (ABAC) MAC policy of the Security Policy Manager (SPM). In general, all configured MAC policies must permit an operation for it to succeed. For more information on TE and SELinux please see the **Trusted RUBIX SELinux Guide** and for more information on ABAC and the SPM please see the **Trusted RUBIX Security Policy Manager Reference Guide and Tutorial**.

#### 4.1.1 Searched DELETE

**FUNCTION**

Delete rows of a table that satisfy a `search-condition`.

**SYNOPSIS**

```
DELETE FROM table-name [ WHERE search-condition]
```

**DESCRIPTION**

This form of DELETE deletes any rows from `table/view-name` that satisfy `search-condition`. If `table/view-name` is a view, DELETE deletes the corresponding rows of the base table from which it is derived. If no row satisfies `search-condition`, DELETE deletes zero rows. If `search-condition` is omitted, DELETE deletes all the rows in the table or view.

`table/view-name` must be updateable and not referenced (directly or indirectly) by a FROM clause of any `sub-query` contained in the `search-condition`.

The scope of the `table/view-name` is the entire searched DELETE statement.

If `search-condition` is specified, then DELETE applies it to each row of `table/view-name` and deletes all rows for which the `search-condition` is true.

Each `sub-query` in the `search-condition` is effectively executed for each row of `table/view-name`. The results are used in the evaluation of the `search-condition` for that row.

If any executed `sub-query` contains a reference to a column of `table/view-name`, the reference is to the
value of that column in the given row.

If the sub-query references any Information Schema view, then the special DAC rules of Information Schema Guide are applied. If the DAC rules defined are not met, then that particular row is not visible. That is, the DELETE behaves as if those rows were not in the underlying tables.

If any deleted row is referenced by a foreign key constraint its ON DELETE behavior is executed as follows:

- **NO ACTION**
  The positioned DELETE or searched DELETE fails with 23000 and makes no change to the database.

- **CASCADE**
  All rows where any referencing-columns are affected are also deleted. If some columns of a deleted row are themselves the object of FOREIGN KEY references, the deletion of the row may cause additional rows to be deleted, possibly including rows in other tables.

- **SET NULL**
  Affected referencing-columns are set to the null value.¹

- **SET DEFAULT**
  Each affected column of referencing-columns is set to its respective default value.

### SECURITY

**MAC**
- Session sensitivity label >= parent catalog sensitivity label (3D000).
- Session sensitivity label >= schema sensitivity label (3F000).
- Session sensitivity label >= referenced table sensitivity label (VT002).
- Session sensitivity label = row sensitivity label (SS003)

Only rows that satisfy the condition (session sensitivity label >= row sensitivity label) are included in the search-condition. The highest labeled polyinstantiated row of each set of polyinstantiated rows is returned.

ON DELETE behavior for foreign key constraints that may alter the contents of other rows (SET NULL, SET DEFAULT, CASCADE) only have effect when the row sensitivity label equals the row sensitivity label of the referenced column’s row.

**DAC**

The user must have the *rubix.dac.rxdelete dbname* authorization or have all of the following privileges:

- EXEC on database (RA001)
- EXEC on catalog (RA008)
- EXEC on schema (RA002)
- DELETE on table (RA003)
- SELECT on referenced query columns (RA003)
- REFVIEW on all tables underlying referenced views (RA003).

If a view is being referenced in the sub-query that resides in the Information Schema, additional DAC rules apply as defined in the Trusted Rubix Information Schema Guide.

¹ Doing so may itself violate an integrity constraint, such as a NOT NULL or CHECK constraint. In this case, the positioned DELETE or searched DELETE fails as described for the NO ACTION case.
4.1.2 INSERT

FUNCTION
Insert rows into a table or view

SYNOPSIS
INSERT INTO [catalog-name].[schema-name.]table/view-name {insert-source | DEFAULT VALUES}

where insert-source is defined as:

[(unqualified-column-name [, unqualified-column-name] ... )]
{query-specification |
 VALUES row-value-constructor}

DESCRIPTION
INSERT inserts row values into table/view-name. (If table/view-name is a viewed table, then INSERT inserts into the base table from which table/view-name is derived.)

Depending on the form used, INSERT inserts one row containing one of the following:
- values, for specified columns or for all columns, derived from query-specification;
- values, for specified columns or for all columns, explicitly specified by a row-value-constructor or,
- values for all columns that are the default values for that column in table/view-name.

In the first two cases, the \( i \)th value specified is assigned to the \( i \)th column of the table or to the \( i \)th unqualified-column-name.

Omitting the unqualified-column-names is equivalent to specifying all the columns of table/view-name, in ascending order of their position in the table.

If unqualified-column-names are present, each must identify a column of table/view-name and there can be no duplicates. There must be the same number of unqualified-table/view-names as the number of values the INSERT statement produces. If INSERT omits some columns of table/view-name, then these columns of the new row are set to their default values. If table/view-name is a view, this is also the case for any column of its base table not included in the viewed table.

The table table/view-name must be updateable and not referenced by a FROM clause of the query-specification or of any sub-query contained in the query-specification.

If the insert-source references any Information Schema view, then the special DAC rules of the Information Schema Guide are applied. If the DAC rules defined are not met, then that particular row is not visible. That is, the DELETE behaves as if those rows were not in the underlying tables.

SECURITY
MAC Session sensitivity label \( \geq \) parent catalog sensitivity label (3D000).
Session sensitivity label \( \geq \) schema sensitivity label (3F000).
Session sensitivity label \( \geq \) referenced table sensitivity label (VT002)
All created rows are labeled with the session sensitivity label.

If the polyinstantiation level of the table is POLYNONE then an error occurs (VI004), and the insert is rejected if a duplicate key exists at any label.

If the polyinstantiation level of the table is POLYLOW then an error occurs (VI004), and the insert is rejected if a duplicate key exists at any label dominated by the session sensitivity label.

If the polyinstantiation level of the table is POLYHIGH then an error occurs (VI004), and the insert is rejected if a duplicate key exists at a label equal to the sessions sensitivity label.

Only rows that satisfy the condition (session sensitivity label \( \geq \) row sensitivity label) are included in the input-source. The highest labeled polyinstantiated row of each set of polyinstantiated rows is returned.

Foreign key constraints are enforced only when the row sensitivity label equals the row sensitivity label of the referenced column’s row. Check constraints operate only on rows that satisfy the condition (session sensitivity label \( \geq \) row sensitivity label).

DAC The user must have the rubix.dac.rxinsert.dbname authorization or have all of the following privileges:

- EXEC on database (RA001)
- EXEC on catalog (RA008)
- EXEC on schema (RA002)
- INSERT on referenced columns (RA003)
- SELECT on referenced query columns (RA003)
- REFVIEW on all tables underlying referenced views (RA003).

If a view is being referenced in the input-source that resides in the Information Schema, additional DAC rules apply as defined in Section 9.

AUDIT (sql_row_create) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, row label, catalog name, schema name, relation name, row data.
### 4.1.3 SELECT

#### FUNCTION
Retrieve rows from a table.

#### SYNOPSIS

```
SELECT [ ALL | DISTINCT] select-list
FROM [catalog-name.][schema-name.]table/view-name [,[catalog-name.][schema-name.]table/view-name . . ] [VIEW BY POLYINSTANTIATION]
[WHERE search-condition]
[GROUP BY column-name [ , column-name] . . ]
[HAVING search-condition]
[ORDER BY select-column-ref [ASC | DESC] [, select-column-ref [ASC | DESC]] . . .]
[LIMIT limit-value] [OFFSET offset-value]
```

where `select-list` is defined as:

* | `select-sublist` [ , `select-sublist`]. .

and `select-sublist` is defined as:

expression [AS column-identifier] . . | {table-name | correlation-name} . *

where `select-column-ref` is defined as:

the output name of a column in `select-list` | ordinal number of column in `select-list`

#### DESCRIPTION

A `query-specification` is the basic data retrieval construct. It derives a table by selecting specific fields from one or more tables subject to specified criteria. Columns of derived tables, defined by `query-specification`, derive their data types from the expression in the `query-specification`, that defines columns. A `sub-query` is a limited form of `query-specification` that derives values for use within a predicate. The `SELECT` statement is used to produce `query-specifications` that are used to output data and may be used in combinations with other data manipulation statements. Performing an `INSERT`, `UPDATE`, `DELETE`, or `SELECT` command on a defined view implicitly executes the view’s `SELECT` definition statement.

If the `query-specification` references any Information Schema view, then the special DAC rules of Section 11 are applied. If the DAC rules defined are not met, then that particular row is not visible. That is, the `SELECT` behaves as if those rows were not in the underlying tables.

If the `VIEW BY POLYINSTANTIATION` option is specified then all visible copies of polyinstantiated rows are returned.

#### EXPRESSIONS

The `expression` part of a query represents a single value. It consists of one of the following:

- a literal value,
- a column-name,
- a `set-function` reference,
- one of the `scalar functions`,
- any valid combination of these primary components connected by `arithmetic operators`,...
→ or any valid combination of these primary components connected by string operators.

Valid scalar functions are CHAR_LENGTH, CHARACTER_LENGTH, EXTRACT, OCTET_LENGTH, POSITION, SUBSTRING, TRIM, or CAST.

Valid set functions are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Data Type of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>Average of the values in the column.</td>
<td>Same generic data type as the argument; ( TR ) defined attributes.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Total number of values in (cardinality of) the column. (^2)</td>
<td>Exact numeric, scale 0, ( TR ) defined precision and range.</td>
</tr>
<tr>
<td>MAX</td>
<td>Largest value in the column.</td>
<td>Same as the argument.</td>
</tr>
<tr>
<td>MIN</td>
<td>Smallest value in the column.</td>
<td>Same as the argument.</td>
</tr>
<tr>
<td>SUM</td>
<td>Sum of the values in the column.</td>
<td>Exact numeric argument: Exact numeric result; same scale as argument; ( TR ) defined precision and range. Approximate numeric argument: Approximate numeric result; ( TR ) defined</td>
</tr>
</tbody>
</table>

Valid arithmetic functions are (in descending order of precedence):

+ , -  unary plus and minus (negation)

\(*, /\)  multiplication and division

+ , -  addition and subtraction

Valid string operators are:

<table>
<thead>
<tr>
<th>Simplified Syntax</th>
<th>Result Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>CHAR_LENGTH(s1)</td>
<td>An exact numeric indicating the length of s1.</td>
</tr>
<tr>
<td>CHARACTER_LENGTH(s1)</td>
<td>Same as CHAR_LENGTH.</td>
</tr>
<tr>
<td>LOWER(s)</td>
<td>A copy of s with any uppercase letters converted to lowercase.</td>
</tr>
<tr>
<td>OCTET_LENGTH(s1)</td>
<td>Same as CHAR_LENGTH.</td>
</tr>
<tr>
<td>POSITION(s1 IN s2)</td>
<td>An exact numeric indicating the character position of the first occurrence of s1 in s2.</td>
</tr>
</tbody>
</table>

\(^2\) COUNT(*) does not refer to columns but returns the cardinality of the entire virtual table (or, if it occurs in a HAVING clause, the cardinality of the current group).
### SECURITY

**MAC**  Session sensitivity label >= parent catalog sensitivity label (3D000).  
Session sensitivity label >= schema sensitivity label (3F000).  
Session sensitivity label >= table/view sensitivity label (VT002)

Only rows that satisfy the condition (session sensitivity label >= row sensitivity label) are included in the query. The highest labeled polynstantiated row of each set of polynstantiated rows is returned. If the VIEW BY POLYINSTANTION is specified all dominated, polynstantiated rows are returned.

**DAC**  The user must have the *rubix.dac.rxselect.dbname* authorization or have all of the following privileges:

↑ EXEC on database (RA001)  
↑ EXEC on catalog (RA008)  
↑ EXEC on schema (RA002)  
↑ SELECT on all referenced columns (RA003)  
↑ REFVIEW on all tables underlying referenced views (RA003).

If a view is being referenced that resides in the Information Schema, additional rules apply as defined in the Information Schema Guide.

**AUDIT**  (sql_fetch) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, composite row label, current catalog, current schema, cursor definition

### 4.1.4 Searched UPDATE

**FUNCTION**  Update values of columns in a table.

**SYNOPSIS**

```sql
UPDATE [catalog-name.][schema-name.]table/view-name SET unqualified-column-name =
{ expression | NULL | DEFAULT } | , unqualified-column-name =
{ expression | NULL | DEFAULT } | . . . [ WHERE search-condition ]
```

**DESCRIPTION**  UPDATE updates zero or more rows of *table/view-name*. If *table-name* is a viewed table, UPDATE applies the updates to the corresponding rows of the base table from which it is derived.
Each *unqualified-column-name* identifies a column to be updated (an update column) and must identify a column of *table-name* (the update table). The same column must not be identified more than once.

The update table must be updateable and must not be referenced (directly or indirectly) by the FROM clause of any sub-query contained in the search-condition.

The scope of the *table/view-name* is the entire searched UPDATE statement.

The syntax to the right of the equal sign represents the update value of the corresponding update column. That is, for each update column, the value of that column in the row to be updated is replaced by the value of the corresponding *expression*, or by null value if NULL is specified, or by the column’s default value if DEFAULT is specified.

If *expression* is specified, it must not include a *set-function-reference* and any column it references must be a column of the update table. The value specified by such a referenced column is the value of the identified column in the row to be updated before any values in that row are updated.

If *search-condition* is not specified, all rows of the update table are updated.

If *search-condition* is specified, it is applied to each row of the update table and all rows for which the result of the *search-condition* is true are updated.

Each sub-query in the *search-condition* is effectively executed for each row of the update table and the results used in the evaluation of the *search-condition* for that row.

If any executed sub-query contains a reference to a column of the update table, the reference is to the value of that column in the given row.

If the sub-query references any Information Schema view, then the special DAC rules of Section 11 are applied. If the DAC rules defined are not met, then that particular row is not visible. That is, the UPDATE behaves as if those rows were not in the underlying tables.

If any update column is referenced by a foreign key constraint its ON UPDATE behavior is executed as follows:

- **NO ACTION**
  - The searched update fails with '23000' and makes no change to the database.

- **CASCADE**
  - All rows where any referencing-columns are affected are also updated. If some columns of an updated row are themselves the object of foreign key references, the update of the row may cause additional rows to be updated, possibly including rows in other tables.

- **SET NULL**
  - Affected referencing-columns are set to the null value

- **SET DEFAULT**
  - Each affected column of referencing-columns is set to its respective default value.

**SECURITY**

MAC  Session sensitivity label >= parent catalog sensitivity label (3D000).

---

3 Doing so may itself violate an integrity constraint, such as a NOT NULL or CHECK constraint. In this case, the searched UPDATE fails as described for the NO ACTION case.
Session sensitivity label >= schema sensitivity label (3F000).
Session sensitivity label >= referenced table sensitivity label (VT002)
Session sensitivity label = row sensitivity label (SS008).

Only rows that satisfy the condition (session sensitivity label >= row sensitivity label) are included in the expression or search-condition. The highest labeled polyinstantiated row of each set of polyinstantiated rows is returned.

If the polyinstantiation level of the table is POLYNONE then an error occurs (VI004), and the update is rejected if a duplicate key exists at any label.

If the polyinstantiation level of the table is POLYLOW then an error occurs (VI004), and the update is rejected if a duplicate key exists at any label dominated by the session sensitivity label.

If the polyinstantiation level of the table is POLYHIGH then an error occurs (VI004), and the update is rejected if a duplicate key exists at a label equal to the session sensitivity label.

ON UPDATE behavior for foreign key constraints that may alter the contents of other rows (SET NULL, SET DEFAULT, CASCADE) only have effect when the row sensitivity label equals the row sensitivity label of the referenced column’s row.

Check constraints operate only on rows that satisfy the condition (session sensitivity label >= row sensitivity label).

**DAC**
The user must have the `rubix.dac.rxupdate.dbname` authorization or have all of the following privileges:

- EXEC on database (RA001)
- EXEC on catalog (RA008)
- EXEC on schema (RA002)
- UPDATE on referenced update columns or table as a whole (RA003)
- SELECT on referenced query columns or table as a whole (RA003)
- REFVIEW on all tables underlying referenced views (RA003).

If a view is being referenced in the sub-query that resides in the Information Schema, additional DAC rules apply as defined in the Information Schema Guide.

**AUDIT**
(sql_row_update) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, row label, catalog name, schema name, relation name, old row data, new row data

### 4.1.5 SET CATALOG

**FUNCTION**
Set the DEFAULT_CATALOG name for unqualified object references.

**SYNOPSIS**

```
SET CATALOG catalog-name
```
### DESCRIPTION
The `SET CATALOG` statement sets the default_catalog name (for references to objects that do not specify a catalog name) to the value of `catalog-name`, with any leading and trailing spaces removed.

`TR` does not check, during the execution of the `SET CATALOG` statement, that `catalog-name` actually exists. This check occurs when the embedded SQL program subsequently tries to reference an object that does not specify a catalog.

### SECURITY
**MAC**  
Session sensitivity label >= catalog sensitivity label (3D000)

**DAC**  
The user must have any `rubix.dac.*.dbname` authorization or have all of the following privileges:

- EXEC on database (RA001)
- EXEC on catalog (RA008)
- EXEC on default_schema (RA002)

**AUDIT**  
None

### 4.1.6 SET SCHEMA

#### FUNCTION
Set the default schema name for unqualified object reference.

#### SYNOPSIS

```
SET SCHEMA [catalog-name.] schema-name
```

#### DESCRIPTION
The `SET SCHEMA` statement sets the default schema name, and may set the default catalog name, for reference to unqualified objects (that is, for objects that do not specify a catalog and schema name).

If `schema-name` contains a `.` character, then the characters preceding `.` are the default catalog name and the characters following `.` are the default schema name. If `schema-name` does not contain a `.` character, the `schema-name` becomes the default schema name and the default catalog name does not change.

**Trusted Rubix** ignores any leading and trailing spaces in `schema-name`.

`TR` does not check, during the execution of the `SET SCHEMA` statement, that any specified schema and catalog actually exist. This check occurs when the embedded SQL program subsequently tries to reference an unqualified object.

#### SECURITY
**MAC**  
Session sensitivity label >= parent catalog sensitivity label (3D000).  
Session sensitivity label >= schema sensitivity label (3F000)

**DAC**  
The user must have any `rubix.dac.*.dbname` authorization or have all of the following privileges:

- EXEC on database (RA001)
- EXEC on catalog (RA008)
↑ EXEC on schema (RA002)

AUDIT  None
5. Transaction Control Statements

5.1 TRANSACTION CONTROL STATEMENTS

A transaction is a sequence of operations on a database that must be atomic. This means that the database system must either effect all of the operations or effect none of them.

A transaction starts (becomes active) when a program that does not already have an active transaction executes an SQL statement that operates on a database. An “SQL statement that operates on a database” does not include:

- the transaction control statements COMMIT, ROLLBACK and SET TRANSACTION discussed later in this section (although, technically, completing a transaction using COMMIT and ROLLBACK operates on a database, it does not start a new transaction);
- the connection statements (see page 94); this means an application can change connections without starting a new transaction;
- the SET CATALOG statement
- the SET SCHEMA statement ; or,

Applications should complete each transaction explicitly with either the COMMIT or ROLLBACK statement. Ceasing execution without ending the transaction causes either a COMMIT or ROLLBACK according to Trusted Rubix-defined criteria.

Applications should not mix data definition statements and data manipulation statements within transactions.

5.1.1 COMMIT

FUNCTION
Complete the current transaction by applying all of the database changes.

SYNOPSIS
COMMIT [WORK]

DESCRIPTION
Complete the current transaction by applying all of the database changes. The COMMIT statement completes the current transaction; closes any cursors that the transaction opened; and commits any changes made to the database since the transaction started.

SECURITY
MAC All updates to the database (INSERT, etc.) labeled at the session sensitivity label are made permanent.

DAC All DAC operations (GRANT, etc.) are made permanent. The DAC operations are immediately visible to transactions that start after the committing transaction.
5.1.2 ROLLBACK

**FUNCTION**
Complete the current transaction by applying none of the database changes.

**SYNOPSIS**
ROLLBACK [ WORK ]

**DESCRIPTION**
Complete the current transaction by applying none of the database changes. The ROLLBACK statement completes the current transaction; closes any cursors that the transaction opened; and provides that any changes made during the transaction do not take effect.

If the ROLLBACK fails internally, the database may be in an inconsistent state and a fatal error has occurred. In this case, the database should be restored using the rxrestore command.

**SECURITY**

- **MAC** All modifications to the database for the current open transaction (INSERT, etc.) labeled at the session sensitivity label are removed from the database.

- **DAC** All DAC operations (GRANT, etc.) are removed from the database.

**AUDIT** (sql_trans_rollback) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID.

5.1.3 SET TRANSACTION

**FUNCTION**
Set attributes of the next transaction.

**SYNOPSIS**
SET TRANSACTION transaction-mode [ , transaction-mode ] ...

where *transaction-mode* is defined as:

- ISOLATION LEVEL READ UNCOMMITTED
- ISOLATION LEVEL READ COMMITTED
- ISOLATION LEVEL REPEATABLE READ
- ISOLATION LEVEL SERIALIZABLE
- READ ONLY
- READ WRITE

**DESCRIPTION**
Set attributes of the next transaction. The SET TRANSACTION statement specifies the values of the transaction attributes. These values apply to the next transaction the application may begin over the current connection. In Trusted Rubix, the specified attributes do not persist if the application
should disconnect and reconnect to the same database. Since SET TRANSACTION affects only the next transaction, applications should execute SET TRANSACTION before the start of each transaction.

The keywords SET TRANSACTION are followed by transaction-mode clauses. The DIAGNOSTICS SIZE clause specifies the size of the diagnostics area; the READ ONLY and READ WRITE clauses specify the access mode, the others specify the isolation level. Redundant and contradictory designations in a single SET TRANSACTION statement are invalid.

Defaults
If a SET TRANSACTION statement does not specify the access mode, the resulting access mode is READ WRITE, except that a statement that specifies ISOLATION LEVEL READ UNCOMMITTED implies READ ONLY. It is invalid to specify both READ WRITE and READ UNCOMMITTED in the same SET TRANSACTION statement.

If a SET TRANSACTION statement does not specify the isolation level, the ISOLATION LEVEL SERIALIZABLE is assumed.

If an application starts a transaction without having executed SET TRANSACTION since the end of any previous transaction, the access mode is READ WRITE, the isolation level is ISOLATION LEVEL SERIALIZABLE, and the size of the diagnostics area is TTR-defined.

↓ APPLICATION USAGE
Since SET TRANSACTION affects only the next transaction, application should execute SET TRANSACTION before the start of each transaction.

↓ SECURITY
MAC None
DAC None
AUDIT None

5.2 SAVEPOINTS
Savepoints allow the system to record the current state of processing during an SQL session. This state is retained so that in the event of an error, the application can return to the most recently declared savepoint.

The use of the savepoint mechanism is described in the SQL-ODBC Tutorial. It is available in both client types (RXISQL and ODBC).

5.2.1 DECLARE SAVEPOINT

↓ FUNCTION
Savepoints allow the system to record the current state of processing during an SQL session.

↓ SYNOPSIS

SAVEPOINT savepoint-specifier

where savepoint-specifier is defined as:
Savepoints allow the system to record the current state of processing during an SQL session. This state is retained so that in the event of an error, the application can return to the most recently declared savepoint.

The SAVEPOINT statement creates a savepoint in the current transaction during the current SQL session. Savepoint-names must be distinct within a given transaction. If a second savepoint is created with the same savepoint-name as an earlier savepoint, the earlier savepoint is erased.

**5.2.2 RELEASE SAVEPOINT**

**FUNCTION**
The RELEASE SAVEPOINT statement releases a previously declared savepoint.

**SYNOPSIS**
```
RELEASE SAVEPOINT savepoint-specifier
```

where savepoint-specifier is defined as:
```
savepoint-name | unsigned-integer
```

**DESCRIPTION**
The RELEASE SAVEPOINT statement releases a previously declared savepoint. If savepoint-specifier does not refer to a savepoint defined within the current SQL transaction, an exception is raised. The savepoint referred to in this statement and all subsequent savepoints established in the current SQL transaction are destroyed.

**SECURITY**

MAC  None.
DAC  None.
AUDIT  (sql_savepoint_release) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, object label, savepoint, savepoint name
**SYNOPSIS**

ROLLBACK TO SAVEPOINT savepoint-specifier

where savepoint-specifier is defined as:

    savepoint-name | unsigned-integer

**DESCRIPTION**

Rollback all database actions since the establishment of a savepoint. This statement is used to cancel all changes to SQL data or schemas that were made by the current SQL transaction since the establishment of the savepoint. If the savepoint referred to in this command does not specify a savepoint that was established within the current SQL transaction, an exception is raised.

If the ROLLBACK fails internally, the database may be in an inconsistent state and a fatal error has occurred. In this case, the database should be restored using the rxrestore command.

**SECURITY**

MAC   All updates (INSERT, etc.) created up to the savepoint (inclusive) labeled at the session sensitivity label are removed from the database.

DAC   All DAC operations (GRANT, etc.) created up to the savepoint (inclusive) are removed from the database.

AUDIT  (sql_savepoint_rollback) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, session ID, object label, savepoint, savepoint name.
6. SESSION CONTROL STATEMENTS

6.1 SET LOGGING

↓ FUNCTION
Enables or disables restore logging for the current session.

↓ SYNOPSIS
SET LOGGING logging_option

where logging_option can be:

ON | OFF

↓ DESCRIPTION
SET LOGGING allows a user to dynamically enable or disable restore logging. Typically, this would be used to increase performance during a bulk load operation. The command fails and an error is returned if there currently is an active transaction. Valid options for this command are:

→ ON
  restore logging is enabled for the current session. The command fails and an error is returned if the database was created with restore logging disabled.

→ OFF
  restore logging is disabled for the current session.

The restore logging setting will only change the logging behavior for the current database session. No other sessions will be affected.

↓ SECURITY
MAC  None.

DAC   None.

AUDIT None.

6.2 ALTER SESSION SET LABEL

↓ FUNCTION
Alters the current session label for the user.

↓ SYNOPSIS
ALTER SESSION SET LABEL = label_parameter

where label_parameter can be:

'label_text'
  | DBHIGH
  | DBLOW
| OSLABEL

**DESCRIPTION**

ALTER SESSION SET LABEL allows a user to dynamically alter the session label for the user. The upper value of the new session label is bounded by the user’s logon clearance label. Valid parameters for this command are:

- `'label_text'`
  any label in valid default label format (as dictated by the operating system)

- `DBHIGH`
  the label equivalent to SYSTEM HIGH

- `DBLOW`
  the label equivalent to SYSTEM LOW

- `OSLABEL`
  the user’s operating system label

The user must have one of the `rubix.mac.setsess.*` authorizations and there must not be a current transaction.

Altering the session label does not alter the operating system label that the user was assigned at system login time.

**SECURITY**

**MAC**

The session sensitivity label is set to the new value.

The new session sensitivity label <= clearance label (SS010).

**DAC**

The user must have one of the `rubix.mac.setsess.*` authorizations (RA009).

If the user has the `rubix.mac.setsess.up*` then the session label may be changed to a value where the new label dominates the user’s original logon session sensitivity label. If the user has the `rubix.mac.setsess.uprd.dbname` then the user may only submit read-only transactions (25500). If the user has the `rubix.mac.setsess.uprdwrt.dbname` then the user may submit read/write transactions.

If the user has the `rubix.mac.setsess.across*` then the session label may be changed to a value where the new label is incomparable with the user’s original logon session sensitivity label. If the user has the `rubix.mac.setsess.acrossrd.dbname` then the user may only submit read-only transactions (25500). If the user has the `rubix.mac.setsess.acrossrdwrt.dbname` then the user may submit read/write transactions.

If the user has the `rubix.mac.setsess.downrdwrt` then the session label may be changed to a value where new label is dominated by the user’s original logon session sensitivity label. The user may submit read/write transactions. In all cases, if the database session label is set back to the user’s original logon session sensitivity label they may submit read/write transactions.

**AUDIT**

(sql_lvl_alter) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID, new session label
7. Connection and Diagnostic Statements

7.1 CONNECTION STATEMENTS

Connection statements give the application access to multiple servers.

7.1.1 Current and Dormant Connections

The CONNECT statement connects the client (and effectively, the application bound to that client) to a server; the DISCONNECT statement ends this connection. A client can be connected to several servers at the same time, and can establish several connections to the same server. (The maximum number of concurrent connections is TR defined.) However, only one connection is current at any time. Other connections are dormant.

The connection that the CONNECT statement specifies becomes the current one. If an application establishes multiple connections, it can use the SET CONNECTION statement to select one connection as the current one. In either case, if a previous connection was current, it becomes dormant. The application cannot use a server through a dormant connection; it must first use SET CONNECTION.

Making a connection dormant and then current again is comparable to disconnecting and then reconnecting to the server, except that dormancy typically avoids the need to perform authentication again, and may avoid costs and uses of resources associated with an initial connection.

A current or dormant connection is active if any SQL statement that operates on TR has been successfully executed at its associated server using that connection during the current transaction. The application must complete the current transaction before DISCONNECT will succeed.

7.1.2 Default Connection

If the first SQL statement an application executes is not CONNECT, a default connection is established to the server. (See the ISQL section of this guide for a discussion of how the default resource is determined.) This remains the current connection until the application executes a CONNECT statement. All connection statements let the application specify this connection explicitly using the keyword DEFAULT. DEFAULT is a discrete connection just as named connections are.

7.1.3 State Table

The following state table shows the state of connection conn1 resulting from the execution of a connection statement referencing it or referencing some other connection conn2. A “—” entry means the state of conn1 is unchanged. A SQLSTATE value means that the statement fails because of the error indicated by that SQLSTATE value.
### 7.1.4 Connection Context

Each current or dormant connection has a set of information called the **connection context**. This includes:

- the position of all open cursors
- the contents of all SQL descriptor areas
- the name of the current user and all information the server associates with this name
- all state information the server needs in order to reference the application
- other **Trusted Rubix**-defined context

When the current connection becomes dormant, the server saves the connection’s context. When a dormant connection becomes current, the server restores the context to its exact state at the time the connection became dormant.

### 7.1.5 General Diagnostics

In **TR** a single transaction may not encompass more than one server. Thus, using CONNECT or SET CONNECTION when a transaction is active sets SQLSTATE to ‘0A001’.

**Connection Errors**

The DISCONNECT statement describes how an application explicitly disconnects from a server. Disconnection can occur independently of the application, because of communication errors, administrative action, or other events. Four SQLSTATE values indicate a disconnection:

- The value ‘08006’ occurs on a SET CONNECTION statement when the selected connection has been independently disconnected and cannot be reestablished.
- The value ‘08007’ occurs on a COMMIT statement when the current connection has been independently disconnected.
- The value ‘40003’ occurs on a statement, other than COMMIT, that is processed by a server when the current connection has been independently disconnected. The transaction is rolled back.
- The value ‘08003’ indicates either the connection selected by a SET CONNECTION or DISCONNECT statement has been explicitly disconnected or the following conditions apply:
  - the statement is processed by a server;
  - the current connection has been explicitly disconnected, or independently disconnected but not since the last statement was executed;
  - the lost current connection was not the implicit default connection.

A DISCONNECT statement specifying an active connection while a transaction is active is rejected.

---

1 A connection’s initial state in this table is Nonexistent.
However, independent disconnection of the current connection (except on a COMMIT) rolls back any active transaction ('43003').

If the current connection is disconnected, then there is no current connection; subsequent SQL statements report the nonexistent connection ('08003') until the application uses a CONNECT or SET CONNECTION statement. If the implicit default connection is independently disconnected, then implicit reconnection occurs, as it did when the application began.

The diagnostics area may contain additional information on the reason for a connection failure.

Figure 1: Reporting of Connection Errors (Flowchart)

The preceding flowchart illustrates the reporting of connection errors and the special considerations that apply to the implicit default connection.

### 7.1.6 CONNECT

**FUNCTION**

CONNECT connects the application to a server.

**SYNOPSIS**

`CONNECT TO { server-name [ AS connection ] }`

where `server` and `connection` are defined as a `user-defined-name`

**DESCRIPTION**

CONNECT connects the application to a server. The `server-name` is the name of the server to which the client will connect. The specified connection becomes the current connection. If another connection was...
current, it becomes dormant.

The optional connection-name can be used to reference the connection to server-name in any subsequent SET CONNECTION or DISCONNECT statements. If the application does not specify connection, then server is the implicit connection.

**SECURITY**

**MAC**  
Session sensitivity label >= database sensitivity label (08001)

**DAC**  
The user must have any rubix.dac.*.dbname authorization or have the following privilege:

↑  
EXEC on database (RA001)

**AUDIT**  
(sql_db_open) event name, user ID, group ID, database name, session sensitivity label, operation status, timestamp, transaction ID, process ID, session ID

### 7.1.7 DISCONNECT

**FUNCTION**

DISCONNECT ends a connection.

**SYNOPSIS**

DISCONNECT { connection | ALL | CURRENT | DEFAULT }

where connection is defined as a user-defined-name

**DESCRIPTION**

The form DISCONNECT connection ends the specified connection. DISCONNECT CURRENT ends the current connection. DISCONNECT DEFAULT ends the default connection. DISCONNECT ALL ends all the application’s current and dormant connections.

A DISCONNECT statement that does not end the current connection makes no change to the context of the current connection.

### 7.1.8 SET CONNECTION

**FUNCTION**

Make a specified connection the current one.

**SYNOPSIS**

SET CONNECTION { connection | DEFAULT }

where connection is defined as a user-defined-name

**DESCRIPTION**

The SET CONNECTION statement makes the specified connection the current one. If another connection was previously current, it becomes dormant.
8. RDBMS USER AND APPLICATION USER STATEMENTS

8.1 RDBMS User Statements

8.1.1 AUTHENTICATE USER

FUNCTION

AUTHENTICATE USER authenticates a RDBMS user and sets the RDBMS user in the current database session.

SYNOPSIS

AUTHENTICATE USER [= rdbms-user] [GROUP = rdbms-group] [PASSWORD = ‘passphrase’]

DESCRIPTION

The AUTHENTICATE USER command authenticates a RDBMS user and sets the RDBMS user in the current database session. The user, group, and password must be valid for the operating system (OS) on the platform hosting the RDBMS server. The authentication occurs on the RDBMS host platform using the Pluggable Authentication Module (PAM). Valid parameters for this command are:

rdbms-user: The user to be authenticated. It may be given as a text user name or an integer user ID. The referenced user must be valid on the RDBMS host platform OS. If using the rxisql interactive SQL client and no rdbms-user is provided, the user will be prompted to input a rdbms-user.

rdbms-group: The group to be associated with the user for the current RDBMS session. It may be given as a text group name or an integer group ID. The referenced group must be valid on the RDBMS host platform OS and valid for the user give by rdbms-user. If no rdbms-group is given then the user’s default group is used.

passphrase: The password used to authenticate the user. It must be given as a single-quoted string. The passphrase must be the passphrase assigned to the user on the RDBMS host platform OS. If using the rxisql interactive SQL client and no passphrase is provided, the user will be prompted to input a passphrase.

8.2 Application and Application User Statements

8.2.1 CREATE APPLICATION

FUNCTION

CREATE APPLICATION creates a named application in the RDBMS.

SYNOPSIS

CREATE APPLICATION app-name
where app-name is a user-defined-name
DESCRIPTION
The application given by \textit{app-name} is created in the RDBMS. Application administrators may subsequently be assigned to the application using the \texttt{CREATE APPLICATION ADMIN} command. These application administrators may then accept operations from application users.

SECURITY
MAC  The application sensitivity label is set to the current session sensitivity label.

DAC  The user must have the \texttt{rubix.app.create[dbname]} authorization.

8.2.2 DROP APPLICATION

FUNCTION
DROP APPLICATION drops a named application, all of the associated application administrators, and optionally all of the associated application users.

SYNOPSIS
DROP APPLICATION \texttt{app-name[CASCADE | RESTRICT]}

DESCRIPTION
Drop the application \textit{app-name}, all associated application administrators, and optionally all of the associated application users. Valid parameters for this command are:

CASCADE: Automatically drop all of the application users associated with the application being dropped.

RESTRICT: Fail (\texttt{AU002}) if any application user exists that is associated with the application being dropped.

If no CASCADE or RESTRICT is given, RESTRICT is assumed.

SECURITY
MAC  Session sensitivity label = application sensitivity label (\texttt{AP001})

DAC  The user must have the \texttt{rubix.app.drop[dbname]} authorization.

8.2.3 ALTER APPLICATION

FUNCTION
ALTER APPLICATION renames an existing application.

SYNOPSIS
ALTER APPLICATION \texttt{app-name SET NAME = 'new-app-name'}

DESCRIPTION
The application given by *app-name* is renamed to *new-app-name*.

**SECURITY**

MAC   Session sensitivity label = application sensitivity label (AP001)

DAC   The user must have the *rubix.app.alter[.dbname]* authorization.

### 8.2.4 ALTER SESSION SET APPLICATION

**FUNCTION**

ALTER SESSION SET APPLICATION sets the named application as being current in the database session.

**SYNOPSIS**

ALTER SESSION SET APPLICATION = *app-name*

where *app-name* is defined as a *user-defined-name*

**DESCRIPTION**

ALTER SESSION SET APPLICATION sets the named application as being current in the database session. The RDBMS user issuing this command must be an application administrator for the named application. Once the current application is set, the middleware application may service operations on behalf of application users.

**SECURITY**

MAC   Session sensitivity label >= application sensitivity label (AP001)

DAC   The RDBMS user must be an application administrator for the application.

### 8.2.5 CREATE APPLICATION_ADMIN

**FUNCTION**

CREATE APPLICATION_ADMIN creates an application administrator for a named application.

**SYNOPSIS**

CREATE APPLICATION_ADMIN APPLICATION = *app-name* USER = *rdbms-user*

where *app-name* is defined as a *user-defined-name* and *rdbms-user* is a valid OS user on the RDBMS server platform.

**DESCRIPTION**

CREATE APPLICATION_ADMIN creates an administrator for the application named by *app-name*. The RDBMS user named by *rdbms-user* will be assigned as the application administrator. The application administrator is permitted to set the application as current in its database session, create application users for the application, and authenticate application users for the application.

**SECURITY**

MAC   The application administrator sensitivity label is set to the current session sensitivity label.

DAC   The user must have the *rubix.app.admin.create[.dbname]* authorization.
8.2.6 DROP APPLICATION_ADMIN

\rightarrow FUNCTION
DROP APPLICATION_ADMIN drops an application administrator from a named application.

\rightarrow SYNOPSIS
DROP APPLICATION_ADMIN APPLICATION = app-name USER = rdbms-user
where app-name is defined as a user-defined-name and rdbms-user is a valid OS user on the RDBMS server platform.

\rightarrow DESCRIPTION
DROP APPLICATION_ADMIN drops an application administrator named by rdbms-user from an application named by app-name.

\rightarrow SECURITY
MAC  Session sensitivity label = application administrator sensitivity label (AA004)
DAC  The user must have the rubix.app.admin.drop[.dbname] authorization.

8.2.7 CREATE APPLICATION_USER

\rightarrow FUNCTION
CREATE APPLICATION_USER creates an application user for the application that is currently set in the database session.

\rightarrow SYNOPSIS
CREATE APPLICATION_USER app-user WITH PASSWORD ‘passphrase’
where app-user is defined as a user-defined-name.

\rightarrow DESCRIPTION
CREATE APPLICATION_USER creates an application user named by app-user for the application that is currently set in the database session. The application user may subsequently be authenticated using the password given by passphrase. The current session application must have been previously set using the ALTER SESSION SET APPLICATION command.

\rightarrow SECURITY
MAC  The application user sensitivity label is set to the current session sensitivity label.
DAC  The current session application must have been previously set using the ALTER SESSION SET APPLICATION command. The RDBMS user must be an application administrator for the application currently set in the database session.

8.2.8 DROP APPLICATION_USER

\rightarrow FUNCTION
DROP APPLICATION_USER drops an application user for the application that is currently set in the database session.
SYNOPSIS

DROP APPLICATION_USER app-user
where app-user is defined as a user-defined-name

DESCRIPTION

DROP APPLICATION_USER drops an application user for the application that is currently set in the database session. The current session application must have been previously set using the ALTER SESSION SET APPLICATION command.

SECURITY

MAC   Session sensitivity label = application user sensitivity label (AU001)

DAC   The current session application must have been previously set using the ALTER SESSION SET APPLICATION command. The RDBMS user must be an application administrator for the application currently set in the database session and either:
   a) the application user being dropped must currently be authenticated in the database session;
   or
   b) the RDBMS user must have the rubix.app.user.drop[.dbname] authorization.

8.2.9 ALTER APPLICATION_USER

FUNCTION

ALTER APPLICATION_USER changes the password or name associated with an application user.

SYNOPSIS

ALTER APPLICATION_USER app-user SET {PASSWORD = 'passphrase' | NAME = 'new-app-user'}
where app-user and new-app-user are defined as a user-defined-name

DESCRIPTION

ALTER APPLICATION_USER changes the password (to passphrase) or the name (to new-app-user) for the application user named by app-user. The current session application must have been previously set using the ALTER SESSION SET APPLICATION command.

SECURITY

MAC   Session sensitivity label = application user sensitivity label (AU001)

DAC   The current session application must have been previously set using the ALTER SESSION SET APPLICATION command. The RDBMS user must be an application administrator for the application currently set in the database session and either:
   a) the application user being altered must currently be authenticated in the database session;
   or
   b) the RDBMS user must have the rubix.app.user.alter.name[.dbname] authorization, if updating the name, or the rubix.app.user.alter.password[.dbname] authorization, if updating the password.
8.2.10 AUTHENTICATE APPLICATION_USER

\subsection{FUNCTION}
AUTHENTICATE APPLICATION_USER authenticates an application user using a password and sets the application user for the current database session.

\subsection{SYNOPSIS}
AUTHENTICATE APPLICATION_USER = app-user PASSWORD = ‘passphrase’
where app-user is defined as a user-defined-name

\subsection{DESCRIPTION}
AUTHENTICATE APPLICATION_USER authenticates an application user named by app-user using the password passphrase and sets the application user for the current database session. The application user will remain authenticated until a failed authentication occurs or for the period of time defined by the application’s authentication timeout, as specified in the rxconfig configuration file. To end an application user’s authenticated session the AUTHENTICATE APPLICATION_USER command may be issued with a null passphrase string. The current session application must have been previously set using the ALTER SESSION SET APPLICATION command.

\subsection{SECURITY}
MAC Session sensitivity label >= application user sensitivity label (AU001)
DAC The current session application must have been previously set using the ALTER SESSION SET APPLICATION command. The RDBMS user must be an application administrator for the application currently set in the database session.

8.2.11 ALTER SESSION SET APPLICATION_USER

\subsection{FUNCTION}
ALTER SESSION SET APPLICATION_USER sets the application user for the current database session.

\subsection{SYNOPSIS}
ALTER SESSION SET APPLICATION_USER = app-user
where app-user is defined as a user-defined-name

\subsection{DESCRIPTION}
ALTER SESSION SET APPLICATION_USER sets the current session application user to app-user in the current database session. The application user must currently be authenticated from a previous use of the AUTHENTICATE APPLICATION_USER command. The current session application must have been previously set using the ALTER SESSION SET APPLICATION command.

\subsection{SECURITY}
MAC Session sensitivity label >= application user sensitivity label (AU001)
DAC The current session application must have been previously set using the ALTER SESSION SET APPLICATION command. The RDBMS user must be an application administrator for the application currently set in the database session. The application user must currently be authenticated from a previous use of the AUTHENTICATE APPLICATION_USER command.
9. TRUSTED RUBIX LIMITS

9.1 LIMITS

This section defines the maximum values for certain limits that application developers may safely assume Trusted Rubix supports. Open Group specifies a portability limit of 254 on the length of a character string. This asserts that all Open Group-compliant implementations shall set any such limit at 254 or higher, and asserts that portable applications must conform to a limit of 254. Thus, Open Group only requires that any such limits be set to at least the value it specifies.

9.1.1 Supplementary Definitions

The guaranteed Open Group minimum values are specified for limits on “attributes” a term used to refer to various aspects of an SQL implementation.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Open Group</th>
<th>TRUSTED RUBIX Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of a character string</td>
<td>254</td>
<td>3MB</td>
</tr>
<tr>
<td>Maximum length of a variable-length character string</td>
<td>254</td>
<td>3MB</td>
</tr>
<tr>
<td>Maximum length of a character large object (CLOB)</td>
<td></td>
<td>2GB</td>
</tr>
<tr>
<td>Length of a bit string</td>
<td></td>
<td>3MB</td>
</tr>
<tr>
<td>Maximum length of a variable length bit string</td>
<td></td>
<td>3MB</td>
</tr>
<tr>
<td>Maximum length of a binary large object (BLOB)</td>
<td></td>
<td>2GB</td>
</tr>
</tbody>
</table>

In the ALLOCATE DESCRIPTOR statement, the maximum value and default value for occurrences

<table>
<thead>
<tr>
<th></th>
<th>Open Group</th>
<th>TRUSTED RUBIX Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision of any DECIMAL or NUMERIC data type, in decimal digits</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>Mantissa precision of any FLOAT data type, in bits</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>Precision of a SMALLINT number, in decimal digits</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Precision of an INTEGER number, in decimal digits</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Mantissa precision of a REAL number, in bits</td>
<td>21</td>
<td>47</td>
</tr>
<tr>
<td>Mantissa precision of a DOUBLE PRECISION number, in bits</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>Precision of SQLCODE, in bits</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>Values of type TIME (decimal fraction second precision)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Values of type TIMESTAMP (decimal fraction second precision)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Values of type INTERVAL (decimal fraction second precision)</td>
<td>N/A</td>
<td>6</td>
</tr>
<tr>
<td>Leading precision for INTERVAL data types, in decimal digits</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Leading precision for INTERVAL data types that have a SECOND field</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

NOTE

Character strings that serve as server names or connection names in connection statements are limited to 128 characters.
### 9.1.2 Additional Limits

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Open Group</th>
<th>TRUSTED RUBIX Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Database Size</td>
<td>N/A</td>
<td>64TB</td>
</tr>
<tr>
<td>Maximum Table Size</td>
<td>N/A</td>
<td>64TB</td>
</tr>
<tr>
<td>Number of rows per table</td>
<td>N/A</td>
<td>no limit</td>
</tr>
<tr>
<td>Number of columns in a table</td>
<td>100</td>
<td>256</td>
</tr>
<tr>
<td>Number of indexes per table</td>
<td>N/A</td>
<td>256</td>
</tr>
<tr>
<td>Number of columns constituting an index</td>
<td>6</td>
<td>256</td>
</tr>
<tr>
<td>Number of tables a statement references</td>
<td>10</td>
<td>128</td>
</tr>
<tr>
<td>Number of tables per database</td>
<td>N/A</td>
<td>2 billion</td>
</tr>
<tr>
<td>Number of databases per server</td>
<td>N/A</td>
<td>no limit</td>
</tr>
<tr>
<td>Max. number of databases referenced per query</td>
<td>N/A</td>
<td>no limit</td>
</tr>
<tr>
<td>Number of cursors simultaneously open</td>
<td>10</td>
<td>256</td>
</tr>
<tr>
<td>Number of columns that a single <code>UPDATE</code> statement can update</td>
<td>20</td>
<td>256</td>
</tr>
<tr>
<td>Levels of nesting of sub-queries, not counting the outermost level</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Predicates in a <code>WHERE</code> or <code>HAVING</code> clause</td>
<td>N/A</td>
<td>No limit</td>
</tr>
<tr>
<td>Number of columns in a <code>UNIQUE</code> constraint</td>
<td>N/A</td>
<td>256</td>
</tr>
<tr>
<td>Number of columns in a <code>GROUP BY</code> column list</td>
<td>N/A</td>
<td>256</td>
</tr>
<tr>
<td>Number of sort items in an <code>ORDER BY</code> clause</td>
<td>N/A</td>
<td>256</td>
</tr>
<tr>
<td>Number of <code>referencing-columns</code> in a <code>FOREIGN KEY</code></td>
<td>N/A</td>
<td>No limit</td>
</tr>
<tr>
<td>Number of columns in a named-column <code>JOIN</code></td>
<td>N/A</td>
<td>256</td>
</tr>
<tr>
<td>Max. length of a row</td>
<td>2000</td>
<td>3MB</td>
</tr>
<tr>
<td>Total length of an index key</td>
<td>120</td>
<td>8KB</td>
</tr>
<tr>
<td>Length of columns specified in a <code>GROUP BY</code> clause</td>
<td>120</td>
<td>3MB</td>
</tr>
<tr>
<td>Length of columns specified in an <code>ORDER BY</code> clause</td>
<td>120</td>
<td>3MB</td>
</tr>
<tr>
<td>Length of columns specified in a <code>FOREIGN KEY</code> column list</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Length of columns specified in a <code>JOIN</code> column list</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Length of columns in a <code>UNIQUE</code> constraint</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**NOTE:**

Databases can span multiple disks, but currently no SQL `standard` syntax exists to specify volume file name. An implementation shall permit at least the following complexity as measured by the Open Group storage model.
9.1.3 Statement Complexity

Any limits on the following attributes imposed by Open Group compliant implementations shall be at least as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Open Group</th>
<th>TRUSTED RUBIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of an SQL &lt;schema definition&gt;</td>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td>Length of an SQL &lt;data statement&gt;</td>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td>Length of an SQL &lt;statement variable&gt;</td>
<td>4000</td>
<td>4000</td>
</tr>
</tbody>
</table>

9.1.4 Large Objects (LOBS)

Large objects (BLOB and CLOB data types) may not be used as an argument to any expression and may not occur in a WHERE clause or a column constraint. When selecting BLOB and CLOB data using the interactive rxisql client, only the BLOB/CLOB ID is displayed. The ID is an integer value that uniquely identifies the BLOB/CLOB value. An ODBC application must be used to insert, update, and select BLOB/CLOB data.

9.2 RESTRICTIONS ON NAMES

Most implementations have reserved words in addition to those listed in this specification (see page 4-9). To ensure portability, applications should avoid using identifiers that might be reserved. For example, include underscore characters or digits in the identifiers.

To avoid name conflicts between system-defined and user-defined objects, applications should not use procedure, function or variable names starting with sql or SQL or SYS.

Some implementations have keyword-oriented parsers. To ensure portability, the names of embedded host variables should be distinct from SQL keywords (including the additional reserved words referred to above).

In TRusted Rubix user-name lengths are operating system dependent since they are really operating system names. The lengths of user-name are the same as the underlying operating system. In Solaris or Trusted Solaris they are set to 32 characters.

9.3 DATA DEFINITION STATEMENTS IN TRANSACTIONS

Many RDBMS do not give the user the granularity of transaction control required by Open Group. In some cases, a data definition statement is a separate transaction that is immediately committed; in other cases, executing a data definition statement automatically commits all previously executed data manipulation statements. In TR, data definition statements are fully controlled by transaction control statements as Open Group specifies.

9.4 SELECT

The definition of a query-specification allows the use of two forms containing * to specify a sequence of expressions. However, the sequence is established at run time. Therefore, the result of evaluating these
query-specifications (for example, the number of columns retrieved by a SELECT statement) can differ among implementations. Portable applications should not use these * forms, except in CREATE VIEW, in which * is expanded when the view definition is originally executed.
10. ERROR CONDITION AND MESSAGES

This chapter documents the error conditions and messages that Trusted RUBIX supports.

10.1 SQL ERROR CONDITIONS

The SQL error conditions and messages consist of two parts. A five digit error code and a corresponding error message. When using ISQL, both the code and the message are printed out automatically in the event of a failure. When using ODBC, the error code and message may be explicitly retrieved from the diagnostic area following the failure of a function.

10.1.1 Common Error Conditions

The set of common error codes and messages may generally be generated by any SQL command.

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A000</td>
<td>syntax error or access rule violation in direct SQL statement</td>
<td>Syntax error in direct SQL statement. The parser has rejected the statement.</td>
</tr>
<tr>
<td>90034</td>
<td>unable to write to the system audit trail</td>
<td>Audit log full (fatal error). The file system used to hold the audit logs do not have enough space to hold an audit record.</td>
</tr>
<tr>
<td>91004</td>
<td>invalid user name specified</td>
<td>Invalid user name on secondary authentication (only for those commands which automatically connect). Secondary authentication was enabled and the user passed in a user name that does not exist on the server host.</td>
</tr>
<tr>
<td>91005</td>
<td>invalid group name specified</td>
<td>Invalid group name on secondary authentication (only for those commands which automatically connect). Secondary authentication was enabled and the user passed in a group name that does not exist on the server host.</td>
</tr>
<tr>
<td>PW009</td>
<td>authentication failure</td>
<td>Authentication failure (only for those commands which automatically connect). Secondary authentication was enabled and the user passed in a password which does not match the password for the given user name on the server host.</td>
</tr>
<tr>
<td>08001</td>
<td>SQL-client unable to establish SQL-connection</td>
<td>Named database does not exist (only for those commands which automatically connect). Note: this is the case when the session label does not dominate the database object label.</td>
</tr>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>User does not have DAC privileges needed to open the database (only for those commands which automatically connect).</td>
</tr>
</tbody>
</table>
An unspecified internal error has occurred which is not fatal. In this case the operation will be automatically rolled back and all write effects will be removed from the database.

An unspecified internal error has occurred which is fatal. In this case, the operation is not automatically rolled back and the database may be in an inconsistent state. The database should be restored using the `rxrestore` command.

Invalid operation while connected to the master database (all but CREATE and DROP DATABASE statements). Only the CREATE DATABASE and the DROP DATABASE may be issued when connected to the master database.

### 10.1.2 The CONNECT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2E000</td>
<td>invalid connection name</td>
<td>The name used for the connection is too long.</td>
</tr>
<tr>
<td>08002</td>
<td>connection name in use</td>
<td>The maximum number of connections from one client has been reached.</td>
</tr>
<tr>
<td>08001</td>
<td>SQL-client unable to establish SQL-connection</td>
<td>Named database does not exist. Note: this is the case when the session label does not dominate the database object label.</td>
</tr>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>User does not have the DAC privileges needed to open the database.</td>
</tr>
</tbody>
</table>

### 10.1.3 The DISCONNECT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>08003</td>
<td>connection does not exist</td>
<td>Named, default, or current connection does not exist.</td>
</tr>
</tbody>
</table>

### 10.1.4 The SET CONNECTION Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>08003</td>
<td>connection does not exist</td>
<td>Named or default connection does not exist.</td>
</tr>
</tbody>
</table>
## 10.1.5 The ALTER TABLE Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to alter the table.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to alter the table. This may correspond to the catalog that holds the table being altered or a catalog that holds a column referenced by a foreign key constraint.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to alter the table. This may correspond to the schema that holds the table being altered or a catalog that holds a column referenced by a foreign key constraint.</td>
</tr>
<tr>
<td>RA003</td>
<td>relation access violation</td>
<td>The user does not have the DAC table column privileges to alter the table. This corresponds to a column referenced by a foreign key constraint.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>A catalog specified to hold the table being altered or the column of a foreign key constraint does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or non-existent</td>
<td>A schema specified to hold the table being altered or the schema of a foreign key constraint does not exist.</td>
</tr>
<tr>
<td>RA005</td>
<td>operation limited to the owner or the DBA</td>
<td>A table specified to be altered or to contain columns referenced by a foreign key does not exist.</td>
</tr>
<tr>
<td>0A514</td>
<td>session label and table label must be equal</td>
<td>Session sensitivity label is not equal to the table sensitivity label of the table being altered.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
<tr>
<td>2D504</td>
<td>item inserted by older transaction, but not committed</td>
<td>The transaction read an uncommitted object and the transaction consistency level does not allow it.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>Transaction is read only.</td>
</tr>
<tr>
<td>VC001</td>
<td>invalid column name specified</td>
<td>The column name specified for a new column is too long.</td>
</tr>
<tr>
<td>VC003</td>
<td>column name used more than once</td>
<td>The column name for a new column already is in use by a pre-existing column.</td>
</tr>
<tr>
<td>VC002</td>
<td>maximum number of columns has been specified</td>
<td>Adding a new column will exceed the maximum number of columns allowed for one table.</td>
</tr>
<tr>
<td>VC004</td>
<td>column name specified not in original</td>
<td>The column name specified for dropping a column or being referenced by a foreign key does not exist in the table.</td>
</tr>
</tbody>
</table>
VC012  no more columns in the relation  Attempting to drop the last column in the table.
VC011  no more key columns in the relation  Attempting to drop the last primary key column in the table.
RD004  drop failed due to referencing view  Dropping a column with the RESTRICT option and there is a referencing view.
RD005  drop failed due to referencing constraint  Dropping a column with the RESTRICT option and there is a referencing constraint.
RD004  drop failed due to referencing view  Dropping a column with the CASCADE option and there is a referencing view at an object sensitivity label is not equal to the session sensitivity label.
RD005  drop failed due to referencing constraint  Dropping a column with the CASCADE option and there is a referencing constraint at an object sensitivity label is not equal to the session sensitivity label.
VX021  named constraint already exists  The specified name for a new constraint already exists in the table for a pre-existing constraint.
VX017  named constraint not found  The name specified for a constraint to be dropped does not exist in the table.
23000  integrity constraint violation  Adding a constraint will result in that constraint being violated from the current state of the database. Some number of rows existed in the table at the time the constraint was being added and the state of those rows violates the new constraint.
RI010  cross level foreign key references not permitted  A foreign key being added references the column of a table whose object sensitivity label does not equal the object sensitivity label of the table being altered.
XX008  index key too large  A constraint being added resulted in an index being automatically created and the required key for that index is too large. Index keys may be approximately ½ the size of the database page size.
22501  coercion between the specified types not defined  The default value provided will not cast into the specified column type.

### 10.1.6 The CREATE CATALOG Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D000</td>
<td>invalid catalog name</td>
<td>The specified catalog name is too long.</td>
</tr>
<tr>
<td>3D501</td>
<td>catalog specified exists already</td>
<td>The specified catalog name already exists.</td>
</tr>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the required DAC privileges on the database to create a catalog.</td>
</tr>
</tbody>
</table>
10.1.7 The CREATE DATABASE Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA011</td>
<td>database access violation</td>
<td>Database name is too long.</td>
</tr>
<tr>
<td>XD001</td>
<td>database already exists</td>
<td>The specified database name already exists.</td>
</tr>
<tr>
<td>xxxxx</td>
<td>Dependent on the CREATE CATALOG error.</td>
<td>If the statement has a sub-operation of CREATE CATALOG then any error that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>corresponds to CREATE CATALOG may be produced.</td>
</tr>
<tr>
<td>xxxxx</td>
<td>Dependent on the CREATE SCHEMA error.</td>
<td>If the statement has a sub-operation of CREATE SCHEMA then any error that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>corresponds to CREATE SCHEMA may be produced.</td>
</tr>
<tr>
<td>XD010</td>
<td>this operation can be performed only when using master</td>
<td>Not connected to the master database. The user must be connected to the</td>
</tr>
<tr>
<td></td>
<td>database</td>
<td>master database to issue the CREATE DATABASE command.</td>
</tr>
</tbody>
</table>

10.1.8 The CREATE INDEX Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the required DAC privileges on the database to create</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the required DAC privileges on the catalog to create</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the required DAC privileges on the schema to create</td>
</tr>
<tr>
<td>0A514</td>
<td>session label and table label must be equal</td>
<td>The session sensitivity label is not equal to the table object sensitivity</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or nonexistent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or nonexistent</td>
<td>The specified schema name does not exist.</td>
</tr>
</tbody>
</table>
### 10.1.9 The CREATE SCHEMA Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the required DAC privileges on the database to create a schema.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the required DAC privileges on the catalog to create a schema.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F501</td>
<td>schema specified exists already</td>
<td>The specified schema name already exists.</td>
</tr>
<tr>
<td>3F000</td>
<td>invalid schema name</td>
<td>The specified schema name is too long.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>xxxxx</td>
<td>Dependent on the CREATE TABLE error.</td>
<td>If the statement has a sub-operation of CREATE TABLE then any error that corresponds to CREATE TABLE may be produced.</td>
</tr>
<tr>
<td>xxxxx</td>
<td>Dependent on the CREATE VIEW error.</td>
<td>If the statement has a sub-operation of CREATE VIEW then any error that corresponds to CREATE VIEW may be produced.</td>
</tr>
<tr>
<td>xxxxx</td>
<td>Dependent on the GRANT error.</td>
<td>If the statement has a sub-operation of GRANT then any error that corresponds to GRANT may be produced.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
</tbody>
</table>
10.1.10 The CREATE TABLE Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to create the table.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to create the table. This may correspond to the catalog that holds the table being to be created or a catalog which holds a column referenced by a foreign key constraint.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to create the table. This may correspond to the schema that holds the table being to be created or a catalog which holds a column referenced by a foreign key constraint.</td>
</tr>
<tr>
<td>RI012</td>
<td>no references access to relation</td>
<td>The user does not have the DAC table column privileges to create the table. This corresponds to a column referenced by a foreign key constraint.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>A catalog specified to hold the table to be created or the column of a foreign key constraint does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or non-existent</td>
<td>A schema specified to hold the table to be created or the column of a foreign key constraint does not exist.</td>
</tr>
<tr>
<td>RR005</td>
<td>named relation is inaccessible or non-existent</td>
<td>A table specified to contain columns referenced by a foreign key does not exist.</td>
</tr>
<tr>
<td>VC004</td>
<td>column name specified not in original</td>
<td>A table column specified to be referenced by a foreign key does not exist.</td>
</tr>
<tr>
<td>RR004</td>
<td>relation named already exists</td>
<td>The specified table name already exists in the schema.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>VC003</td>
<td>column name used more than once</td>
<td>More than one column has been given the same name.</td>
</tr>
<tr>
<td>VC005</td>
<td>key fields have been identified twice</td>
<td>The same column has been specified as part of the primary key more than once.</td>
</tr>
<tr>
<td>VC002</td>
<td>maximum number of columns has been specified</td>
<td>The maximum number of columns for one table has been exceeded.</td>
</tr>
<tr>
<td>RR012</td>
<td>invalid relation name</td>
<td>The specified table name is too long.</td>
</tr>
<tr>
<td>VC001</td>
<td>invalid column name specified</td>
<td>The specified column name is too long.</td>
</tr>
<tr>
<td>RI018</td>
<td>invalid constraint name specified</td>
<td>The specified constraint name is too long.</td>
</tr>
</tbody>
</table>
### 10.1.11 The CREATE VIEW Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to create the view.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to create the view. This may correspond to the catalog that holds the view being to be created or a catalog that holds a column in a table/view referenced by the new view.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to create the view. This may correspond to the catalog that holds the view being to be created or a catalog that holds a column in a table/view referenced by the new view.</td>
</tr>
<tr>
<td>RC003</td>
<td>view creation requires CRVIEW privilege on underlying tables</td>
<td>The user does not have the DAC table column privileges to create the view. This corresponds to a column in a table/view referenced by the new view.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>A catalog specified to hold the view to be created or the table/view column referenced by the new view does not exist.</td>
</tr>
</tbody>
</table>
### 10.1.12 The DROP CATALOG Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to drop the catalog.</td>
</tr>
<tr>
<td>3D506</td>
<td>session level not equal to catalog level</td>
<td>The session sensitivity label is not equal to the catalog object sensitivity label.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3D503</td>
<td>catalog is in use, exclusive access required</td>
<td>The specified catalog name is currently set to be the current default catalog. It was previous set with the SET CATALOG command.</td>
</tr>
<tr>
<td>3D508</td>
<td>may not drop the default_catalog</td>
<td>The specified catalog name is “default_catalog”. The “default_catalog” may not be dropped.</td>
</tr>
<tr>
<td>3D505</td>
<td>catalog not empty</td>
<td>Dropping a catalog with the RESTRICT option and the catalog contains an object.</td>
</tr>
<tr>
<td>3D505</td>
<td>catalog not empty</td>
<td>Dropping a catalog with the CASCADE option and the catalog contains an object with an object sensitivity label that is not equal to the session sensitivity label.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
</tbody>
</table>
2D504 item inserted by older transaction, but not committed  The transaction read an uncommitted object and the transaction consistency level does not allow it.

10.1.13 The DROP DATABASE Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to drop the database.</td>
</tr>
<tr>
<td>XD014</td>
<td>session label and database label must be equal</td>
<td>The session sensitivity label is not equal to the database sensitivity label.</td>
</tr>
<tr>
<td>RA007</td>
<td>named database is inaccessible or non-existent</td>
<td>The specified database name does not exist.</td>
</tr>
<tr>
<td>XD008</td>
<td>database is in use, exclusive access required</td>
<td>The specified database is current in use. Another user is currently connected to the database being dropped.</td>
</tr>
<tr>
<td>XD010</td>
<td>this operation can be performed only when using master database</td>
<td>Not connected to the master database.</td>
</tr>
</tbody>
</table>

10.1.14 The DROP INDEX Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to drop the index.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to drop the index.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to drop the index.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or non-existent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td>XX002</td>
<td>named index does not exist</td>
<td>The specified index name does not exist.</td>
</tr>
<tr>
<td>XX009</td>
<td>may not drop an automatically created system index</td>
<td>The specified index name belongs to a system index that was automatically generated.</td>
</tr>
<tr>
<td>XX007</td>
<td>session label and index label must be equal</td>
<td>The session sensitivity label and the index object sensitivity label are equal.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
</tbody>
</table>
2D504  item inserted by older transaction, but not committed  The transaction read an uncommitted object and the transaction consistency level does not allow it.

### 10.1.15 The DROP SCHEMA Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to drop the schema.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to drop the schema.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or non-existent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td>3F503</td>
<td>schema is in use, exclusive access required</td>
<td>The specified schema is the current default schema. It was previously with the SET SCHEMA command.</td>
</tr>
<tr>
<td>3F507</td>
<td>may not drop the default_schema</td>
<td>The specified schema name is “default_schema”. The “default_schema” may not be dropped from any catalog.</td>
</tr>
<tr>
<td>3F506</td>
<td>session level not equal to schema level</td>
<td>The session sensitivity label is not equal to the schema object sensitivity label.</td>
</tr>
<tr>
<td>3F505</td>
<td>schema not empty</td>
<td>Dropping a schema with the RESTRICT option and the schema contains an object.</td>
</tr>
<tr>
<td>3F505</td>
<td>schema not empty</td>
<td>Dropping a schema with the CASCADE option and the schema contains an object with an object sensitivity label that is not equal to the session sensitivity label.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
<tr>
<td>2D504</td>
<td>item inserted by older transaction, but not committed</td>
<td>The transaction read an uncommitted object and the transaction consistency level does not allow it.</td>
</tr>
</tbody>
</table>

### 10.1.16 The DROP TABLE Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to drop the table.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to drop the table.</td>
</tr>
<tr>
<td>SQL Code</td>
<td>Printed Message</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to drop the table.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or non-existent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td>RR005</td>
<td>named relation is inaccessible or non-existent</td>
<td>The specified table name does not exist.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>RD004</td>
<td>drop failed due to referencing view</td>
<td>Dropping a table with the RESTRICT option and there is a referencing view.</td>
</tr>
<tr>
<td>RD005</td>
<td>drop failed due to referencing constraint</td>
<td>Dropping a table with the RESTRICT option and there is a referencing constraint.</td>
</tr>
<tr>
<td>RD004</td>
<td>drop failed due to referencing view</td>
<td>Dropping a table with the CASCADE option and there is a referencing view at an object sensitivity label is not equal to the session sensitivity label.</td>
</tr>
<tr>
<td>RD005</td>
<td>drop failed due to referencing constraint</td>
<td>Dropping a table with the CASCADE option and there is a referencing constraint at an object sensitivity label is not equal to the session sensitivity label.</td>
</tr>
<tr>
<td>SS004</td>
<td>session label and relation label must be equal</td>
<td>The session sensitivity label is not equal to the table object sensitivity label.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
<tr>
<td>2D504</td>
<td>item inserted by older transaction, but not committed</td>
<td>The transaction read an uncommitted object and the transaction consistency level does not allow it.</td>
</tr>
</tbody>
</table>

### 10.1.17 The DROP VIEW Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to drop the view.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to drop the view.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to drop the view.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or non-existent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The specified view name does not exist.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>RD004</td>
<td>drop failed due to referencing view</td>
<td>Dropping a view with the RESTRICT option and a referencing view exists.</td>
</tr>
</tbody>
</table>
### 10.1.18 The GRANT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to grant the privilege. This may correspond to accessing the target object or to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to grant the privilege. This may correspond to accessing the target object or to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to grant the privilege. This may correspond to accessing the target object or to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA003</td>
<td>relation access violation</td>
<td>The user does not have the DAC table column privileges to grant the privilege. This always corresponds to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA003</td>
<td>relation access violation</td>
<td>The user does not have the DAC view column privileges to grant the privilege. This always corresponds to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or nonexistent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or nonexistent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td>RR005</td>
<td>named relation is inaccessible or nonexistent</td>
<td>The specified table name does not exist.</td>
</tr>
<tr>
<td>RC007</td>
<td>named view is inaccessible or nonexistent</td>
<td>The specified view name does not exist.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Error Message</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VC004</td>
<td>column name specified not in original</td>
<td>The specified table/view column name does not exist.</td>
</tr>
<tr>
<td>PA005</td>
<td>session label and object label must be equal</td>
<td>The session sensitivity label and the object sensitivity label are not equal.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
<tr>
<td>2D504</td>
<td>item inserted by older transaction, but not committed</td>
<td>The transaction read an uncommitted object and the transaction consistency level does not allow it.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>91001</td>
<td>the privilege being modified does not exist</td>
<td>Attempting to grant the GRANT OPTION on a privilege not held by the grantee.</td>
</tr>
<tr>
<td>91010</td>
<td>existing NULL privilege would override specified privilege(s)</td>
<td>Attempting to grant the GRANT OPTION on the NULL privilege.</td>
</tr>
<tr>
<td>91004</td>
<td>invalid user name specified</td>
<td>The named user grantee is not a valid user on the server host.</td>
</tr>
<tr>
<td>91005</td>
<td>invalid group name specified</td>
<td>The named group grantee is not a valid group on the server host.</td>
</tr>
<tr>
<td>91011</td>
<td>specified privilege already exists</td>
<td>Attempting to grant a privilege or the GRANT OPTION and the grantee already holds it.</td>
</tr>
<tr>
<td>91015</td>
<td>user is not the grantor of the privilege being modified</td>
<td>Attempting to grant the GRANT OPTION but the grantor of the original privilege is not the current grantor. Only the user who granted to original privilege may subsequently add the GRANT OPTION.</td>
</tr>
<tr>
<td>PW006</td>
<td>user does not have the authorization to perform the command</td>
<td>The GRANT ASSUME was specified but the user does not have the proper authorization.</td>
</tr>
<tr>
<td>91013</td>
<td>target user not a proper candidate for GRANT ASSUME</td>
<td>The GRANT ASSUME was specified and grantee’s parent is RUBIX.</td>
</tr>
<tr>
<td>91013</td>
<td>target user not a proper candidate for GRANT ASSUME</td>
<td>The GRANT ASSUME was specified and grantee has no specified privilege.</td>
</tr>
<tr>
<td>91013</td>
<td>target user not a proper candidate for GRANT ASSUME</td>
<td>The GRANT ASSUME was specified and the grantee’s privilege is from the assumee.</td>
</tr>
<tr>
<td>91013</td>
<td>target user not a proper candidate for GRANT ASSUME</td>
<td>The GRANT ASSUME was specified and the assumee’s privilege is from the grantee.</td>
</tr>
<tr>
<td>91013</td>
<td>target user not a proper candidate for GRANT ASSUME</td>
<td>The GRANT ASSUME was specified and the grantee is the grant parent of the assumee.</td>
</tr>
</tbody>
</table>
## 10.1.19 The REVOKE Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to revoke the privilege. This may correspond to accessing the target object or to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to revoke the privilege. This may correspond to accessing the target object or to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to revoke the privilege. This may correspond to accessing the target object or to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA003</td>
<td>relation access violation</td>
<td>The user does not have the DAC table column privileges to revoke the privilege. This corresponds to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>RA003</td>
<td>relation access violation</td>
<td>The user does not have the DAC view column privileges to revoke the privilege. This corresponds to needing the privilege with the GRANT OPTION on the target object.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or nonexistent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or nonexistent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td>RR005</td>
<td>named relation is inaccessible or nonexistent</td>
<td>The specified table name does not exist.</td>
</tr>
<tr>
<td>RC007</td>
<td>named view is inaccessible or nonexistent</td>
<td>The specified view name does not exist.</td>
</tr>
<tr>
<td>VC004</td>
<td>column name specified not in original</td>
<td>The specified table/view column name does not exist.</td>
</tr>
<tr>
<td>PA005</td>
<td>session label and object label must be equal</td>
<td>The session sensitivity label and the object sensitivity label are not equal.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
<tr>
<td>2D504</td>
<td>item inserted by older transaction, but not committed</td>
<td>The transaction read an uncommitted object and the transaction consistency level does not allow it.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>The transaction is read only.</td>
</tr>
<tr>
<td>91001</td>
<td>the privilege being modified does not exist</td>
<td>Attempting to revoke a privilege or the GRANT OPTION and the revokee does not hold it.</td>
</tr>
</tbody>
</table>
91015 | user is not the grantor of the privilege being modified | Attempting to revoke a privilege or the GRANT OPTION but the grantor of the original privilege is not the current revoker. Only the user who granted to original privilege may subsequently revoke the privilege or the GRANT OPTION.

91010 | existing NULL privilege would override specified privilege(s) | Attempting to revoke the GRANT OPTION on the NULL privilege.

2B000 | dependent privilege descriptors still exist | The REVOKE RESTRICT was attempted and a dependent privilege exists.

PW006 | user does not have the authorization to perform the command | The REVOKE ASSIGN was specified and the user does not have the proper authorization.

91012 | target user not a proper candidate for REVOKE ASSIGN | The REVOKE ASSIGN was specified and revokee or assignee is a group or PUBLIC.

91012 | target user not a proper candidate for REVOKE ASSIGN | The REVOKE ASSIGN was specified and the assignee has the specified privilege.

91012 | target user not a proper candidate for REVOKE ASSIGN | The REVOKE ASSIGN was specified and the grantor is the assignee.

91012 | target user not a proper candidate for REVOKE ASSIGN | The REVOKE ASSIGN was specified and grantor of assignee’s privilege is the revokee.

91012 | target user not a proper candidate for REVOKE ASSIGN | The REVOKE ASSIGN was specified and the grantor of the revokee’s privilege is the assignee.

### 10.1.20 The SET CATALOG Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to set the catalog.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to set the catalog.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to set the catalog (default_schema).</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
</tbody>
</table>

### 10.1.21 The SET SCHEMA Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to set the schema.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to set the schema.</td>
</tr>
</tbody>
</table>
RA002 | schema access violation | The user does not have the DAC schema privileges to set the schema.
---|---|---
3D001 | named catalog is inaccessible or non-existent | The specified catalog name does not exist.
3F001 | named schema is inaccessible or non-existent | The specified schema name does not exist.

### 10.1.22 The SELECT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to perform the select.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to perform the select.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to perform the select.</td>
</tr>
<tr>
<td>RA003</td>
<td>relation access violation</td>
<td>The user does not have the DAC table/view column privileges to perform the select. This may occur from a table/view column directly referenced by the command or from a table column that is referenced by a view.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or non-existent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or non-existent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td>RR005</td>
<td>named relation is inaccessible or non-existent</td>
<td>The specified table/view name does not exist.</td>
</tr>
<tr>
<td>VC004</td>
<td>column name specified not in original</td>
<td>The specified table/view column name does not exist.</td>
</tr>
<tr>
<td>22501</td>
<td>coercion between the specified types not defined</td>
<td>Cannot cast between data types. This may be an explicit or implicit cast.</td>
</tr>
<tr>
<td>VC007</td>
<td>number of columns in source and target differ</td>
<td>Mismatch in number of columns.</td>
</tr>
<tr>
<td>VP019</td>
<td>operation no defined for type</td>
<td>Operation cannot be applied to expression (type error; e.g., add two chars).</td>
</tr>
<tr>
<td>VP020</td>
<td>mathematical overflow or underflow</td>
<td>Mathematical overflow or underflow.</td>
</tr>
<tr>
<td>22012</td>
<td>division by zero</td>
<td>Mathematical divide by zero.</td>
</tr>
</tbody>
</table>

### 10.1.23 The DELETE Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to perform the delete.</td>
</tr>
</tbody>
</table>
RA008  catalog access violation  The user does not have the DAC catalog privileges to perform the delete.
RA002  schema access violation  The user does not have the DAC schema privileges to perform the delete.
RA003  relation access violation  The user does not have the DAC table/view column privileges to perform the delete. This may occur from a table/view column directly referenced by the command or from a table column that is referenced by a view.
3D001  named catalog is inaccessible or nonexistent  The specified catalog name does not exist.
3F001  named schema is inaccessible or nonexistent  The specified schema name does not exist.
RR005  named relation is inaccessible or nonexistent  The specified table/view name does not exist.
VC004  column name specified not in original  The specified table/view column name does not exist.
SS002  session label and record label must be equal  Session sensitivity label and row sensitivity label are not equal.
VU003  view is not updateable  The user attempted to delete a row from a view that is not updateable.
22501  Coercion between the specified types not defined.  Cannot cast between data types. This may be an explicit or implicit cast.
VC007  number of columns in source and target differ  Mismatch in number of columns.
VP019  operation no defined for type  Operation cannot be applied to expression (type error; e.g., add two chars).
VP020  mathematical overflow or underflow  Mathematical overflow or underflow.
22012  division by zero  Mathematical divide by zero.
23000  integrity constraint violation  Foreign key constraint integrity violation
23000  integrity constraint violation  Check constraint integrity violation.
2D501  item has been read by a younger transaction  The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.
2D504  item inserted by older transaction, but not committed  The transaction read an uncommitted object and the transaction consistency level does not allow it.
25500  read only transaction  Transaction is read only.

10.1.24 The INSERT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to perform the insert.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to perform the insert.</td>
</tr>
<tr>
<td>RA002</td>
<td>schema access violation</td>
<td>The user does not have the DAC schema privileges to perform the insert.</td>
</tr>
<tr>
<td>RA003</td>
<td>relation access violation</td>
<td>The user does not have the DAC table/view column privileges to perform the insert. This may occur from a table/view column directly referenced by the command or from a table column that is referenced by a view.</td>
</tr>
<tr>
<td>3D001</td>
<td>named catalog is inaccessible or nonexistent</td>
<td>The specified catalog name does not exist.</td>
</tr>
<tr>
<td>3F001</td>
<td>named schema is inaccessible or nonexistent</td>
<td>The specified schema name does not exist.</td>
</tr>
<tr>
<td>RR005</td>
<td>named relation is inaccessible or nonexistent</td>
<td>The specified table/view name does not exist.</td>
</tr>
<tr>
<td>VC004</td>
<td>column name specified not in original</td>
<td>The specified table/view column name does not exist.</td>
</tr>
<tr>
<td>VU003</td>
<td>view is not updateable</td>
<td>The user attempted to delete a row from a view that is not updateable.</td>
</tr>
<tr>
<td>22501</td>
<td>Coercion between the specified types not defined.</td>
<td>Cannot cast between data types. This may be an explicit or implicit cast.</td>
</tr>
<tr>
<td>VC007</td>
<td>number of columns in source and target differ</td>
<td>Mismatch in number of columns.</td>
</tr>
<tr>
<td>VP019</td>
<td>operation no defined for type</td>
<td>Operation cannot be applied to expression (type error; e.g., add two chars).</td>
</tr>
<tr>
<td>VP020</td>
<td>mathematical overflow or underflow</td>
<td>Mathematical overflow or underflow.</td>
</tr>
<tr>
<td>22012</td>
<td>division by zero</td>
<td>Mathematical divide by zero.</td>
</tr>
<tr>
<td>23000</td>
<td>integrity constraint violation</td>
<td>Not null constraint integrity violation.</td>
</tr>
<tr>
<td>23000</td>
<td>integrity constraint violation</td>
<td>Foreign key constraint integrity violation.</td>
</tr>
<tr>
<td>23000</td>
<td>integrity constraint violation</td>
<td>Check constraint integrity violation.</td>
</tr>
<tr>
<td>23000</td>
<td>integrity constraint violation</td>
<td>Unique constraint integrity violation.</td>
</tr>
<tr>
<td>23000</td>
<td>integrity constraint violation</td>
<td>Primary key constraint integrity violation.</td>
</tr>
<tr>
<td>2D501</td>
<td>item has been read by a younger transaction</td>
<td>The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.</td>
</tr>
<tr>
<td>2D504</td>
<td>item inserted by older transaction, but not committed</td>
<td>The transaction read an uncommitted object and the transaction consistency level does not allow it.</td>
</tr>
<tr>
<td>25500</td>
<td>read only transaction</td>
<td>Transaction is read only.</td>
</tr>
</tbody>
</table>

**10.1.25 The UPDATE Statement**

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA001</td>
<td>database access violation</td>
<td>The user does not have the DAC database privileges to perform the insert.</td>
</tr>
<tr>
<td>RA008</td>
<td>catalog access violation</td>
<td>The user does not have the DAC catalog privileges to perform the insert.</td>
</tr>
</tbody>
</table>
### 10.1.26 The COMMIT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT010</td>
<td>no current transaction</td>
<td>There is no current transaction to commit.</td>
</tr>
</tbody>
</table>

The user does not have the DAC schema privileges to perform the insert.

The user does not have the DAC table/view column privileges to perform the insert. This may occur from a table/view column directly referenced by the command or from a table column that is referenced by a view.

The specified catalog name does not exist.

The specified schema name does not exist.

The specified table/view name does not exist.

The specified table/view column name does not exist.

The user attempted to delete a row from a view that is not updateable.

Session sensitivity label and row sensitivity label are not equal.

Cannot cast between data types. This may be an explicit or implicit cast.

Mismatch in number of columns.

Operation cannot be applied to expression (type error; e.g., add two chars).

Mathematical overflow or underflow.

Mathematical divide by zero.

Not null constrain integrity violation.

Foreign key constraint integrity violation

Check constraint integrity violation.

Unique constraint integrity violation.

Primary key constraint integrity violation.

NULL value in a unique key.

The transaction attempted to perform a write that should have been seen by a transaction that previously read the object.

The transaction read an uncommitted object and the transaction consistency level does not allow it.

Transaction is read only.
### 10.1.27 The ROLLBACK Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT010</td>
<td>no current transaction</td>
<td>There is no current transaction to rollback.</td>
</tr>
</tbody>
</table>

### 10.1.28 The SET TRANSACTION Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25100</td>
<td>operation not permitted while transaction is active</td>
<td>There is a currently active transaction.</td>
</tr>
</tbody>
</table>

### 10.1.29 The DECLARE SAVEPOINT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT010</td>
<td>no current transaction</td>
<td>There is no current transaction to declare a savepoint for.</td>
</tr>
<tr>
<td>RS002</td>
<td>maximum number of savepoints exceeded</td>
<td>The maximum number of savepoints for the current transaction has been reached.</td>
</tr>
</tbody>
</table>

### 10.1.30 The RELEASE SAVEPOINT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT010</td>
<td>no current transaction</td>
<td>There is no current transaction to release a savepoint for.</td>
</tr>
<tr>
<td>RS001</td>
<td>named savepoint does not exist</td>
<td>The specified savepoint does not exist.</td>
</tr>
</tbody>
</table>

### 10.1.31 The ROLLBACK TO SAVEPOINT Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT010</td>
<td>no current transaction</td>
<td>There is no current transaction to rollback the savepoint of.</td>
</tr>
<tr>
<td>RS001</td>
<td>named savepoint does not exist</td>
<td>The specified savepoint does not exist.</td>
</tr>
</tbody>
</table>
## 10.1.32 The ALTER SESSION SET LABEL Statement

<table>
<thead>
<tr>
<th>SQL Code</th>
<th>Printed Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS010</td>
<td>new session label must be dominated by the clearance label</td>
<td>The new session is not dominated by the user’s operating system clearance.</td>
</tr>
<tr>
<td>PW006</td>
<td>user does not have the authorization to perform the command</td>
<td>The user does not have the proper authorization to perform the command.</td>
</tr>
<tr>
<td>25100</td>
<td>operation not permitted while transaction is active</td>
<td>A transaction currently exists.</td>
</tr>
</tbody>
</table>
11. SYNTAX SUMMARY

This summary uses the notation defined in Chapter 3. In addition, the symbol \( ::= \) indicates that the syntactic element on its left is defined in the terms described on its right. Footnotes appear when they are more helpful than a mathematical description. The main text contains additional syntactic and semantics rules that are essential to the definitions.

11.1 Common Elements

\[\text{approximate-numeric-literal} ::= \text{mantissa}\text{exponent}\]
\[\text{mantissa} ::= \text{exact numeric-literal}\]
\[\text{exponent} ::= [ + | - ] \text{unsigned-integer}\]
\[\text{approximate-numeric-type} ::= \text{FLOAT \{\text{precision}\}}\]
\[\quad \| \text{DOUBLE PRECISION} \| \text{REAL}\]
\[\text{base-table-identifier} ::= \text{user-defined-identifier}\]
\[\text{base-table-name} ::= [ \text{user-name} . ] \text{base-table-identifier}\]
\[\text{between-predicate} ::= \text{expression \{NOT\} BETWEEN expression}
\quad \text{AND expression}\]
\[\text{character}^1\]
\[\text{character-string-literal} ::= \text{"{character}\ldots\"}\]
\[\text{character-string type} ::= \]
\[\quad \text{CHARACTER \{\text{length}\}}\]
\[\quad \| \text{CHARACTER VARYING}\{\text{length}\}\]
\[\quad \| \text{VARCHAR}\{\text{length}\}\]
\[\text{length} ::= \text{unsigned-integer}\]
\[\text{column-identifier} ::= \text{user-defined-name}\]
\[\text{column-name} ::= [ \{\text{table-name} | \text{correlation-name}\}.] \text{column-identifier}\]
\[\text{comparison-operator} ::= < | > | \leq | \geq | = | \neq | \lt | \gt\]
\[\text{comparison-predicate} ::= \]
\[\quad \text{expression comparison-operator \{ expression \| \text{(sub-query)}\}\}
\[\text{correlation-name} ::= \text{user-defined-name}\]

\(^1\) Any character in the character set except the newline indication.
cursor-name : ::= user-defined-name

data-type : ::= 
  character-string-type 
  | exact-numeric-type 
  | approximate-numeric-type 

digit : ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 

dynamic-parameter : ::= ? 

exact-numeric-literal : ::= 
  [ + | - ] {unsigned-integer [ . unsigned-integer] 
  | unsigned-integer. 
  | . unsigned-integer } 

exact-numeric-type : ::= 
  DECIMAL [(precision [, scale] ) ] 
  | INTEGER 
  | SMALLINT 
  | NUMERIC [(precision [, scale] ) ] 

  precision : ::= unsigned-integer 
  scale : ::= unsigned-integer 

expression : ::= term | expression { + | - } term 

term : ::= factor | term { * | / } factor 

factor : ::= [ + | - ] primary 

primary : ::= 
  column-name 
  | dynamic-parameter 
  | host-variable-reference 
  | literal 
  | set-function-reference 
  | USER 
  | (expression) 

host-variable-reference : ::= 
  embedded-variable-name [ indicator-variable ] 

  embedded-variable-name : ::= : host-identifier 
  indicator-variable : ::= : host identifier 

host-identifier \(^2\) 

index-identifier : ::= user-defined-name 

\(^2\) For C, any valid variable name
index-name : = [ user-name. ] index-identifier

in-predicate : =
  expression [NOT] IN { ( value {, value }.. ) | (sub-query) }

value : =
  host-variable-reference | literal | USER
  | dynamic-parameter

keyword

letter : = lower-case-letter | upper-case-letter

like-predicate : =
  column-name [NOT] LIKE pattern-value [ESCAPE escape-character ]

pattern-value : =
  character-string-literal
  | dynamic-parameter
  | host-variable-reference
  | USER

escape-character : =
  character-string-literal
  | dynamic-parameter
  | host-variable-reference
  | USER

literal : = character-string-literal | numeric-literal

lower-case-letter : =
  a | b | c | d | e | f | g | h | i | j | k | l | m
  | n | o | p | q | r | s | t | u | v | w | x | y | z

null-predicate : = column-name IS [NOT] NULL

numeric-literal : = exact-numeric-literal
  | approximate-numeric-literal

predicate : =
  between-predicate
  | comparison-predicate
  | exists-predicate
  | in-predicate
  | like-predicate
  | null-predicate
  | quantified-predicate

query-specification : =

3 For the complete list of keywords, see page 13
SELECT [ALL | DISTANT] select-list
[WHERE search-condition]
[GROUP BY column-name [ , column-name ] . . .]
[HAVING search-condition]

search-condition  ::=  
  boolean-term [ OR search-condition ]

boolean-term  ::=  boolean-factor [ AND boolean-term ]

boolean-factor  ::=  [ NOT ] boolean primary

boolean-primary  ::=  predicate | ( search-condition )

select-list  ::=  * | select-sublist [ , select-sublist ] . . .

select-sublist  ::=  
  expression [ AS column-identifier ]
  | { table-name | correlation-name } . *

separator

set-function-reference  ::=  COUNT ( * ) | distinct-function
  | all-function

distinct-function  ::=  
  { AVG | COUNT | MAX | MIN | SUM }
  ( DISTINCT column-name )

all function  ::=  
  { AVG | MAX | MIN | SUM } ( [ ALL ] expression )

SQL-escape-clause  ::=  
  -- ( * SQL-escape-identification, extended-SQL-text * ) --

SQL-escape-identification  ::=  
  SQL-escape-ISO-clause | SQL-escape-vendor-clause

SQL-escape-ISO-clause  ::=  
  YEAR ( ISO-year ) , CONFORMANCE ( ISO-conformance )

SQL-escape-vendor-clause  ::=  
  VENDOR( vendor-identifier ) , PRODUCT ( product-identifier )

statement-identifier  ::=  user-defined-name

sub-query  ::=  
  SELECT [ ALL | DISTINCT] select-list

---

4 The blank character, the newline indication, or an SQL comment
[ WHERE search-condition ]
[ GROUP BY column-name [ , column-name] . . .]
[ HAVING search-condition]

table-identifier  ::=  user-defined-name

table-name  ::=  [ user-name. ] table-identifier

table-reference  ::=  table-name. [ correlation-name]

token  ::=  delimiter-token  |  non-delimiter-token

delimiter-token  ::=  character-string-literal
|  | ( | ) | < | > | . | :
| = | * | + | - | | | <= | ?

non-delimiter-token  ::=  keyword  |  numeric-literal  |  user-defined-name  | host-identifier

unsigned-integer  ::=  (digit) . . .

upper-case-letter  ::=  
A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

user-defined-name  ::=  letter [ digit | letter | _ ] . . .

user-name  ::=  user-defined-name

viewed-table-identifier  ::=  user-defined-name

viewed-table-name  ::=  [ user-name. ] viewed-table-identifier

11.2 Executable SQL Statements

alter-table-statement  ::=  
ALTER TABLE base-table-name
{  ADD column-identifier data-type

close-statement  ::=  CLOSE cursor-name

commit-statement  ::=  COMMIT WORK

connect-statement  ::=  
CONNECT TO server [ AS connection-name]

server  ::=  

character-string-literal | embedded-variable-name
| DEFAULT

collection-name : := character-string-literal
| embedded-variable-name

create-index-statement : :=
CREATE [ UNIQUE ] INDEX index-name ON base-table-name (column-identifier [ ASC | DESC]
[ , column-identifier [ ASC | DESC ] . . . ]

create-table-statement : :=
CREATE TABLE base-table-name-1 ( column-element
[ , column-element ] . . . )

column-element : := column-definition
| table-constraint-definition

column-definition : :=
column-identifier data type
| DEFAULT default-value ]
| column-constraint-definition
| [ , column-constraint-definition ] . . . ]

column-constraint-definition : :=
NOT NULL
| [ UNIQUE | PRIMARY KEY ]
| REFERENCES base-table-name-2
| ( column-identifier ) ]
| CHECK ( search-condition )

default-value : := literal | NULL | USER

table-constraint-definition : :=
UNIQUE ( column-identifier [ , column-identifier ] . . . )
| PRIMARY KEY ( column-identifier [ , column-identifier ] . . . )
| CHECK ( search-condition )
| FOREIGN KEY referencing-columns
REFERENCES
base-table-name-2 referenced-columns

create-view-statement : :=
CREATE VIEW viewed-table-name
[ ( column-identifier [ , column-identifier ] . . . )
As query-specification

delete-statement-positioned : :=
DELETE FROM table-name WHERE CURRENT OF cursor-name

delete-statement-searched : :=
DELETE FROM table-name [ WHERE search-condition ]
descriptor-name : ::= character-string-literal
                   | embedded-variable-name

disconnect-statement : ::= DISCONNECT [ connection-name
                            | ALL | CURRENT ]

drop-index-statement : ::= DROP INDEX index-name

drop-table-statement : ::= DROP TABLE base-table-name
                        [ CASCADE | RESTRICT ]

drop-view-statement : ::= DROP VIEW viewed-table-name
                        [ CASCADE | RESTRICT ]

eexecute-statement : ::= EXECUTE statement-identifier
                     [ using-clause]

eexecute-immediate-statement : ::= EXECUTE IMMEDIATE embedded-variable-name

fetch-statement : ::= FETCH cursor-name INTO host-variable-reference
                     [ , host-variable-reference ] . . .

field-name : ::= TYPE | LENGTH | PRECISION | SCALE | INDICATOR
              | DATA

gget-diagnosics-statement : ::= GET DIAGNOSTICS { statement-information | exception-information }

statement-information : ::= 

                        statement-information-item [ , statement-information-item ] . . .

                        statement-information-item : ::= 
                        embedded-variable-name-1 = field-name-1

exception-information-item : ::= 

                        EXCEPTION exception-number
                        exception-information-item [ , exception-information-item ] . . .

                        exception-information-item : ::= 
                        embedded-variable-name-2 = field-name-2

exception-number : ::= 

                        unsigned-integer | embedded-variable-name-3

field-name-1 : ::= NUMBER | MORE
                | ROW_COUNT

field-name-2 : ::=
grant-statement : : =
    GRANT { ALL [ PRIVILEGES ] | grant-privilege
        [ , grant-privilege ] . . . } 
    ON table-name { TO PUBLIC | user-name [ , user-name ] . . . }
    [ WITH GRANT OPTION ]

grant-privilege : : =
    DELETE
    | INSERT
    | SELECT
    | UPDATE [ ( column-identifier
        [ , column-identifier ] . . . ) ]
    | REFERENCES [ column-identifier
        [ , column-identifier ] . . . ) ]

insert-statement : : =
    INSERT INTO table-name
    [ ( column-identifier [ , column-identifier ] . . . ) ]
    { query-specification
    | VALUES ( insert-value [ , insert-value ] . . . ) }

insert-value : : =
    host-variable-reference
    | dynamic-parameter
    | literal
    | NULL
    | USER

item-number : : = unsigned-integer | embedded-variable-name

open-statement : : = OPEN cursor-name

prepare-statement : : =
    PREPARE statement-identifier FROM embedded-variable-name

revoke-statement : : =
    REVOKE { ALL | revoke-privilege [ , revoke-privilege ] . . . } 
    ON table-name
    FROM { PUBLIC | user-name [ , user-name ] . . . }
    [ CASCADE | RESTRICT ]

revoke-privilege : : =
    DELETE
    | INSERT
    | SELECT
    | UPDATE
    | REFERENCES
rollback-statement : : = ROLLBACK WORK

select-statement : : =
    SELECT [ ALL | DISTINCT ] select-list
    [ WHERE search-condition

set-connection-statement : : = SET CONNECTION server

update-statement-positioned : : =
    UPDATE table-name
    SET column-identifier = { expression | NULL}
    [ , column-identifier = { expression | NULL} ] . . .
    WHERE CURRENT OF cursor-name

update-statement-searched : : =
    UPDATE table-name
    SET column-identifier = { expression | NULL}
    [ , column-identifier = { expression | NULL} ] . . .
    [ WHERE search-condition ]

using-clause : : = using-arguments | using-descriptor

using-arguments : : =
    { USING | INTO } host-variable-reference
    [ , host-variable-reference ] . . .

using-descriptor : : =
    { USING | INTO } SQL DESCRIPTOR
descriptor-name
GLOSSARY

active connection
A current or dormant connection on which any SQL statement has been successfully executed during the current transaction.

application
The end-user program written to operate on a database. This term is used to distinguish that program from Trusted RUBIX (the database client or server) when discussing the calling interface.

authentication string
A string that is used to validate the user identifier.

base table
A table that is not a view. See also View.

byte
An octet. This specification avoids the term byte. It uses octet to refer to an amount of storage, and character to refer to a component of character-string data independent of the amount of storage.

cardinality
The cardinality of a collection is the number of objects in the collection. The cardinality of a table is its number of rows, which is the cardinality of each column.

character
A component of character-string data, which requires one or more octets of storage.

client
The component of a database application that requests data for processing.

column
A sequence of values in a table.

column attribute
An item of information about a column, such as the column length.

compilation
The process of converting a program to its executable form. This may include pre-compilation.

concurrent
Transactions begun by different programs are concurrent if they gain access to the same metadata or data and overlap in time.

connection
An association between a client and a server.
**current connection**
The connection on which a function operates.

**current user**
The user on behalf of whom the program is executed.

**cursor**
A movable pointer into a result set, by which a program gains access to its rows, one at a time.

**cursor-specification**
cursor-specification refers to the entire syntax of the cursor-specification (SELECT statement) defined in the International Standard. This does not include the SELECT . . . INTO syntax of the dynamic FETCH statement of embedded SQL.

**data**
The data in a database is every value in every active table, as opposed to the metadata.

**database**
The set of information contained at a server, organized into metadata and data.

**data type**
A set of possible values.

**deadlock**
A condition under which an activity may not proceed because it is dependent on exclusive resources that are owned by some other activity, which in turn is dependent on exclusive resources in use by the original activity.

**default value**
The value to be given to a column when a program inserts a new row into a table without specifying a value for that column.

**degree**
The degree of a table is its number of columns, which is the cardinality of each row.

**delimited identifier**
A double-quoted string that identifies an object within a database. All text within the double quotes is assumed to be part of the identifier and is interpreted literally.

**deprecated feature**
A term used in discussing Open Group-compliance.

**descriptor**
A data structure inside Trusted RUBIX that contains information about one or more columns.

**dormant connection**
A connection to a database that the application has established previously and has not disconnected, but is not using in the current database activity.

**dynamic SQL**
An execution model in which the actual SQL statement is not known until run time.
dynamic parameter
   A position in an SQL statement that represents a literal value the application must define before executing the statement.

dynamic argument
   The value that is bound to a dynamic parameter of an SQL statement.

embedded host variable
   A variable in the host language that an embedded SQL statement uses to either obtain a value from, or convey a value to, the program.

embedded SQL
   An execution model in which SQL statements are included as part of the compilable source code of a program.

error condition
   A condition in the execution of a function that indicates failure to perform the requested operation. Contrast warning condition.

error status
   A return code from a function that reports an error condition.

expression
   A combination of values on which database queries can be based.

foreign key
   Columns in one table that specify the valid values for one or more columns in another table.

generic data type
   A general category of data types, within which are defined several named data types with specific attributes.

grantable
   A privilege is grantable if a user who holds that privilege on a table can grant it to, or revoke it from, any user.

identifier
   A text string, whose content is subject to certain rules, that identifies an object in a database, such as a cursor, table or user.

index
   A data structure that behaves like an ordered list of pointers to the rows of a table.

indicator variable
   An integer host-language variable that indicates whether an associated variable is the null value and whether it contains a truncated character string.

integrity constraint
   One of several techniques that constrain the values in a base table.
isolation

A transaction attribute that specifies the degree to which the operations on metadata or data interact with the effects of concurrent transactions.

literal

A specific syntactical form in SQL that represents a value.

logical representation

The display form of a value in the syntax of SQL.

metadata

The definitions of all active base tables, viewed tables, indexes, privileges and user names in a database.

multi-set

An unordered collection of objects that are not necessarily distinct.

named data type

A data type, identified by name, with specific attributes.

null value

A value, distinct from all non-null values, representing a case where an actual value is not known or not applicable.

octet

A discrete unit of memory accommodating at least 8 bits. An octet is the unit used to measure the memory requirements of a character.

open connection

A connection that is either active or dormant.

optional feature

A term used in discussing Open Group-compliance.

parameter number

The ordinal position of a dynamic parameter.

pending transaction

A transaction that has not been completed.

phantom

A row that a program retrieves, that it did not retrieve before using the same search condition, caused by action of a concurrent transaction.

positioned update

An UPDATE statement in SQL that operates on data identified by a cursor.

positioned delete

A DELETE statement in SQL that operates on data identified by a cursor.

precision
The number of bits or digits representing a number.

**predicate**

A syntactic element of SQL that represents an assertion that is true or false.

**privilege**

Authorization of a user to perform a given category of action on a specified object.

**program**

The host application program that uses SQL to access data.

**pseudo-column**

A column in a table other than those specified by an application using the CREATE TABLE or ALTER TABLE statements of embedded SQL. Pseudo-columns are undefined by this document. An example of a pseudo-column is a row identifier.

**qualification**

An applicant’s optional use of a schema name to make unique the reference to a database object.

**RDA**

(Remote Database Access) A mechanism by which clients and servers communicate to carry out application database requests; specified in the Open Group RDA specification.

**read-only**

A table or a query-specification that is not updatable; or a transaction that is not read-write.

**read-write**

A transaction access mode that indicates that the transaction is allowed to perform updates to data and metadata.

**result set**

A derived table produced as the result of SQL statement or ODBC function execution. It is this table from which rows are fetched.

**row**

A non-empty sequence of values in a table.

**scale**

The number of bits or digits representing the fractional part of a number.

**schema**

A collection of related objects in a database. If a schema contains base tables, it also contains any indexes that are defined on the base tables.

**SELECT statement**

See cursor-specification.

**sequence**

An ordered collection of objects that are not necessarily distinct.

**serializable**
An attribute of concurrent transactions in Open Group-compliant SQL implementations, which provides that the overall result is the same as some serial sequence of the same transactions.

**server**

The component of a database application that provides data on request.

**set**


**SQL**

(Structured Query Language) A standard language for codifying database queries and updates as text, specified in this document.

**status record**

A record of some error or warning event maintained by Trusted RUBIX. The application can retrieve elements of the status record using GetDiagRec() or GetDiagField().

**stored routine**

A routine (procedure or function) that resides on the server, which the client can invoke.

**system view**

A predefined read-only view that provides schema information.

**table**

A named collection of values, arranged into rows and columns.

**TOE**

Target of Evaluation

**transaction**

An action or series of actions that are atomic with respect to recovery and concurrency. Atomicity means that either all actions take permanent effect or none do.

**Transitional SQL**

A set of SQL features defined by FIPS 127-2.

**two-phase commit**

A protocol for ensuring that a transaction is atomic (see transaction) among all sites (for example, servers) where the actions take place.

**type convertible**

Data types whose values may be assigned to objects of another data type.

**unbound column**

A column value whose associated descriptor record is not bound.

**undefined**

A term used in discussing Open Group-compliance.

**unknown**

One of the three logical truth values (along with true and false).
**updatable**

1. An updatable table is a table that can be modified.
2. An updatable query-specification is one that contains valid syntax to be used in the modification of a table.

**user**

An entity, distinguished from other users for security purposes, that is entitled to use the database.

**user identifier**

The string that represents the security identity under whose auspices the connection is established.

**value**

A data item in the database.

**view**

A derived table with a name that is defined in terms of other tables. Conceptually, a base table is a table that is stored; a view is a table that is computed.

**viewed table**

A view.

**warning condition**

A condition in the execution of a function that does not imply a failure to perform the requested operation but provides the application with additional information. Contrast error condition.

**warning status**

A return code from a function that reports a warning condition.
INDEX

A
Absence of phantoms ................................................. 9
Access control ....................................................... 7
Access mode ......................................................... 9
Active connection .................................................. 95
Active transaction ................................................... 8
ADD COLUMN constraint violation ................................. 55
Administrative action disconnection ............................... 95
ALL
default for set function ............................................. 37
following UNION ..................................................... 47
in set function ......................................................... 36
quantified predicate .................................................. 41
ALTER SESSION SET LABEL ........................................ 92
ALTER TABLE .......................................................... 52
constraint violation .................................................... 55
Ambiguity in date/time ............................................... 20
ANY, in quantified predicate ........................................ 41
Application .......................................................... 1, 6, 98
Application Administrator .......................................... 6, 98
Application User ..................................................... 6, 98
Approximate numeric value ........................................ 19
Arithmetic
data type of result ............................................... 37
error .................................................................. 32
on date/time .......................................................... 32
operator ................................................................. 31
Array of char (C) ....................................................... 28
ASC (ascending) in CREATE INDEX ............................... 58
Assignment ............................................................ 27
and CAST function .................................................... 38
of null value ........................................................... 30
within generic data type ............................................. 16
Asterisk .................................................................. 49
Atomicity ................................................................. 8
Attribute
max limits ................................................................ 104
of column ............................................................... 2
of transaction ........................................................... 4
Authentication information is connection context .............. 95
Authorizations .......................................................... 8
AVG ......................................................................... 37
B
Base table ................................................................. 2
creating ................................................................. 60
format of qualification ............................................... 14
BETWEEN predicate ................................................... 41
Binding client to server ............................................... 4
Binding operator/operand
in arithmetic expression ............................................. 32
in logical expression .................................................. 40
BIT ......................................................................... 16, 17
Bit string literal ......................................................... 24
BIT VARYING ........................................................... 16, 17
BIT_LENGTH ............................................................ 34
Blank
significance of trailing ............................................... 29
valid in delimited name .............................................. 12
Blank padding
during string assignment ........................................... 29
in comparison ........................................................... 29
BLOB ................................................................. 18, 30, 40
Body of user-defined name, in quotes ............................ 12
C
C string storage .......................................................... 28
Calendar discontinuity ................................................ 32
Capitalization
functions ................................................................. 34
implicit ................................................................. 12
in delimited name .................................................... 12
Cardinality ............................................................... 1
Cartesian product ...................................................... 44
CASCADE
in ALTER TABLE ...................................................... 52
in DROP CATALOG .................................................... 65
in DROP SCHEMA ..................................................... 67
in DROP TABLE ........................................................ 68
in DROP VIEW ........................................................ 69
in REVOKE ............................................................. 73
Case sensivity ........................................................... 14
in delimited name .................................................... 12
CAST
in generic data type .................................................. 16
Catalogs .................................................................. 5
CHAR, synonym for CHARACTER ............................... 18
CHAR_LENGTH .......................................................... 34
CHARACTER ............................................................ 18
Character
casting from ........................................................... 39
casting to .............................................................. 39
Character set ............................................................ 1
Character string assignment of ................................... 28
Character string literal ................................................ 25
CHARACTER VARYING ............................................... 18
CHECK constraint
in query specification ...........................................44
prohibited in some queries ...........................................45
treatment of null value ...........................................31
Grouped table ...........................................3
Grouped view ...........................................3

H
HAVING
effect on upatability ...........................................45
in query specification ...........................................44
prohibited in some queries ...........................................45
Hidden columns ...........................................39
Hidden fields ...........................................39
Hyphen ...........................................11

I
Implementation note unique constraint ..............54
IN predicate ...........................................41
Index ...........................................3
dropping ...........................................66
format of qualification ...........................................13
Indicator variables
  indicating null value ...........................................43
  may be set in numeric assignment ...........................................28
Inherited data type ...........................................23
Initial state connection ...........................................94
Initial value
  added by ALTER TABLE ...........................................53
  of column ...........................................53
INSERT privilege ...........................................7, 71, 73
INSERT statement ...........................................3, 78
INT, synonym for INTEGER ...........................................19
INTEGER ...........................................19
Integrity constraint ...........................................3
INTERVAL ...........................................21
Interval
  arithmetic ...........................................32
  assignment ...........................................29
  by subtracting date/time ...........................................32
  casting from ...........................................39
  comparison ...........................................29
  literal ...........................................26
  qualifier ...........................................21
Invalid characters, valid in delimited name ........12
IS [ NOT] NULL ...........................................43
Isolation level ...........................................9

K
Key, foreign ...........................................3
Keyword ...........................................14
  additional reserved words ...........................................16
  case-sensitivity ...........................................12
valid in delimited name ...........................................12

L
Language structure ...........................................11
Large Objects (LOBs) ...........................................106
LEFT OUTER JOIN ...........................................47
Legal time ...........................................20
Length
  of CAST result ...........................................39
  of concatenation result ...........................................34
  optional for CHARACTER ...........................................18
LIKE predicate ...........................................42
Literal ...........................................1, 24
  and pseudo-literal ...........................................26
  bit string ...........................................24
  character ...........................................25
  date/time ...........................................25
  interval ...........................................26
  numeric ...........................................25
Logical operator precedence ...........................................40
Logical representation ...........................................1
Long strings ...........................................34
Loss of significant digit ...........................................24
LOWER ...........................................34
Lowercase in delimited name ...........................................12

M
Mantissa ...........................................19
MAX ...........................................37
Metadata ...........................................5
  modification by data definition statement ...........................................52
  read atomically ...........................................10
MIN ...........................................37
Multiple references to the same table ...........................................50
Multiple server ...........................................4
Multiple servers ...........................................94
Multi-set ...........................................1

N
Name
  delimited ...........................................12
  of UNION column ...........................................48
  restrictions ...........................................106
  user-defined ...........................................12
Name of user ...........................................5
Named data type ...........................................16
Namespace of objects in schema ...........................................5
NATURAL JOIN ...........................................46
Nested sub-query ...........................................49
Newline indication use as separator ...........................................11
Non-delimiter token ...........................................12
NOT NULL violation by ALTER TABLE ...........................................54
<table>
<thead>
<tr>
<th>Notation</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL byte</td>
<td>28</td>
</tr>
<tr>
<td>NULL predicate</td>
<td>43</td>
</tr>
<tr>
<td>NULL privilege</td>
<td>7, 70, 73</td>
</tr>
<tr>
<td>Phantom, absence of</td>
<td>9</td>
</tr>
<tr>
<td>Pluggable Authentication Module (PAM)</td>
<td>6, 98</td>
</tr>
<tr>
<td>POSITION</td>
<td>34</td>
</tr>
<tr>
<td>Position of cursor as connection context</td>
<td>95</td>
</tr>
<tr>
<td>Precedence</td>
<td>31</td>
</tr>
<tr>
<td>of arithmetic operator</td>
<td>40</td>
</tr>
<tr>
<td>of logical operator</td>
<td>40</td>
</tr>
<tr>
<td>Precision</td>
<td>21</td>
</tr>
<tr>
<td>optional for date/time types</td>
<td>19</td>
</tr>
<tr>
<td>optional for numeric types</td>
<td>19</td>
</tr>
<tr>
<td>Pre-compilation</td>
<td>1</td>
</tr>
<tr>
<td>Predicate</td>
<td>41</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>41</td>
</tr>
<tr>
<td>comparison</td>
<td>41</td>
</tr>
<tr>
<td>EXISTS</td>
<td>43</td>
</tr>
<tr>
<td>IN predicate</td>
<td>41</td>
</tr>
<tr>
<td>LIKE</td>
<td>42</td>
</tr>
<tr>
<td>NULL</td>
<td>43</td>
</tr>
<tr>
<td>OVERLAPS</td>
<td>43</td>
</tr>
<tr>
<td>quantified comparison</td>
<td>41</td>
</tr>
<tr>
<td>PRIMARY KEY</td>
<td>55</td>
</tr>
<tr>
<td>constraint on column</td>
<td>31</td>
</tr>
<tr>
<td>treatment of null value</td>
<td>31</td>
</tr>
<tr>
<td>Privilege</td>
<td>7</td>
</tr>
<tr>
<td>catalog</td>
<td>7</td>
</tr>
<tr>
<td>database</td>
<td>7</td>
</tr>
<tr>
<td>relation</td>
<td>7</td>
</tr>
<tr>
<td>schema</td>
<td>7</td>
</tr>
<tr>
<td>view</td>
<td>7</td>
</tr>
<tr>
<td>Pseudo-literal</td>
<td>26</td>
</tr>
<tr>
<td>as column DEFAULT value</td>
<td>55</td>
</tr>
<tr>
<td>prohibited from CHECK clause</td>
<td>63</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>8</td>
</tr>
<tr>
<td>authorization</td>
<td>71</td>
</tr>
<tr>
<td>granting rights to</td>
<td>71</td>
</tr>
<tr>
<td>Qualification</td>
<td>14</td>
</tr>
<tr>
<td>default schema</td>
<td>14</td>
</tr>
<tr>
<td>of table name</td>
<td>14</td>
</tr>
<tr>
<td>Qualified JOIN</td>
<td>46</td>
</tr>
<tr>
<td>Qualifier, interval</td>
<td>21</td>
</tr>
<tr>
<td>Qualifying row/group</td>
<td>40</td>
</tr>
<tr>
<td>Quantified predicate use of sub-query</td>
<td>49</td>
</tr>
<tr>
<td>Query-specification</td>
<td>44</td>
</tr>
<tr>
<td>Quote</td>
<td>12</td>
</tr>
<tr>
<td>R</td>
<td>32</td>
</tr>
<tr>
<td>Range, exceeding</td>
<td>6, 98</td>
</tr>
<tr>
<td>RDBMS User</td>
<td>10</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>7, 70, 73</td>
</tr>
</tbody>
</table>