4
Creating and Managing Schema Objects

To create, change, and drop schema objects, you use data definition language (DDL) statements.

About Data Definition Language (DDL) Statements

Data definition language (DDL) statements create, change, and drop schema objects. Before and after a DDL statement, Oracle Database issues an implicit COMMIT statement; therefore, you cannot roll back a DDL statement.

> **Note:**
>
> When creating schema objects, you must observe the schema object naming rules in *Oracle Database SQL Language Reference*.

In the SQL*Plus environment, you can enter a DDL statement after the `SQL>` prompt.

In the SQL Developer environment, you can enter a DDL statement in the Worksheet. Alternatively, you can use SQL Developer tools to create, change, and drop objects.

Some DDL statements that create schema objects have an optional OR REPLACE clause, which allows a statement to replace an existing schema object with another that has the same name and type. When SQL Developer generates code for one of these statements, it always includes the OR REPLACE clause.

To see the effect of a DDL statement in SQL Developer, you might have to select the schema object type of the newly created object in the Connections frame and then click the **Refresh** icon.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for more information about DDL statements
> - "Committing Transactions"

Creating and Managing Tables

Tables are the basic units of data storage in Oracle Database. Tables hold all user-accessible data. Each table contains rows that represent individual data records. Rows are composed of columns that represent the fields of the records.

> **Note:**
>
> To do the tutorials in this document, the `hr` sample schema must be installed and you must be connected to Oracle Database as the user HR from SQL Developer.

> **See Also:**
>
> - "Tutorial: Viewing EMPLOYEES Table Properties and Data with SQL Developer"
> - *Oracle SQL Developer User's Guide* for a SQL Developer tutorial that includes creating and populating tables
> - *Oracle Database Concepts* for general information about tables

About SQL Data Types

When you create a table, you must specify the SQL data type for each column, which determines what values the column can contain.

For example, a column of type DATE can contain the value `'01-MAY-05'`, but it cannot contain the numeric value 2 or the character value 'shoe'. SQL data types fall into two categories: built-in and user-defined. (PL/SQL has additional data types—see "About PL/SQL Data Types".)

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for a summary of built-in SQL data types
> - *Oracle Database Concepts* for introductions to each of the built-in SQL data types
> - *Oracle Database SQL Language Reference* for more information about user-defined data types
> - "About PL/SQL Data Types"

Creating Tables

To create tables, use either the SQL Developer tool Create Table or the DDL statement CREATE TABLE.

This section shows how to use both of these ways to create these tables, which will contain data about employee evaluations:

- PERFORMANCE_PARTS, which contains the categories of employee performance that are evaluated and their relative weights

- EVALUATIONS, which contains employee information, evaluation date, job, manager, and department
- SCORES, which contains the scores assigned to each performance category for each evaluation

These tables appear in many tutorials and examples in this document.

Tutorial: Creating a Table with the Create Table Tool

This tutorial shows how to create the `PERFORMANCE_PARTS` table using the SQL Developer tool Create Table.

To create the PERFORMANCE_PARTS table using the Create Table tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, right-click **Tables**.
3. In the list of choices, click **New Table**.

   The Create Table window opens, with default values for a new table, which has only one row.
4. For Schema, accept the default value, HR.
5. For Name, enter `PERFORMANCE_PARTS`.
6. In the default row:
   - For PK (primary key), accept the default option, deselected.
   - For Column Name, enter `PERFORMANCE_ID`.
   - For Type, accept the default value, VARCHAR2.
   - For Size, enter `2`.
   - For Not Null, accept the default option, deselected.
7. Click **Add Column**.
8. For Column Name, enter `NAME`.
9. For Type, accept the default value, VARCHAR2.
10. For Size, enter `80`.
11. Click **Add Column**.
12. For Column Name, enter `WEIGHT`.
13. For Type, select `NUMBER` from the menu.
14. Click **OK**.

    The table `PERFORMANCE_PARTS` is created. Its name appears under Tables in the Connections frame.

    To see the CREATE TABLE statement for creating this table, select `PERFORMANCE_PARTS` and click the tab **SQL**.

> **See Also:**
>
> *Oracle SQL Developer User's Guide* for more information about using SQL Developer to create tables

Creating Tables with the CREATE TABLE Statement

This section shows how to use the CREATE TABLE statement to create the EVALUATIONS and SCORES tables.

The CREATE TABLE statement in Example 4-1 creates the EVALUATIONS table.

The CREATE TABLE statement in Example 4-2 creates the SCORES table.

In SQL Developer, in the Connections frame, if you expand Tables, you can see the tables EVALUATIONS and SCORES.

Example 4-1 Creating the EVALUATIONS Table with CREATE TABLE

```
CREATE TABLE EVALUATIONS (
  EVALUATION_ID    NUMBER(8,0),
  EMPLOYEE_ID      NUMBER(6,0),
  EVALUATION_DATE  DATE,
  JOB_ID           VARCHAR2(10),
  MANAGER_ID       NUMBER(6,0),
  DEPARTMENT_ID    NUMBER(4,0),
  TOTAL_SCORE      NUMBER(3,0)
);
```

Result:

```
Table created.
```

Example 4-2 Creating the SCORES Table with CREATE TABLE

```
CREATE TABLE SCORES (
  EVALUATION_ID   NUMBER(8,0),
  PERFORMANCE_ID  VARCHAR2(2),
  SCORE           NUMBER(1,0)
);
```

Result:

```
Table created.
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the CREATE TABLE statement

Ensuring Data Integrity in Tables

To ensure that the data in your tables satisfies the business rules that your application models, you can use constraints, application logic, or both.

> **Tip:**
>
> Wherever possible, use constraints instead of application logic. Oracle Database checks that all data obeys constraints much faster than application logic can.

> **See Also:**
>
> - *Oracle Database Concepts* for additional general information about data integrity
> - *Oracle Database SQL Language Reference* for syntactic information about constraints
> - *Oracle Database Development Guide* for information about enabling and disabling constraints

About Constraints

Constraints restrict the values that columns can have. Trying to change the data in a way that violates a constraint causes an error and rolls back the change. Trying to add a constraint to a populated table causes an error if existing data violates the constraint.

Constraints can be enabled and disabled. By default, they are created in the enabled state.

The constraint types are:

- **Not Null**, which prevents a value from being null

  In the EMPLOYEES table, the column LAST_NAME has the NOT NULL constraint, which enforces the business rule that every employee must have a last name.

- **Unique**, which prevents multiple rows from having the same value in the same column or combination of columns, but allows some values to be null

  In the EMPLOYEES table, the column EMAIL has the UNIQUE constraint, which enforces the business rule that an employee can have no email address, but cannot have the same email address as another employee.

- Primary Key, which is a combination of NOT NULL and UNIQUE

  In the EMPLOYEES table, the column EMPLOYEE_ID has the PRIMARY KEY constraint, which enforces the business rule that every employee must have a unique employee identification number.

- **Foreign Key**, which requires values in one table to match values in another table

  In the EMPLOYEES table, the column JOB_ID has a FOREIGN KEY constraint that references the JOBS table, which enforces the business rule that an employee cannot have a JOB_ID that is not in the JOBS table.

- **Check**, which requires that a value satisfy a specified condition

  The EMPLOYEES table does not have CHECK constraints. However, suppose that EMPLOYEES needs a new column, EMPLOYEE_AGE, and that every employee must be at least 18. The constraint CHECK `(EMPLOYEE_AGE >= 18)` enforces the business rule.

> **Tip:**
>
> Use check constraints only when other constraint types cannot provide the necessary checking.

- REF, which further describes the relationship between a REF column and the object that it references

  A REF column references an object in another object type or in a relational table.

  For information about REF constraints, see *Oracle Database Concepts*.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for syntactic information about constraints

Tutorial: Adding Constraints to Existing Tables

This tutorial shows how to add constraints to existing tables using both SQL Developer tools and the ALTER TABLE statement.

To add constraints to existing tables, use either SQL Developer tools or the DDL statement ALTER TABLE. This topic shows how to use both of these ways to add constraints to the tables created in "Creating Tables".

This tutorial has several procedures. The first procedure uses the Edit Table tool to add a Not Null constraint to the `NAMES` column of the `PERFORMANCE_PARTS` table. The remaining procedures show how to use other tools to add constraints; however, you could add the same constraints using the Edit Table tool.

> **Note:**
>
> After any step of the tutorial, you can view the constraints that a table has:
>
> 1. In the Connections frame, select the name of the table.
> 2. In the right frame, click the tab **Constraints**.
>
> For more information about viewing table properties and data, see "Tutorial: Viewing EMPLOYEES Table Properties and Data with SQL Developer".

To add a Not Null constraint using the Edit Table tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, right-click **PERFORMANCE_PARTS**.
4. In the list of choices, click **Edit**.
5. In the Edit Table window, click the column **NAME**.

6. Select the property **Not Null**.
7. Click **OK**.

   The Not Null constraint is added to the `NAME` column of the `PERFORMANCE_PARTS` table.

The following procedure uses the ALTER TABLE statement to add a Not Null constraint to the `WEIGHT` column of the `PERFORMANCE_PARTS` table.

To add a Not Null constraint using the ALTER TABLE statement:

1. If a pane with the tab **hr_conn** is there, select it. Otherwise, click the icon **SQL Worksheet**, as in "Running Queries in SQL Developer".
2. In the Worksheet pane, type this statement:

```
ALTER TABLE PERFORMANCE_PARTS
MODIFY WEIGHT NOT NULL;
```

3. Click the icon **Run Statement**.

   The statement runs, adding the Not Null constraint to the `WEIGHT` column of the `PERFORMANCE_PARTS` table.

The following procedure uses the Add Unique tool to add a Unique constraint to the `SCORES` table.

To add a Unique constraint using the Add Unique tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, right-click **SCORES**.
4. In the list of choices, select **Constraint**.
5. In the list of choices, click **Add Unique**.
6. In the Add Unique window:
   a. For Constraint Name, enter `SCORES_EVAL_PERF_UNIQUE`.
   b. For Column 1, select **EVALUATION_ID** from the menu.
   c. For Column 2, select **PERFORMANCE_ID** from the menu.
   d. Click **Apply**.
7. In the Confirmation window, click **OK**.

   A unique constraint named `SCORES_EVAL_PERF_UNIQUE` is added to the `SCORES` table.

The following procedure uses the Add Primary Key tool to add a Primary Key constraint to the `PERFORMANCE_ID` column of the `PERFORMANCE_PARTS` table.

To add a Primary Key constraint using the Add Primary Key tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, right-click **PERFORMANCE_PARTS**.
4. In the list of choices, select **Constraint**.
5. In the list of choices, click **Add Primary Key**.

6. In the Add Primary Key window:
   a. For Primary Key Name, enter `PERF_PERF_ID_PK`.
   b. For Column 1, select **PERFORMANCE_ID** from the menu.
   c. Click **Apply**.
7. In the Confirmation window, click **OK**.

   A primary key constraint named `PERF_PERF_ID_PK` is added to the `PERFORMANCE_ID` column of the `PERFORMANCE_PARTS` table.

The following procedure uses the ALTER TABLE statement to add a Primary Key constraint to the `EVALUATION_ID` column of the `EVALUATIONS` table.

To add a Primary Key constraint using the ALTER TABLE statement:

1. If a pane with the tab **hr_conn** is there, select it. Otherwise, click the icon **SQL Worksheet**, as in "Running Queries in SQL Developer".
2. In the Worksheet pane, type this statement:

```
ALTER TABLE EVALUATIONS
ADD CONSTRAINT EVAL_EVAL_ID_PK PRIMARY KEY (EVALUATION_ID);
```

3. Click the icon **Run Statement**.

   The statement runs, adding the Primary Key constraint to the `EVALUATION_ID` column of the `EVALUATIONS` table.

The following procedure uses the Add Foreign Key tool to add two Foreign Key constraints to the `SCORES` table.

To add two Foreign Key constraints using the Add Foreign Key tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, right-click **SCORES**.
4. In the list of choices, select **Constraint**.
5. In the list of choices, click **Add Foreign Key**.
6. In the Add Foreign Key window:
   a. For Constraint Name, enter `SCORES_EVAL_FK`.
   b. For Column Name, select **EVALUATION_ID** from the menu.
   c. For References Table Name, select **EVALUATIONS** from the menu.
   d. For Referencing Column, select **EVALUATION_ID** from the menu.
   e. Click **Apply**.
7. In the Confirmation window, click **OK**.

   A foreign key constraint named `SCORES_EVAL_FK` is added to the `EVALUTION_ID` column of the `SCORES` table, referencing the `EVALUTION_ID` column of the `EVALUATIONS` table.

   The following steps add another foreign key constraint to the `SCORES` table.
8. In the list of tables, right-click **SCORES**.

9. In the list of tables, select **Constraint**.
10. In the list of choices, click **Add Foreign Key**.

    The Add Foreign Key window opens.
11. In the Add Foreign Key window:
    a. For Constraint Name, enter `SCORES_PERF_FK`.
    b. For Column Name, select **PERFORMANCE_ID** from the menu.
    c. For Reference Table Name, select **PERFORMANCE_PARTS** from the menu.
    d. For Referencing Column, select **PERFORMANCE_ID** from the menu.
    e. Click **Apply**.
12. In the Confirmation window, click **OK**.

    A foreign key constraint named `SCORES_PERF_FK` is added to the `EVALUTION_ID` column of the `SCORES` table, referencing the `EVALUTION_ID` column of the `EVALUATIONS` table.

The following procedure uses the ALTER TABLE statement to add a Foreign Key constraint to the `EMPLOYEE_ID` column of the `EVALUATIONS` table, referencing the `EMPLOYEE_ID` column of the `EMPLOYEES` table.

To add a Foreign Key constraint using the ALTER TABLE statement:

1. If a pane with the tab **hr_conn** is there, select it. Otherwise, click the icon **SQL Worksheet**, as in "Running Queries in SQL Developer".
2. In the Worksheet pane, type this statement:

```
ALTER TABLE EVALUATIONS
ADD CONSTRAINT EVAL_EMP_ID_FK FOREIGN KEY (EMPLOYEE_ID)
REFERENCES EMPLOYEES (EMPLOYEE_ID);
```

3. Click the icon **Run Statement**.

   The statement runs, adding the Foreign Key constraint to the `EMPLOYEE_ID` column of the `EVALUATIONS` table, referencing the `EMPLOYEE_ID` column of the `EMPLOYEES` table.

The following procedure uses the Add Check tool to add a Check constraint to the `SCORES` table.

To add a Check constraint using the Add Check tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, right-click **SCORES**.
4. In the list of choices, select **Constraint**.
5. In the list of choices, click **Add Check**.
6. In the Add Check window:
    a. For Constraint Name, enter `SCORE_VALID`.
    b. For Check Condition, enter `score >= 0 and score <+ 9`.
    c. For Status, accept the default, `ENABLE`.
    d. Click **Apply**.

7. In the Confirmation window, click **OK**.

   A Check constraint named `SCORE_VALID` is added to the `SCORES` table.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for more information about the ALTER TABLE statement
> - *Oracle SQL Developer User's Guide* for information about adding constraints to a table when you create it with SQL Developer
> - *Oracle Database SQL Language Reference* for information about adding constraints to a table when you create it with the CREATE TABLE statement

Tutorial: Adding Rows to Tables with the Insert Row Tool

This tutorial shows how to use the Insert Row tool to add six populated rows to the PERFORMANCE_PARTS table.

To add rows to the PERFORMANCE_PARTS table using the Insert Row tool:

1. In the Connections frame, expand hr_conn.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, select **PERFORMANCE_PARTS**.
4. In the right frame, click the tab **Data**.

   The Data pane appears, showing the names of the columns of the PERFORMANCE_PARTS table and no rows.
5. In the Data pane, click the icon **Insert Row**.

   A new row appears, with empty columns. A green border around the row number indicates that the insertion has not been committed.
6. Click the cell under the column heading PERFORMANCE_ID.
7. Type the value of PERFORMANCE_ID: `WM`
8. Either press the key **Tab** or click the cell under the column heading **NAME**.
9. Type the value of NAME: `Workload Management`
10. Either press the key **Tab** or click the cell under the column heading WEIGHT.
11. Type the value of WEIGHT: `0.2`
12. Press the key **Enter**.
13. Add and populate a second row by repeating steps 5 through 12 with these values:
    - For PERFORMANCE_ID, type `BR`.
    - For NAME, type `Building Relationships`.
    - For WEIGHT, type `0.2`.

14. Add and populate a third row by repeating steps 5 through 12 with these values:
    - For PERFORMANCE_ID, type `CF`.
    - For NAME, type `Customer Focus`.
    - For WEIGHT, type `0.2`.
15. Add and populate a fourth row by repeating steps 5 through 12 with these values:
    - For PERFORMANCE_ID, type `CM`.
    - For NAME, type `Communication`.
    - For WEIGHT, type `0.2`.
16. Add and populate a fifth row by repeating steps 5 through 12 with these values:
    - For PERFORMANCE_ID, type `TW`.
    - For NAME, type `Teamwork`.
    - For WEIGHT, type `0.2`.
17. Add and populate a sixth row by repeating steps 5 through 12, using these values:
    - For PERFORMANCE_ID, type `RO`.
    - For NAME, type `Results Orientation`.
    - For WEIGHT, type `0.2`.
18. Click the icon **Commit Changes**.

    The green borders around the row numbers disappear.

    Under the Data pane is the label Messages - Log.
19. Check the Messages - Log pane for the message `Commit Successful`.
20. In the Data Pane, check the new rows.

> **See Also:**
>
> "About the INSERT Statement"

Tutorial: Changing Data in Tables in the Data Pane

This tutorial shows how to change three of the WEIGHT values in the PERFORMANCE_PARTS table in the Data pane.

The PERFORMANCE_PARTS table was populated in "Tutorial: Adding Rows to Tables with the Insert Row Tool".

To change data in the PERFORMANCE_PARTS table using the Data pane:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, select **PERFORMANCE_PARTS**.
4. In the right frame, click the tab **Data**.

5. In the Data Pane, in the row where NAME is "Workload Management":
   a. Click the **WEIGHT** value.
   b. Enter the value `0.3`.
   c. Press the key **Enter**.

      An asterisk appears to the left of the row number to indicate that the change has not been committed.
6. In the row where **NAME** is "Building Relationships":
   a. Click the **WEIGHT** value.
   b. Enter the value `0.15`.
   c. Press the key **Enter**.

      An asterisk appears to the left of the row number to indicate that the change has not been committed.
7. In the row where **NAME** is "Customer Focus" :
   a. Click the **WEIGHT** value.
   b. Enter the value `0.15`.
   c. Press the key **Enter**.

      An asterisk appears to the left of the row number to indicate that the change has not been committed.
8. Click the icon **Commit Changes**.

   The asterisks to the left of the row numbers disappear.
9. Under the Data pane, check the Messages - Log pane for the message `Commit Successful`.
10. In the Data Pane, check the new data.

> **See Also:**
>
> "About the UPDATE Statement"

Tutorial: Deleting Rows from Tables with the Delete Selected Row(s) Tool

This tutorial shows how to use the Delete Selected Row(s) tool to delete a row from the PERFORMANCE_PARTS table.

The PERFORMANCE_PARTS table was populated in "Tutorial: Adding Rows to Tables with the Insert Row Tool").

To delete row from PERFORMANCE_PARTS using Delete Selected Row(s) tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.

3. In the list of tables, select **PERFORMANCE_PARTS**.
4. In the right frame, click the tab **Data**.
5. In the Data pane, click the row where NAME is "Results Orientation".
6. Click the icon **Delete Selected Row(s)**.

   A red border appears around the row number to indicate that the deletion has not been committed.
7. Click the icon **Commit Changes**.

   The row is deleted.
8. Under the Data pane, check the Messages - Log pane for the message `Commit Successful`.

> **Note:**
>
> If you delete every row of a table, the empty table still exists. To delete a table, see "Dropping Tables".

> **See Also:**
>
> "About the DELETE Statement"

Managing Indexes

You can create indexes on one or more columns of a table to speed SQL statement execution on that table. When properly used, indexes are the primary means of reducing disk input/output (I/O).

When you define a primary key on a table:

- If an existing index starts with the primary key columns, then Oracle Database uses that existing index for the primary key. The existing index need not be Unique.

  For example, if you define the primary key (A, B), Oracle Database uses the existing index (A, B, C).
- If no existing index starts with the primary key columns and the constraint is immediate, then Oracle Database creates a Unique index on the primary key.
- If no existing index starts with the primary key columns and the constraint is deferrable, then Oracle Database creates a non-Unique index on the primary key.

For example, in "Tutorial: Adding Constraints to Existing Tables", you added a Primary Key constraint to the EVALUATION_ID column of the EVALUATIONS table. Therefore, if you select the EVALUATIONS table in the SQL Developer Connections frame and click the Indexes tab, the Indexes pane shows a Unique index on the EVALUATION_ID column.

> **See Also:**
>
> For more information about indexes:
>
> - *Oracle Database Concepts*
> - *Oracle Database Development Guide*

Tutorial: Adding an Index with the Create Index Tool

This tutorial shows how to use the Create Index tool to add an index to the EVALUATIONS table.

The EVALUATIONS table was created in Example 4-1.

To create an index, use either the SQL Developer tool Create Index or the DDL statement CREATE INDEX. The equivalent DDL statement is:

```
CREATE INDEX EVAL_JOB_IX
ON EVALUATIONS (JOB_ID ASC) NOPARALLEL;
```

To add an index to the EVALUATIONS table using the Create Index tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, right-click **EVALUATIONS**.
4. In the list of choices, select **Index**.
5. In the list of choices, select **Create Index**.
6. In the Create Index window:
   a. For Schema, accept the default, HR.
   b. For Name, type `EVAL_JOB_IX`.
   c. If the Definition pane does not show, select the tab **Definition**.
   d. In the Definition pane, for Index Type, select **Unique** from the menu.
   e. Click the icon **Add Expression**.

      The Expression EMPLOYEE_ID with Order <Not Specified> appears.
   f. Over EMPLOYEE_ID, type `JOB_ID`.
   g. For Order, select **ASC** (ascending) from the menu.
   h. Click **OK**.

      Now the EVALUATIONS table has an index named EVAL_JOB_IX on the column JOB_ID.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the CREATE INDEXstatement

Tutorial: Changing an Index with the Edit Index Tool

This tutorial shows how to use the Edit Index tool to reverse the sort order of the index EVAL_JOB_IX.

To change an index, use either the SQL Developer tool Edit Index or the DDL statements DROP INDEX and CREATE INDEX.

The equivalent DDL statements are:

```
DROP INDEX EVAL_JOB_ID;

CREATE INDEX EVAL_JOB_IX
ON EVALUATIONS (JOB_ID DESC) NOPARALLEL;
```

To reverse the sort order of the index EVAL_JOB_IX using the Edit Index tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Indexes**.
3. In the list of indexes, right-click **EVAL_JOB_IX**.
4. In the list of choices, click **Edit**.
5. In the Edit Index window, change Order to **DESC**.
6. Click **OK**.
7. In the Confirm Replace window, click either **Yes** or **No**.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the ALTER INDEX statement

Tutorial: Dropping an Index

This tutorial shows how to use the Connections frame and Drop tool to drop the index EVAL_JOB_IX.

To drop an index, use either the SQL Developer Connections frame and Drop tool or the DDL statement DROP INDEX. The equivalent DDL statement is:

```
DROP INDEX EVAL_JOB_ID;
```

To drop the index EVAL_JOB_IX:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Indexes**.
3. In the list of indexes, right-click **EVAL_JOB_IX**.
4. In the list of choices, click **Drop**.
5. In the Drop window, click **Apply**.
6. In the Confirmation window, click **OK**.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the DROP INDEX statement

Dropping Tables

To drop a table, use either the SQL Developer Connections frame and Drop tool, or the DDL statement DROP TABLE.

> **Caution:**
>
> Do not drop any tables that you created in "Creating Tables"—you need them for later tutorials. If you want to practice dropping tables, create simple ones and then drop them.

To drop a table using the Drop tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Tables**.
3. In the list of tables, right-click the name of the table to drop.
4. In the list of choices, select **Table**.
5. In the list of choices, click **Drop**.
6. In the Drop window, click **Apply**.
7. In the Confirmation window, click **OK**.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the statement DROP TABLE

Creating and Managing Views

A view presents a query result as a table. In most places that you can use a table, you can use a view. Views are useful when you need frequent access to information that is stored in several different tables.

> **See Also:**
>
> - "Selecting Table Data" for information about queries
> - *Oracle Database Concepts* for additional general information about views

Creating Views

To create views, use either the SQL Developer tool Create View or the DDL statement CREATE VIEW.

This topic shows how to use both of these ways to create these views:

- SALESFORCE, which contains the names and salaries of the employees in the Sales department
- EMP_LOCATIONS, which contains the names and locations of all employees

  This view is used in "Creating an INSTEAD OF Trigger".

> **See Also:**
>
> - *Oracle SQL Developer User's Guide* for more information about using SQL Developer to create a view
> - *Oracle Database SQL Language Reference* for more information about the statement CREATE VIEW

Tutorial: Creating a View with the Create View Tool

This tutorial shows how to create the SALESFORCE view using the Create View tool.

To create the SALESFORCE view using the Create View tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, right-click **Views**.
3. In the list of choices, click **New View**.

   The Create View window opens, with default values for a new view.
4. For Schema, accept the default value, HR.
5. For Name, enter `SALESFORCE`.
6. If the SQL Query pane does not show, click the tab **SQL Query**.
7. In the SQL Query pane, in the SQL Query field:
   - After `SELECT`, type:

     ```
     FIRST_NAME || ' ' || LAST_NAME "Name", SALARY*12 "Annual Salary"
     ```
   - After `FROM`, type:

     ```
     EMPLOYEES WHERE DEPARTMENT_ID = 80
     ```

8. Click **Check Syntax**.
9. Under Syntax Results, if the message is not `No errors found in SQL`, then return to step 7 and correct the syntax errors in the query.
10. Click **OK**.

    The view SALESFORCE is created. To see it, expand Views in the Connections frame.

    To see the CREATE VIEW statement for creating this view, select its name and click the tab **SQL**.

> **See Also:**
>
> *Oracle SQL Developer User's Guide* for more information about using SQL Developer to create views

Creating Views with the CREATE VIEW Statement

This example shows how to use the CREATE VIEW statement to create the EMP_LOCATIONS view, which joins four tables.

The CREATE VIEW statement in Example 4-3 creates the EMP_LOCATIONS view, which joins four tables. (For information about joins, see "Selecting Data from Multiple Tables".)

Example 4-3 Creating the EMP_LOCATIONS View with CREATE VIEW

```
CREATE VIEW EMP_LOCATIONS AS
SELECT e.EMPLOYEE_ID,
  e.LAST_NAME || ', ' || e.FIRST_NAME NAME,
  d.DEPARTMENT_NAME DEPARTMENT,
  l.CITY CITY,
  c.COUNTRY_NAME COUNTRY
FROM EMPLOYEES e, DEPARTMENTS d, LOCATIONS l, COUNTRIES c
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID AND
 d.LOCATION_ID = l.LOCATION_ID AND
 l.COUNTRY_ID = c.COUNTRY_ID
ORDER BY LAST_NAME;
```

Result:

```
View EMP_LOCATIONS created.
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the CREATE VIEW statement

Changing Queries in Views

To change the query in a view, use the DDL statement CREATE VIEW with the OR REPLACE clause.

The CREATE OR REPLACE VIEW statement in Example 4-4 changes the query in the SALESFORCE view.

Example 4-4 Changing the Query in the SALESFORCE View

```
CREATE OR REPLACE VIEW SALESFORCE AS
  SELECT FIRST_NAME || ' ' || LAST_NAME "Name",
  SALARY*12 "Annual Salary"
  FROM EMPLOYEES
  WHERE DEPARTMENT_ID = 80 OR DEPARTMENT_ID = 20;
```

Result:

```
View SALESFORCE created.
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the CREATE VIEW with the OR REPLACE clause

Tutorial: Changing View Names with the Rename Tool

This tutorial shows how to use the Rename tool to change the name of the SALESFORCE view.

To change the name of a view, use either the SQL Developer tool Rename or the RENAME statement. The equivalent DDL statement is:

```
RENAME SALESFORCE to SALES_MARKETING;
```

To change the SALESFORCE view using the Rename tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Views**.
3. In the list of views, right-click **SALESFORCE**.
4. In the list of choices, select **Rename**.
5. In the Rename window, in the New View Name field, type `SALES_MARKETING`.
6. Click **Apply**.
7. In the Confirmation window, click **OK**.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the RENAME statement

Dropping a View

To drop a view, use either the SQL Developer Connections frame and Drop tool or the DDL statement DROP VIEW.

The following tutorial shows how to use the Connections frame and Drop tool to drop the view SALES_MARKETING (changed in "Tutorial: Changing View Names with the Rename Tool"). The equivalent DDL statement is:

```
DROP VIEW SALES_MARKETING;
```

To drop the view SALES_MARKETING using the Drop tool:

1. In the Connections frame, expand **hr_conn**.
2. In the a list of schema object types, expand **Views**.
3. In the a list of views, right-click **SALES_MARKETING**.
4. In the a list of choices, click **Drop**.
5. In the Drop window, click **Apply**.
6. In the Confirmation window, click **OK**.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the DROP VIEW statement

Creating and Managing Data Use Case Domains

Data use case domains are lightweight data type modifiers that encapsulate a set of optional properties and constraints, allowing for the modeling of real-world information such as credit card numbers, email addresses, dates of birth, postal codes, and so on. You can use data use case domains to define how you intend to use the data centrally and share the data with other applications.

> **See Also:**
>
> *Oracle Database Concepts* for additional general information about use case domains

Creating Use Case Domains

To create a use case domain, use the DDL statement CREATE DOMAIN.

The following statement creates the `email` domain.

```
CREATE DOMAIN email AS VARCHAR2(30)
CONSTRAINT EMAIL_C CHECK (REGEXP_LIKE (email, '^(\S+)\@(\S+)\.(\S+)$'))
DISPLAY '---' || SUBSTR(email, INSTR(email, '@') + 1);
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the CREATE DOMAIN statement

Dropping Use Case Domains

To drop a use case domain, use the DDL statement DROP DOMAIN.

1. The following statement drops the `email` domain.

   ```
   DROP DOMAIN email;
   ```

   The DROP DOMAIN command does not allow you to drop a domain if that domain is associated with any column on any table.

   You can use the DROP DOMAIN ... FORCE command to disassociate the domain from all columns and drop the domain. Use the FORCE option with caution, because you will lose all domain-specified knowledge on all columns that have been associated with the domain.

2. The following statement drops the `email` domain and disassociates the domain from all of its dependent columns.

   ```
   DROP DOMAIN email FORCE
   ```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the DROP DOMAIN statement

Creating and Managing Schema Annotations

Schema annotations are free-form text fields that contain extended or custom properties of database objects such as tables, views, columns, indexes, and data use case domains. Annotations are a lightweight declarative facility to centrally register usage properties for database schema objects. They can be thought of as lightweight standardized markup for

database metadata, for use by applications to register and process extended and custom usage properties.

> **See Also:**
>
> *Oracle Database Concepts* for additional general information about annotations

Creating Annotations

Annotations are properties of schema objects, and can be specified in the CREATE statement for the object.

1. The following statement creates a new table with annotations for the table itself and the underlying columns.

```
CREATE TABLE employees
(
    id NUMBER(5) PRIMARY KEY ANNOTATIONS (Identity, Display
'Employee Name'),
    ename VARCHAR2(50) ANNOTATIONS(Display 'Employee Name'),
    salary NUMBER ANNOTATION(Display 'Employee Name', Confidential)
) ANNOTATIONS (Display 'Employee Table');
```

2. The following statement includes an annotation in the creation of the `dept_codes` application usage domain in the `hr` schema.

```
CREATE DOMAIN dept_codes AS NUMBER(3)
  CONSTRAINT dept_chk CHECK (dept_codes > 99 AND dept_codes != 200)
  ANNOTATIONS (Title 'Domain Annotation);
```

> **See Also:**
>
> *Oracle Database Development Guide* for information about DDL statements for annotations

Listing Annotations

You can use dictionary views to get the list of annotations that are used for specific objects.

The following dictionary views are defined for annotations and annotation usage.

- ALL_ANNOTATION_VALUES, DBA_ANNOTATION_VALUES, USER_ANNOTATION_VALUES
- ALL_ANNOTATIONS, DBA_ANNOTATIONS, USER_ANNOTATIONS
- ALL_ANNOTATIONS_USAGE, DBA_ANNOTATIONS_USAGE, USER_ANNOTATIONS_USAGE

The ALL_* views include all annotations for objects owned by the user, all annotations for objects owned by other users where the user has the ALTER privilege, and all annotations for objects where the user has system privileges for that object type. The DBA_* views include all annotations for all objects in the database. The USER_* views include all annotations for objects owned by the user.

- The following statement gets the table-level annotations for the `EMPLOYEE` table:

```
SELECT * from USER_ANNOTATIONS_USAGE WHERE Object_Name = 'EMPLOYEE' AND
Object_Type = 'TABLE' AND Column_Name IS NULL;
```

- The following statement gets the column-level annotations for the `EMPLOYEE` table:

```
SELECT * from USER_ANNOTATIONS_USAGE WHERE Object_Name = 'EMPLOYEE' AND
Object_Type = 'TABLE' AND Column_Name IS NOT NULL;
```

- The following statement gets the column-level annotations for the `EMPLOYEE` table as a single JSON collection per column:

```
SELECT  U.Column_Name, JSON_ARRAYAGG(JSON_OBJECT(U.Annotation_Name,
U.Annotation_Value)) FROM USER_ANNOTATIONS_USAGE U
WHERE Object_Name = 'EMPLOYEE' AND Object_Type = 'TABLE' AND
Column_Name IS NOT NULL GROUP BY Column_Name;
```

> **See Also:**
>
> *Oracle Database Reference* for information about the dictionary views for annotations.

Modifying Annotations

To modify an annotation, use the ALTER statement for the annotated schema object.

1. The following statement adds an additional annotation `Department` with the value `HR` to the `employees` table.

```
ALTER TABLE employees ANNOTATIONS (ADD Department 'HR');
```

2. The following statement drops the annotation `Title` and adds a new annotation `Name` with the value `Domain` to the `dept_code` domain.

```
ALTER DOMAIN dept_codes ANNOTATIONS(DROP Title, ADD Name 'Domain');
```

> **See Also:**
>
> *Oracle Database Development Guide* for information about DDL statements for annotations

Creating and Managing Sequences

Sequences are schema objects from which you can generate unique sequential values, which are very useful when you need unique primary keys. Sequences are used through the pseudocolumns CURRVAL and NEXTVAL, which return the current and next values of the sequence, respectively.

After creating a sequence, you must initialize it by using NEXTVAL to get its first value. Only after you initialize a sequence does CURRVAL return its current value.

The HR schema has three sequences: DEPARTMENTS_SEQUENCE, EMPLOYEES_SEQUENCE, and LOCATIONS_SEQUENCE.

Tip:

When you plan to use a sequence to populate the primary key of a table, give the sequence a name that reflects this purpose. (This topic uses the naming convention *TABLE_NAME*_SEQUENCE.)

See Also:

- *Oracle Database Concepts* for an overview of sequences
- *Oracle Database SQL Language Reference* for more information about the CURRVAL and NEXTVAL pseudocolumns
- *Oracle Database Administrator's Guide* for information about managing sequences
- "Editing Installation Scripts that Create Sequences"
- "About Sequences and Concurrency"

Tutorial: Creating a Sequence

This tutorial shows how to use the Create Database Sequence tool to create a sequence to use to generate primary keys for the EVALUATIONS table.

The EVALUATIONS table was created in Example 4-1.

To create a sequence, use either the SQL Developer tool Create Sequence or the DDL statement CREATE SEQUENCE. The equivalent DDL statement is:

```
CREATE SEQUENCE evaluations_sequence
INCREMENT BY 1
START WITH 1 ORDER;
```

To create EVALUATIONS_SEQUENCE using the Create Database Sequence tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, right-click **Sequences**.

3. In the list of choices, click **New Sequence**.
4. In the Create Sequence window, in the Name field, type `EVALUATIONS_SEQUENCE` over the default value "SEQUENCE1".
5. If the Properties pane does not show, click the tab **Properties**.
6. In the Properties pane:
   a. In the field Increment, type `1`.
   b. In the field Start with, type `1`.
   c. For the remaining fields, accept the default values.
   d. Click **OK**.

      The sequence EVALUATIONS_SEQUENCE is created. Its name appears under Sequences in the Connections frame.

> **See Also:**
>
> - *Oracle SQL Developer User's Guide* for more information about using SQL Developer to create a sequence
> - *Oracle Database SQL Language Reference* for information about the CREATE SEQUENCE statement
> - "Tutorial: Creating a Trigger that Generates a Primary Key for a Row Before It Is Inserted" to learn how to create a trigger that inserts the primary keys created by EVALUATIONS_SEQUENCE into the EVALUATIONS table

Dropping Sequences

To drop a sequence, use either the SQL Developer Connections frame and Drop tool, or the DDL statement DROP SEQUENCE.

This statement drops the sequence EVALUATIONS_SEQUENCE:

```
DROP SEQUENCE EVALUATIONS_SEQUENCE;
```

> **Caution:**
>
> Do not drop the sequence `EVALUATIONS_SEQUENCE`—you need it for Example 5-3. If you want to practice dropping sequences, create others and then drop them.

To drop a sequence using the Drop tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Sequences**.
3. In the list of sequences, right-click the name of the sequence to drop.
4. In the list of choices, click **Drop**.
5. In the Drop window, click **Apply**.

6. In the Confirmation window, click **OK**.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the DROP SEQUENCE statement

Creating and Managing Synonyms

A synonym is an alias for another schema object. Some reasons to use synonyms are security (for example, to hide the owner and location of an object) and convenience.

Examples of convenience are:

- Using a short synonym, such as `SALES`, for a long object name, such as `ACME_CO.SALES_DATA`
- Using a synonym for a renamed object, instead of changing that object name throughout the applications that use it

  For example, if your application uses a table named `DEPARTMENTS`, and its name changes to `DIVISIONS`, you can create a `DEPARTMENTS` synonym for that table and continue to reference it by its original name.

> **See Also:**
>
> *Oracle Database Concepts* for additional general information about synonyms

Creating Synonyms

To create a synonym, use either the SQL Developer tool Create Database Synonym or the DDL statement CREATE SYNONYM .

The following tutorial shows how to use the Create Database Synonym tool to create the synonym EMP for the EMPLOYEES table. The equivalent DDL statement is:

```
CREATE SYNONYM EMPL FOR EMPLOYEES;
```

To create the synonym EMP using the Create Database Synonym tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, right-click **Synonyms**.
3. In the list of choices, click **New Synonym**.
4. In the New Synonym window:
   a. In the Synonym Name field, type `EMPL`.
   b. In the Object Owner field, select **HR** from the menu.
   c. In the Object Name field, select **EMPLOYEES** from the menu.

The synonym refers to a specific schema object; in this case, the table EMPLOYEES.

d. Click **Apply**.

5. In the Confirmation window, click **OK**.

   The synonym EMPL is created. To see it, expand **Synonyms** in the Connections frame. You can now use EMPL instead of EMPLOYEES.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the CREATE SYNONYM statement

Dropping Synonyms

To drop a synonym, use either the SQL Developer Connections frame and Drop tool, or the DDL statement DROP SYNONYM.

This statement drops the synonym EMP:

```
DROP SYNONYM EMP;
```

To drop a synonym using the Drop tool:

1. In the Connections frame, expand **hr_conn**.
2. In the list of schema object types, expand **Synonyms**.
3. In the list of synonyms, right-click the name of the synonym to drop.
4. In the list of choices, click **Drop**.
5. In the Drop window, click **Apply**.
6. In the Confirmation window, click **OK**.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the DROP SYNONYM statement

5

Developing Stored Subprograms and Packages

Stored subprograms and packages can be used as building blocks for many different database applications.

About Stored Subprograms

A **stored subprogram** is a subprogram that is stored in the database. Because they are stored in the database, stored programs can be used as building blocks for many different database applications.

A **subprogram** is a PL/SQL unit that consists of SQL and PL/SQL statements that solve a specific problem or perform a set of related tasks. A subprogram can have parameters, whose values are supplied by the invoker. A subprogram can be either a procedure or a function. Typically, you use a procedure to perform an action and a function to compute and return a value.

Because stored subprograms are stored in the database, stored programs can be used as building blocks for many different database applications. A subprogram that is declared within another subprogram, or within an anonymous block, is called a **nested subprogram** or **local subprogram**. It cannot be invoked from outside the subprogram or block in which it is declared. An **anonymous block** is a block that is not stored in the database.

There are two kinds of stored subprograms:

- **Standalone subprogram**, which is created at schema level
- **Package subprogram**, which is created inside a package

Standalone subprograms are useful for testing pieces of program logic, but when you are sure that they work as intended, Oracle recommends that you put them into packages.

> **See Also:**
>
> - *Oracle Database Concepts* for general information about stored subprograms
> - *Oracle Database PL/SQL Language Reference* for complete information about PL/SQL subprograms

About Packages

A **package** is a PL/SQL unit that consists of related subprograms and the declared cursors and variables that they use. Oracle recommends that you put your subprograms into packages.

Some reasons that Oracle recommends that you put your subprograms into packages are: