APPENDIX C

SQL Syntax Summary

This appendix contains a summary of SQL syntax used throughout this book. The first table (Table C.1) describes SQL statements, arranged alphabetically by command. The notation is as follows:

- Keywords that must be typed exactly as they appear are in uppercase characters, such as REFERENCES.
- Parts of commands that are determined by the user appear in italics and name the item that must be supplied, such as *table_name*.
- Optional portions of a command are surrounded by brackets ([and]).
- Portions of commands that form a single clause are grouped within braces ({ and }).
- Sets of options from which you choose one or more are separated by vertical lines (|).
- Portions of commands that may be repeated as needed are followed by an ellipsis (…).

Table C.1
SQL Statements

Allocate space for a descriptor area for a dynamic SQL statement
```
ALLOCATE DESCRIPTOR descriptor_name
        [ WITH MAX number_of_parameters ]
```
Change the specifications of a domain
```
ALTER DOMAIN domain_name
        { SET DEFAULT default_value }
        | { DROP DEFAULT }
        | { ADD constraint_definition_clause }
        | { DROP CONSTRAINT constraint_name }
```
Change the specifications of a table
```
ALTER TABLE table_name
    { ADD [COLUMN] column_defintion }
    | { ALTER [COLUMN]
         {SET DEFAULT default_value }
         | { DROP DEFAULT }
         | { DROP [COLUMN] column_name RESTRICT | CASCADE }
    | { ADD table_constraint_definition_clause }
    | { DROP CONSTRAINT constraint_name RESTRICT | CASCADE }
```
Declare host language variables for use in an embedded SQL statement
```
BEGIN DECLARE SECTION
        Declarations
END DECLARE SECTION
```
Close an embedded SQL cursor
```
CLOSE cursor_name
CLOSE cursor_name
```

Commit a transaction, making its changes permanent

```
COMMIT [ WORK ]
```

Connect to a database, specify its cluster, catalog, and schema if necessary

```
CONNECT TO {cluster.catalog.schema.database_name
      { [ AS connection_name ] }
      { [ USER user_name
        | DEFAULT ] }
```

Create an assertion, a constraint that is not attached to a specific table

```
CREATE ASSERTION assertion_name
     CHECK ( check_predicate )
          [ { INITIALLY DEFERRED } | { INITIALLY IMMEDIATE } ]
          [ DEFERRABLE | { NOT DEFERRABLE } ]
```

Create a domain

```
CREATE DOMAIN domain_name
     [ AS ] data_type
          [ DEFAULT default_value ]
          CHECK ( check_clause )
          { [ INITIALLY DEFERRED ] | [ INITIALLY IMMEDIATE ] }
          [ DEFERRABLE | { NOT DEFERRABLE } ]
```

Define a method for a UDT

```
CREATE METHOD method_name FOR UDT_name
BEGIN
     // body of method
END
```

Create an index

```
CREATE INDEX index_name ON table_name (index_key_column_list)
```

Note: Indexes are no longer part of the SQL standard, but are still supported by most relational DBMSs.

Create a schema

```
CREATE SCHEMA { schema_name
     | AUTHORIZATION authorization_ID
     | schema_name AUTHORIZATION authorization_ID }
```

Create a table

```
CREATE [ [ GLOBAL | LOCAL ] TEMPORARY ] table_name
     ( { column_name { data_type | domain_name } [ column_size ]
     [ column_constraint … ] , …
     [ DEFAULT default_value ]
     [ table_constraint ], …
     [ ON COMMIT DELETE | PRESERVE ROWS ] )
```

Create a UDT

```
CREATE TYPE type_name AS [ OBJECT ](column_definitions)
      [ INSTANTIABLE | { NOT INSTANTIABLE } ]
      [ FINAL | { NOT FINAL } ]
      [ { METHOD method_name (parameter_list) }, … ]
```

Create a typed table

```
CREATE TABLE table_name OF UDT_name
     [ UNDER supertype_name (added_column_list) ]
     [ REF IS reference_column_name
```

```
    [ REF IS reference_column_name
            ( { REF USING existing_data_type }
            | { REF IS identifier_name SYSTEM GENERATED }
            | { REF FROM attribute_list } ) ]
```

Create a database user account and password

```
CREATE USER | LOGIN implementation_specific_syntax
```

Note: Creating user accounts is not part of the SQL standard, and much of the syntax is implementation dependent.

Create a view

```
CREATE VIEW view_name [ (column_list ) ]
      AS (complete_SELECT_statement
      [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Remove a dynamic SQL descriptor area from main memory

```
DEALLOCATE DESCRIPTOR descriptor_name
```

Declare a cursor for processing an embedded SQL SELECT that returns multiple rows

```
DECLARE CURSOR cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR FOR
      (complete_SELECT_statement)
      [ FOR ( { READ ONLY } | UPDATE [ OF column_name, … ] ) ]
  | prepared_dynamic_SQL_statement_name
```

Delete rows from a table

```
DELETE FROM table_name
      [ { WHERE row_selection_predicate }
      | { WHERE CURRENT OF cursor_name } ]
```

Describe the dynamic parameters in a prepared dynamic SQL statement for a descriptor area

```
DESCRIBE [ INPUT | OUTPUT ]
      Prepared_dyamic_SQL_statement_name
      USING SQL DESCRIPTOR descriptor_name
```

Disconnect from a database

```
DISCONNECT connection_identifier
```

Remove an assertion from a schema

```
DROP ASSERTION assertion_name
```

Remove a domain from a schema

```
DROP DOMAIN domain_name CASCADE | RESTRICT
```

Remove an index from a schema

```
DROP INDEX index_name
```

Remove a schema from a catalog

```
DROP SCHEMA schema_name CASCADE | RESTRICT
```

Remove a table from a schema

```
DROP TABLE table_name CASCADE | RESTRICT
```

Remove a view from a schema

```
DROP VIEW view_name CASCADE | RESTRICT
```

Execute an embedded SQL statement

```
EXEC SQL complete_SQL_statement
```

Execute a prepared dynamic SQL statement

```
EXECTUE [ GLOBAL | LOCAL ] prepared_dynamic_SQL_statement
    [ INTO { parameter
```

```
     [ INTO { parameter, … }
     | { SQL DESCRIPTOR [ GLOBAL | LOCAL ] descriptor_name } ]
     [ USING { parameter, … }
     | { SQL DESCRIPTOR [ GLOBAL | LOCAL ] descriptor_name } ]
```

Execute a dynamic SQL statement immediately, without a separate preparation step

```
EXECUTE IMMEDIATE SQL_statement_text_literal_or_variable
```

Retrieve a row from an open cursor's result table

```
FETCH [ [ NEXT | PRIOR | FIRST | LAST | ABSOLUTE | { RELATIVE
row_number } ]
     FROM cursor_name
     INTO host_language_variable, …
```

Retrieve information from a dynamic SQL descriptor area

```
GET DESCRIPTOR descriptor_name
     { host_langague_variable = COUNT | KEY_TYPE | DYNAMIC_FUNCTION |
DYNAMIC_FUNCTION_CODE | TOP_LEVEL_COUNT }
     |   VALUE descriptor_number { host_language_variable = descriptor_
field }, …
```

Note: Descriptor field types most commonly used are TYPE (data type of parameter), DATA (actual value of parameter), and INDICATOR (value of indicator variable associated with parameter).

Grant access rights to other users

```
GRANT { ALL PRIVILEGES }
             | SELECT
             | DELETE
             | INSERT [ (column_name, …) ]
             | UPDATE [ (column_name, …) ]
             | REFERENCES { (column_name, …) }
             | USAGE
     ON { [ TABLE ] table_name }
             | { DOMAIN domain_name }
     TO { user_id, … } | PUBLIC
     [ WITH GRANT OPTION ]
```

Insert new rows into a table

```
INSERT INTO table_name
     [ {column_name, …) ]
     { VALUES (value1, value2, …) }
     | complete_SELECT_statement
     | DEFAULT VALUES
```

Conditionally update, delete, or insert data from one table into another

```
MERGE INTO target_table_name USING source_table_name ON merge_condition
WHEN MATCHED THAN
     Update/delete specifications
WHEN NOT MATCHED THEN
     insert specification
```

Open a cursor, executing the SELECT and positioning the cursor at the first row

```
OPEN cursor_name
     [ { USING host_language_variable_or_literal, … }
```

```
    | { SQL DESCRIPTOR descriptor_name } ]
```
Prepare a dynamic SQL statement for execution
```
PREPARE [ GLOBAL | LOCAL ]
    prepared_dynamic_SQL_statement_name
    FROM SQL_statement_text_literal_or_variable
```
Remove access rights from a user
```
REMOVE [GRANT OPTION FOR ]
            { ALL PRIVILEGES }
            | SELECT
            | DELETE
            | UPDATE
            | REFERENCES
            | USAGE
      ON [ TABLE ] table_name
            | DOMAIN domain_name
      FROM PUBLIC | { user_id, … }
      CASCADE | RESTRICT
```
Roll back a transaction
```
ROLLBACK [ WORK ]
```
Retrieve rows from a table
```
SELECT [DISTINCT]
        { { summary_function, … }
        | { data_manipulation_expression, … }
        | { column_name, … } }
    FROM { { table_name [ AS ] [ correlation_name ] }
        | joined_tables
        | complete_SELECT_statement }
    [ WHERE row_selection_predicate ]
    [ GROUP BY column_name, … ]
        [ HAVING group_selection_predicate ]
    [ UNION | INTERSECT | EXCEPT [CORRESPONDING BY (column_name, …) ]
        complete_SELECT_statement ]
    [ ORDER BY (column_name [ ASC | DESC ], …) ]
```
Retrieve rows from a common table expression (CTE)
```
WITH [ RECURSIVE ] CTE_name (column_list) AS
     (SELECT_statement_defining_table
complete_SELECT_using_result_of_CTE_query
```
Choose the current catalog
```
SET CATALOG catalog_name
```
Choose an active connection

Choose an active connection
```
SET CONNECTION connection_name | DEFAULT
```
Choose when constraints are checked
```
SET CONSTRAINTS MODE { constraint_name, … | ALL }
      DEFFERED | IMMEDIATE
```
Store values in a SQL descriptor area
```
SET DESCRIPTOR [ GLOBAL | LOCAL ]
          descriptor_name { COUNT = integer_value }
     | {VALUE descriptor_number { descriptor_field = value, …}, …}
```
Choose the current schema
```
SET SCHEMA schema_name
```
Choose the characteristics of the next transaction
```
SET TRANSACTION
     { ISOLATION LEVEL
          { READ UNCOMMITED }
          | { READ COMMITTED }
          | { REPEATABLE READ }
          | { SERIALIZABLE } }
     | { READ ONLY } | { READ WRITE }
```
Begin a transaction
```
START TRANSACTION transaction_mode
```
Remove all rows from a table leaving the table structure intact
```
TRUNCATE TABLE table_name
```
Change the data in a table
```
UPDATE table_name
      SET { column_name = { value
                          | NULL
                          | DEFAULT }, … }
          [ { WHERE row_selection_predicate }
          | { WHERE CURRENT OF cursor_name } ]
```

The second table (Table C.2) describes SQL built-in functions discussed in this book, including input data types. In Table C.3 you will find SQL operators covered in the text.

Table C.2
SQL Functions

Function	Returns	Input Data
AVG ( )	Average of values	Numeric values
COUNT (*)	Number of rows in a result set	none
LOWER ( )	Convert to lowercase	Character value
MAX ( )	Maximum value	Number, character, or datetime values

MAX ( )	Maximum value	Number, character, or datetime values
MIN ( )	Minimum value	Number, character, or datetime values
SUBSTRING ( )	Portion of a character string	Character value
SUM ( )	Sum of values	Numeric values
TRIM ( )	Remove trailing blanks	Character value
UPPER ( )	Convert to uppercase	Character value
XMLATTRIBUTES	Create XML element attributes	Attribute value, attribute name
XML COMMENT ( )	Append comment to XML document string	Character value
XMLCONCAT ( )	Concatenate XML fragments	Character values containing XML text
XMLELEMENT ( )	Create an XML element	Element name, optional attributes, content of element
XMLFOREST ( )	Create nested XML element	Element content, element name
XMLPARSE ( )	Convert text to XML	Element type, content of element
XMLROOT ( )	Modify XML Prolog	XML character string, XML version, standalone property
XMLSERIALIZE ( )	Covert an XML string to text	Character string formatted as XML

Table C.3

SQL Operators

Operator	Use	Operates on:
Arithmetic	Compute arithmetic quantities	
+	Preserve the sign of a value	Numeric value
-	Change the sign of a value	Numeric value
*	Multiply two values	Numeric values
/	Divide one value by another	Numeric values
+	Add two values	Numeric values
-	Subtract one value from another	Numeric values
Comparison	Compare two values	
=	Equality	Any compatible data types
>	Greater than	Any compatible data types
> =	Greater than or equal to	Any compatible data types
<	Less than	Any compatible data types
< =	Less than or equal to	Any compatible data types
!= *or* <>	Note equal to	Any compatible data types
Logical		
AND	Determine if two expressions are true	Expressions returning a Boolean value
OR	Determine if at least one of two expressions is true	Expressions returning a Boolean value
NOT	Change the truth value	Expression returning a Boolean value
= *or* :=	Assignment	Any compatible data types
||	Concatenate two strings	Character strings
Specialty operators		
BETWEEN	Determine if a value falls inside an interval	Numeric, characters, or datetime values
DISTINCT	Remove duplicate rows	Table
EXCEPT	Find the difference between two tables	Tables
EXISTS	Determine if a subquery result table contains at least one row	Table
EXTRACT	Pull out portion of a datetime	Datetime
IN	Determine if a value is in a set	Any set of values of the same datatype
INTERSECT	Find rows in common of two tables	Tables
IS NULL	Determine if a value is null	Any data type
IS NOT NULL	Determine if a value is not null	Any data type
JOIN	Combine two tables horizontally	Tables
LIKE	Perform string pattern matching	Character value
MULTISET EXCEPT	Find elements unique to each of two multisets	Multisets
MULTISET INTERSECT	Find elements common to two multisets	Multisets

MULTISET UNION	Combine two multisets vertically	Multisets
NOT IN	Determine if a value is not in a set of values	Any sets of values of the same data type
OVERLAPS	Determine if two datetime intervals overlap	Datetimes
UNION	Combine to tables vertically	Tables