10

Deploying an Oracle Database Application

After you develop your application, you can install it on other databases, called deployment environments, where other users can run it.

About Development and Deployment Environments

The database on which you develop your application is called the **development environment**. After developing your application, you can install it on other databases, called **deployment environments**, where other users can run it.

The first deployment environment is the **test environment**. In the test environment, you can thoroughly test the functionality of the application, determine whether it is structured correctly, and fix any problems before deploying it in the **production environment**.

You might also deploy your application to an **education environment**, either before or after deploying it to the production environment. An education environment provides a place for users to practice running the application without affecting other environments.

If the desired deployment environments do not exist in your organization, you can create them.

About Installation Scripts

An **installation script** can either have all the SQL statements needed to create the application or it can be a **primary script** that runs other scripts.

A **script** is a series of SQL statements in a file whose name ends with `.sql` (for example, `create_app.sql`). When you run a script in a client program such as SQL*Plus or SQL Developer, the SQL statements run in the order in which they appear in the script. A script whose SQL statements create an application is called an **installation script**.

To deploy an application, you run one or more installation scripts in the deployment environment. For a new application, you must create the installation scripts. For an older application, the installation scripts might exist, but if they do not, you can create them.

About DDL Statements and Schema Object Dependencies

An installation script contains DDL statements that create schema objects and, optionally, INSERT statements that load data into tables that DDL statements create. To create installation scripts correctly, and to run multiple installation scripts in the correct order, you must understand the dependencies between the schema objects of your application.

If the definition of object A references object B, then A depends on B. Therefore, you must create B before you create A. Otherwise, the statement that creates B either fails or creates B in an invalid state, depending on the object type.

For a complex application, the order for creating the objects is rarely obvious. Usually, you must consult the database designer or a diagram of the design.

> **See Also:**
>
> - *Oracle Database Development Guide* for more information about schema object dependencies
> - "About Data Definition Language (DDL) Statements"

About INSERT Statements and Constraints

When you run an installation script that contains INSERT statements, you must determine whether constraints could be violated when data from source tables (in the development environment) is inserted into new tables in the deployment environment.

For each source table in your application, you must determine whether any constraints could be violated when their data is inserted in the new table. If so, you must first disable those constraints, then insert the data, and then try to re-enable the constraints. If a data item violates a constraint, then you cannot re-enable that constraint until you correct the data item.

If you are simply inserting lookup data in correct order (as in "Loading the Data"), then constraints are not violated. Therefore, you do not need to disable them first.

If you are inserting data from an outside source (such as a file, spreadsheet, or older application), or from many tables that have much dependent data, disable the constraints before inserting the data.

Some possible ways to disable and re-enable the constraints are:

- Using SQL Developer, disable and re-enable the constraints one at a time:
  1. In the Connections frame, select the appropriate table.
  2. In the pane labeled with table name, select the subtab **Constraints**.
  3. In the list of all constraints on the table, change `ENABLED` to `DISABLED` (or the reverse).
- Edit the installation script, adding SQL statements that disable and re-enable each constraint.
- Create a SQL script with SQL statements that disable and enable each constraint.
- Find the constraints in the Oracle Database data dictionary, and create a SQL script with the SQL statements to disable and enable each constraint.

  For example, to find and enable the constraints used in the EVALUATIONS, PERFORMANCE_PARTS, and SCORES tables from "Creating Tables", enter these statements in the Worksheet:

```
SELECT 'ALTER TABLE '|| TABLE_NAME || ' DISABLE CONSTRAINT '||
  CONSTRAINT_NAME ||';'
  FROM user_constraints
 WHERE table_name IN ('EVALUATIONS','PERFORMANCE_PARTS','SCORES');

SELECT 'ALTER TABLE '|| TABLE_NAME || ' ENABLE CONSTRAINT '||
  CONSTRAINT_NAME ||';'
  FROM user_constraints
 WHERE table_name IN ('EVALUATIONS','PERFORMANCE_PARTS','SCORES');
```

> **See Also:**
>
> - "About the INSERT Statement"
> - "Ensuring Data Integrity in Tables"

Creating Installation Scripts

You can create installation scripts in SQL Developer or a text editor.

If an installation script needs only DDL and INSERT statements, then you can create it with either SQL Developer or any text editor. In SQL Developer, you can use either the Cart or the Database Export wizard. Oracle recommends the Cart for installation scripts that you expect to run in multiple deployment environments and the Database Export wizard for installation scripts that you expect to run in only one deployment environment.

If an installation script needs SQL statements that are neither DDL nor INSERT statements, then you must create it with a text editor.

This section explains how to create installation scripts with the Cart and the Database Export wizard, when and how to edit installation scripts that create sequences and triggers, and how create installation scripts for the application in Developing a Simple Oracle Database Application ("the sample application").

Creating Installation Scripts with the Cart

The SQL Developer Cart is a convenient tool for deploying Oracle Database objects from one or more database connections to a destination connection.

You drag and drop objects from the navigator frame into the Cart window, specify the desired options, and click the Export Cart icon to display the Export Objects dialog box. After you complete the information in that dialog box, SQL Developer creates a `.zip` file containing scripts (including a primary script) to create the objects in the schema of a desired destination connection.

To create installation scripts with the Cart:

1. In the SQL Developer window, click the menu **View**.
2. From the View menu, select **Cart**.

   The Cart window opens. The Export Cart icon is inactive (gray).

   > **Tip:**
   >
   > In the Cart window, for information about Cart user preferences, press the key **F1**.

3. In the Connections frame, select the schema objects that you want the installation script to create and drag them into the Cart window.

   In The Cart window, the Export Cart icon is now active (not gray).

4. For each Selected Object of type TABLE, if you want the installation script to export data, then select the option **Data**.
5. Click **Export Cart**.
6. In the Export Objects dialog box, enter the desired values in the fields.

   For information about these fields, see *Oracle SQL Developer User's Guide*.
7. Click **Apply**.

   SQL Developer creates a `.zip` file containing scripts (including a primary script) to create the objects in the schema of a desired destination connection.
8. In the primary script and the scripts that it runs, check that:
   - Referenced objects are created before their dependent objects.
   - Tables are created before data is inserted into them.

   If the installation scripts create sequences, see "Editing Installation Scripts that Create Sequences".

   If the installation scripts create triggers, see "Editing Installation Scripts that Create Sequences".

   If necessary, edit the installation files in the Worksheet or any text editor.

> **See Also:**
>
> *Oracle SQL Developer User's Guide* for more information about the Cart

Creating an Installation Script with the Database Export Wizard

To create an installation script in SQL Developer with the Database Export wizard, you specify the name of the installation script, the objects and data to export, and the desired options, and the wizard generates an installation script.

> **Note:**
>
> In the following procedure, you might have to enlarge the SQL Developer windows to see all fields and options.

To create an installation script with the Database Export wizard:

1. If you have not done so, create a directory for the installation script, separate from the Oracle Database installation directory (for example, `C:\my_exports`).
2. In the SQL Developer window, click the menu **Tools**.
3. From the menu, select **Database Export**.
4. In the Export Wizard - Step 1 of 5 (Source/Destination) window:
   a. In the Connection field, select your connection to the development environment.

b. Select the desired Export DDL options (and deselect any selected undesired options).

> **Note:**
>
> Do not deselect Terminator, or the installation script will fail.

c. If you do *not* want the installation script to export data, then deselect **Export Data**.

d. In the Save As field, accept the default `Single File` and type the full path name of the installation script (for example, `C:\my_exports\hr_export.sql`).

The file name must end with `.sql`.

e. Click **Next**.

5. In the Export Wizard - Step 2 of 5 (Types to Export) window:

a. Deselect the check boxes for the types that you do *not* want to export.

Selecting or deselecting **Toggle All** selects or deselects all check boxes.

b. Click **Next**.

6. In the Export Wizard - Step 3 of 5 (Specify Objects) window:

a. Click **More**.

b. In the Schema field, select your schema from the menu.

c. In the Type field, select from the menu either `ALL OBJECTS` or a specific object type (for example, `TABLE`).

d. Click **Lookup**.

A list of objects appears in the left frame. If the value of the Type field is `ALL OBJECTS`, then the list contains all objects in the selected schema. If the value of the Type field is a specific object type, then the list contains all objects of that type in the selected schema.

e. Move the objects that you want to export from the left frame to the right frame:

To move all objects, click **>>**. (To move all objects back, click **<<**.)

To move selected objects, select them and then click **>**. (To move selected objects back, select them and click **<**.)

f. (Optional) Repeat steps 6.c through 6.e for other object types.

g. Click **Next**.

If you deselected `Export Data` in the Source/Destination window, then the Export Summary window appears—go to step 8.

If you did *not* deselect `Export Data` in the Source/Destination window, then the Export Wizard - Step 4 of 5 (Specify Data) window appears. The lower frame lists the objects that you specified in the Specify Objects window.

7. In the Specify Data window:

a. Move the objects whose data you *do not* want to export from the lower frame to the upper frame:

To move all objects, click the double upward arrow icon. (To move all objects back, click the double downward arrow icon.)

To move selected objects, select them and then click the single upward arrow icon.

b. Click **Next**.

8. In the Export Wizard - Step 5 of 5 (Export Summary) window, click **Finish**.

   The Exporting window opens, showing that exporting is occurring. When exporting is complete, the Exporting window closes, and the Worksheet shows the contents of the installation script that you specified in the Source/Destination window.

9. In the installation script, check that:

   - Referenced objects are created before their dependent objects.
   - Tables are created before data is inserted into them.

   If necessary, edit the file in the Worksheet or any text editor.

> **See Also:**
>
> *Oracle SQL Developer User's Guide* for more information about the Database Export wizard

Editing Installation Scripts that Create Sequences

If your application uses the sequence to generate unique keys, and you *will not* insert the data from the source tables into the corresponding new tables, then you might want to edit the START WITH value in the installation script.

For a sequence, SQL Developer generates a CREATE SEQUENCE statement whose START WITH value is relative to the current value of the sequence in the development environment.

If your application uses the sequence to generate unique keys, and you *will not* insert the data from the source tables into the corresponding new tables, then you might want to edit the START WITH value in the installation script.

You can edit the installation script in either the Worksheet or any text editor.

> **See Also:**
>
> "Tutorial: Creating a Sequence"

Editing Installation Scripts that Create Triggers

If your application has a BEFORE INSERT trigger on a source table, and you *will* insert data from that source table into the corresponding new table, you must decide if

you want the trigger to fire before each INSERT statement in the installation script inserts data into the new table.

For example, NEW_EVALUATION_TRIGGER (created in "Tutorial: Creating a Trigger that Generates a Primary Key for a Row Before It Is Inserted") fires before a row is inserted into the EVALUATIONS table. The trigger generates the unique number for the primary key of that row, using EVALUATIONS_SEQUENCE.

The source EVALUATIONS table is populated with primary keys. If you do not want the installation script to put new primary key values in the new EVALUATIONS table, then you must edit the CREATE TRIGGER statement in the installation script as shown in bold font:

```
CREATE OR REPLACE
TRIGGER NEW_EVALUATION_TRIGGER
BEFORE INSERT ON EVALUATIONS
FOR EACH ROW
BEGIN
  IF :NEW.evaluation_id IS NULL THEN
    :NEW.evaluation_id := evaluations_sequence.NEXTVAL
  END IF;
END;
```

Also, if the current value of the sequence is not greater than the maximum value in the primary key column, then you must make it greater.

You can edit the installation script in either the Worksheet or any text editor.

Two alternatives to editing the installation script are:

- Change the trigger definition in the source file and then re-create the installation script.

  For information about changing triggers, see "Changing Triggers".
- Disable the trigger before running the data installation script, and then re-enable it afterward.

  For information about disabling and enabling triggers, see "Disabling and Enabling Triggers".

> **See Also:**
>
> "Creating Triggers"

Creating Installation Scripts for the Sample Application

You can create installation scripts for the sample application.

These scripts are for the application in Developing a Simple Oracle Database Application:

- **schemas.sql**, which does in the deployment environment what you did in the development environment in "Creating the Schemas for the Application" and "Granting Privileges to the Schemas"
- **objects.sql**, which does in the deployment environment what you did in the development environment in "Creating the Schema Objects and Loading the Data"
- **employees.sql**, which does in the deployment environment what you did in the development environment in "Creating the employees_pkg Package"

- **admin.sql**, which does in the deployment environment what you did in the development environment in "Creating the admin_pkg Package"
- **create_app.sql**, a primary script that runs the preceding scripts, thereby deploying the sample application in the deployment environment

You can create the scripts in any order. To create schemas.sql and create_app.sql, you must use a text editor. To create the other scripts, you can use either a text editor or SQL Developer.

Creating Installation Script schemas.sql

The installation script schemas.sql does in the deployment environment what you did in the development environment in "Creating the Schemas for the Application" and "Granting Privileges to the Schemas".

To create schemas.sql, enter the following text in any text editor and save the file as schemas.sql.

> **Caution:**
>
> Choose secure passwords. For guidelines for secure passwords, see *Oracle Database Security Guide*.

```
-----------------
-- Create schemas
-----------------

DROP USER app_data CASCADE;

CREATE USER app_data IDENTIFIED BY password
DEFAULT TABLESPACE USERS
QUOTA UNLIMITED ON USERS
ENABLE EDITIONS;

DROP USER app_code CASCADE;

CREATE USER app_code IDENTIFIED BY password
DEFAULT TABLESPACE USERS
QUOTA UNLIMITED ON USERS
ENABLE EDITIONS;

DROP USER app_admin CASCADE;

CREATE USER app_admin IDENTIFIED BY password
DEFAULT TABLESPACE USERS
QUOTA UNLIMITED ON USERS
ENABLE EDITIONS;

DROP USER app_user CASCADE;

CREATE USER app_user IDENTIFIED BY password
ENABLE EDITIONS;

DROP USER app_admin_user CASCADE;

CREATE USER app_admin_user IDENTIFIED BY password
```

```
ENABLE EDITIONS;

-----------------------------
-- Grant privileges to schemas
-----------------------------

GRANT CREATE SESSION TO app_data;
GRANT CREATE TABLE, CREATE VIEW, CREATE TRIGGER, CREATE SEQUENCE TO app_data;
GRANT SELECT ON HR.DEPARTMENTS TO app_data;
GRANT SELECT ON HR.EMPLOYEES TO app_data;
GRANT SELECT ON HR.JOB_HISTORY TO app_data;
GRANT SELECT ON HR.JOBS TO app_data;

GRANT CREATE SESSION, CREATE PROCEDURE, CREATE SYNONYM TO app_code;

GRANT CREATE SESSION, CREATE PROCEDURE, CREATE SYNONYM TO app_admin;

GRANT CREATE SESSION, CREATE SYNONYM TO app_user;

GRANT CREATE SESSION, CREATE SYNONYM TO app_admin_user;
```

See Also:

"Schemas for the Application" for descriptions of the schemas for the sample application

Creating Installation Script objects.sql

The installation script objects.sql does in the deployment environment what you did in the development environment in "Creating the Schema Objects and Loading the Data".

You can create objects.sql using either a text editor or SQL Developer.

To create objects.sql in any text editor, enter the following text and save the file as objects.sql. For password, use the password that schema.sql specifies when it creates the user app_data.

Note:

The INSERT statements that load the data work only if the deployment environment has a standard HR schema. If it does not, then either use SQL Developer to create a script that loads the new tables (in the deployment environment) with data from the source tables (in the development environment) or modify the INSERT statements in the following script.

```
------------------------
-- Create schema objects
------------------------

CONNECT app_data/password

CREATE TABLE jobs#
( job_id      VARCHAR2(10)
```

```
              CONSTRAINT jobs_pk PRIMARY KEY,
  job_title   VARCHAR2(35)
              CONSTRAINT jobs_job_title_not_null NOT NULL,
  min_salary  NUMBER(6)
              CONSTRAINT jobs_min_salary_not_null NOT NULL,
  max_salary  NUMBER(6)
              CONSTRAINT jobs_max_salary_not_null NOT NULL
)
/

CREATE TABLE departments#
( department_id    NUMBER(4)
                   CONSTRAINT departments_pk PRIMARY KEY,
  department_name  VARCHAR2(30)
                   CONSTRAINT dept_department_name_not_null NOT NULL
                   CONSTRAINT dept_department_name_unique UNIQUE,
  manager_id       NUMBER(6)
)
/

CREATE TABLE employees#
( employee_id     NUMBER(6)
                  CONSTRAINT employees_pk PRIMARY KEY,
  first_name      VARCHAR2(20)
                  CONSTRAINT emp_first_name_not_null NOT NULL,
  last_name       VARCHAR2(25)
                  CONSTRAINT emp_last_name_not_null NOT NULL,
  email_addr      VARCHAR2(25)
                  CONSTRAINT emp_email_addr_not_null NOT NULL,
  hire_date       DATE
                  DEFAULT TRUNC(SYSDATE)
                  CONSTRAINT emp_hire_date_not_null NOT NULL
                  CONSTRAINT emp_hire_date_check
                    CHECK(TRUNC(hire_date) = hire_date),
  country_code    VARCHAR2(5)
                  CONSTRAINT emp_country_code_not_null NOT NULL,
  phone_number    VARCHAR2(20)
                  CONSTRAINT emp_phone_number_not_null NOT NULL,
  job_id          CONSTRAINT emp_job_id_not_null NOT NULL
                  CONSTRAINT emp_to_jobs_fk REFERENCES jobs#,
  job_start_date  DATE
                  CONSTRAINT emp_job_start_date_not_null NOT NULL,
                  CONSTRAINT emp_job_start_date_check
                    CHECK(TRUNC(JOB_START_DATE) = job_start_date),
  salary          NUMBER(6)
                  CONSTRAINT emp_salary_not_null NOT NULL,
  manager_id      CONSTRAINT emp_mgrid_to_emp_empid_fk REFERENCES employees#,
  department_id   CONSTRAINT emp_to_dept_fk REFERENCES departments#
)
/

CREATE TABLE job_history#
( employee_id  CONSTRAINT job_hist_to_emp_fk REFERENCES employees#,
  job_id       CONSTRAINT job_hist_to_jobs_fk REFERENCES jobs#,
  start_date   DATE
               CONSTRAINT job_hist_start_date_not_null NOT NULL,
  end_date     DATE
               CONSTRAINT job_hist_end_date_not_null NOT NULL,
  department_id
             CONSTRAINT job_hist_to_dept_fk REFERENCES departments#
             CONSTRAINT job_hist_dept_id_not_null NOT NULL,
```

```
            CONSTRAINT job_history_pk PRIMARY KEY(employee_id,start_date),
            CONSTRAINT job_history_date_check CHECK( start_date < end_date )
)
/

CREATE EDITIONING VIEW jobs AS SELECT * FROM jobs#
/
CREATE EDITIONING VIEW departments AS SELECT * FROM departments#
/
CREATE EDITIONING VIEW employees AS SELECT * FROM employees#
/
CREATE EDITIONING VIEW job_history AS SELECT * FROM job_history#
/

CREATE OR REPLACE TRIGGER employees_aiufer
AFTER INSERT OR UPDATE OF salary, job_id ON employees FOR EACH ROW
DECLARE
  l_cnt NUMBER;
BEGIN
  LOCK TABLE jobs IN SHARE MODE;  -- Ensure that jobs does not change
                                 -- during the following query.
  SELECT COUNT(*) INTO l_cnt
  FROM jobs
  WHERE job_id = :NEW.job_id
  AND :NEW.salary BETWEEN min_salary AND max_salary;

  IF (l_cnt<>1) THEN
    RAISE_APPLICATION_ERROR( -20002,
      CASE
        WHEN :new.job_id = :old.job_id
        THEN 'Salary modification invalid'
        ELSE 'Job reassignment puts salary out of range'
      END );
  END IF;
END;
/

CREATE OR REPLACE TRIGGER jobs_aufer
AFTER UPDATE OF min_salary, max_salary ON jobs FOR EACH ROW
WHEN (NEW.min_salary > OLD.min_salary OR NEW.max_salary < OLD.max_salary)
DECLARE
  l_cnt NUMBER;
BEGIN
  LOCK TABLE employees IN SHARE MODE;

  SELECT COUNT(*) INTO l_cnt
  FROM employees
  WHERE job_id = :NEW.job_id
  AND salary NOT BETWEEN :NEW.min_salary and :NEW.max_salary;

  IF (l_cnt>0) THEN
    RAISE_APPLICATION_ERROR( -20001,
      'Salary update would violate ' || l_cnt || ' existing employee records' );
  END IF;
END;
/

CREATE SEQUENCE employees_sequence START WITH 210;
CREATE SEQUENCE departments_sequence START WITH 275;

------------
```

```
-- Load data
------------

INSERT INTO jobs (job_id, job_title, min_salary, max_salary)
SELECT job_id, job_title, min_salary, max_salary
  FROM HR.JOBS
/

INSERT INTO departments (department_id, department_name, manager_id)
SELECT department_id, department_name, manager_id
  FROM HR.DEPARTMENTS
/

INSERT INTO employees (employee_id, first_name, last_name, email_addr,
  hire_date, country_code, phone_number, job_id, job_start_date, salary,
  manager_id, department_id)
SELECT employee_id, first_name, last_name, email, hire_date,
  CASE WHEN phone_number LIKE '011.%'
    THEN '+' || SUBSTR( phone_number, INSTR( phone_number, '.' )+1,
      INSTR( phone_number, '.', 1, 2 ) -  INSTR( phone_number, '.' ) - 1 )
    ELSE '+1'
  END country_code,
  CASE WHEN phone_number LIKE '011.%'
    THEN SUBSTR( phone_number, INSTR(phone_number, '.', 1, 2 )+1 )
    ELSE phone_number
  END phone_number,
  job_id,
  NVL( (SELECT MAX(end_date+1)
        FROM HR.JOB_HISTORY jh
        WHERE jh.employee_id = employees.employee_id), hire_date),
  salary, manager_id, department_id
  FROM HR.EMPLOYEES
/

INSERT INTO job_history (employee_id, job_id, start_date, end_date,
  department_id)
SELECT employee_id, job_id, start_date, end_date, department_id
  FROM HR.JOB_HISTORY
/

COMMIT;

-----------------------------
-- Add foreign key constraint
-----------------------------

ALTER TABLE departments#
ADD CONSTRAINT dept_to_emp_fk
FOREIGN KEY(manager_id) REFERENCES employees#;

---------------------------------------------
-- Grant privileges on schema objects to users
---------------------------------------------

GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO app_code;
GRANT SELECT ON departments TO app_code;
GRANT SELECT ON jobs TO app_code;
GRANT SELECT, INSERT on job_history TO app_code;
GRANT SELECT ON employees_sequence TO app_code;

GRANT SELECT, INSERT, UPDATE, DELETE ON jobs TO app_admin;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON departments TO app_admin;
GRANT SELECT ON employees_sequence TO app_admin;
GRANT SELECT ON departments_sequence TO app_admin;

GRANT SELECT ON jobs TO app_admin_user;
GRANT SELECT ON departments TO app_admin_user;
```

> **See Also:**
>
> - "Schema Objects of the Application" for descriptions of the schema objects of the sample application
> - "Creating Installation Scripts with the Cart"
> - "Creating an Installation Script with the Database Export Wizard"

Creating Installation Script employees.sql

The installation script employees.sql does in the deployment environment what you did in the development environment in "Creating the employees_pkg Package".

You can create employees.sql using either a text editor or SQL Developer.

To create employees.sql in any text editor, enter the following text and save the file as employees.sql. For password, use the password that schema.sql specifies when it creates the user app_code.

```
-----------------------
-- Create employees_pkg
-----------------------

CONNECT app_code/password

CREATE SYNONYM employees FOR app_data.employees;
CREATE SYNONYM departments FOR app_data.departments;
CREATE SYNONYM jobs FOR app_data.jobs;
CREATE SYNONYM job_history FOR app_data.job_history;

CREATE OR REPLACE PACKAGE employees_pkg
AS
  PROCEDURE get_employees_in_dept
    ( p_deptno     IN     employees.department_id%TYPE,
      p_result_set IN OUT SYS_REFCURSOR );

  PROCEDURE get_job_history
    ( p_employee_id  IN     employees.department_id%TYPE,
      p_result_set   IN OUT SYS_REFCURSOR );

  PROCEDURE show_employee
    ( p_employee_id  IN     employees.employee_id%TYPE,
      p_result_set   IN OUT SYS_REFCURSOR );

  PROCEDURE update_salary
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_salary  IN employees.salary%TYPE );

  PROCEDURE change_job
```

```
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_job     IN employees.job_id%TYPE,
      p_new_salary  IN employees.salary%TYPE := NULL,
      p_new_dept    IN employees.department_id%TYPE := NULL );
END employees_pkg;
/

CREATE OR REPLACE PACKAGE BODY employees_pkg
AS
  PROCEDURE get_employees_in_dept
    ( p_deptno     IN     employees.department_id%TYPE,
      p_result_set IN OUT SYS_REFCURSOR )
  IS
     l_cursor SYS_REFCURSOR;
  BEGIN
    OPEN p_result_set FOR
      SELECT e.employee_id,
        e.first_name || ' ' || e.last_name name,
        TO_CHAR( e.hire_date, 'Dy Mon ddth, yyyy' ) hire_date,
        j.job_title,
        m.first_name || ' ' || m.last_name manager,
        d.department_name
      FROM employees e INNER JOIN jobs j ON (e.job_id = j.job_id)
        LEFT OUTER JOIN employees m ON (e.manager_id = m.employee_id)
        INNER JOIN departments d ON (e.department_id = d.department_id)
      WHERE e.department_id = p_deptno ;
  END get_employees_in_dept;

  PROCEDURE get_job_history
    ( p_employee_id  IN     employees.department_id%TYPE,
      p_result_set   IN OUT SYS_REFCURSOR )
  IS
  BEGIN
    OPEN p_result_set FOR
      SELECT e.First_name || ' ' || e.last_name name, j.job_title,
        e.job_start_date start_date,
        TO_DATE(NULL) end_date
      FROM employees e INNER JOIN jobs j ON (e.job_id = j.job_id)
      WHERE e.employee_id = p_employee_id
      UNION ALL
      SELECT e.First_name || ' ' || e.last_name name,
        j.job_title,
        jh.start_date,
        jh.end_date
      FROM employees e INNER JOIN job_history jh
        ON (e.employee_id = jh.employee_id)
        INNER JOIN jobs j ON (jh.job_id = j.job_id)
      WHERE e.employee_id = p_employee_id
      ORDER BY start_date DESC;
  END get_job_history;

  PROCEDURE show_employee
    ( p_employee_id  IN     employees.employee_id%TYPE,
      p_result_set   IN OUT sys_refcursor )
  IS
  BEGIN
    OPEN p_result_set FOR
      SELECT *
      FROM (SELECT TO_CHAR(e.employee_id) employee_id,
              e.first_name || ' ' || e.last_name name,
              e.email_addr,
```

```
              TO_CHAR(e.hire_date,'dd-mon-yyyy') hire_date,
              e.country_code,
              e.phone_number,
              j.job_title,
              TO_CHAR(e.job_start_date,'dd-mon-yyyy') job_start_date,
              to_char(e.salary) salary,
              m.first_name || ' ' || m.last_name manager,
              d.department_name
            FROM employees e INNER JOIN jobs j on (e.job_id = j.job_id)
              RIGHT OUTER JOIN employees m ON (m.employee_id = e.manager_id)
              INNER JOIN departments d ON (e.department_id = d.department_id)
            WHERE e.employee_id = p_employee_id)
      UNPIVOT (VALUE FOR ATTRIBUTE IN (employee_id, name, email_addr, hire_date,
        country_code, phone_number, job_title, job_start_date, salary, manager,
        department_name) );
  END show_employee;

  PROCEDURE update_salary
    ( p_employee_id IN employees.employee_id%type,
      p_new_salary  IN employees.salary%type )
  IS
  BEGIN
    UPDATE employees
    SET salary = p_new_salary
    WHERE employee_id = p_employee_id;
  END update_salary;

  PROCEDURE change_job
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_job     IN employees.job_id%TYPE,
      p_new_salary  IN employees.salary%TYPE := NULL,
      p_new_dept    IN employees.department_id%TYPE := NULL )
  IS
  BEGIN
    INSERT INTO job_history (employee_id, start_date, end_date, job_id,
      department_id)
    SELECT employee_id, job_start_date, TRUNC(SYSDATE), job_id, department_id
      FROM employees
      WHERE employee_id = p_employee_id;

    UPDATE employees
    SET job_id = p_new_job,
      department_id = NVL( p_new_dept, department_id ),
      salary = NVL( p_new_salary, salary ),
      job_start_date = TRUNC(SYSDATE)
    WHERE employee_id = p_employee_id;
  END change_job;
END employees_pkg;
/

---------------------------------------------
-- Grant privileges on employees_pkg to users
---------------------------------------------

GRANT EXECUTE ON employees_pkg TO app_user;
GRANT EXECUTE ON employees_pkg TO app_admin_user;
```

> **See Also:**
>
> - "Creating Installation Scripts with the Cart"
> - "Creating an Installation Script with the Database Export Wizard"

Creating Installation Script admin.sql

The installation script admin.sql does in the deployment environment what you did in the development environment in "Creating the admin_pkg Package".

You can create admin.sql using either a text editor or SQL Developer.

To create admin.sql in any text editor, enter the following text and save the file as admin.sql. For password, use the password that schema.sql specifies when it creates the user app_admin.

```
-------------------
-- Create admin_pkg
-------------------

CONNECT app_admin/password

CREATE SYNONYM departments FOR app_data.departments;
CREATE SYNONYM jobs FOR app_data.jobs;
CREATE SYNONYM departments_sequence FOR app_data.departments_sequence;

CREATE OR REPLACE PACKAGE admin_pkg
AS
  PROCEDURE update_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE := NULL,
      p_min_salary  IN jobs.min_salary%TYPE := NULL,
      p_max_salary  IN jobs.max_salary%TYPE := NULL );

  PROCEDURE add_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE,
      p_min_salary  IN jobs.min_salary%TYPE,
      p_max_salary  IN jobs.max_salary%TYPE );

  PROCEDURE update_department
    ( p_department_id     IN departments.department_id%TYPE,
      p_department_name   IN departments.department_name%TYPE := NULL,
      p_manager_id        IN departments.manager_id%TYPE := NULL,
      p_update_manager_id IN BOOLEAN := FALSE );

  FUNCTION add_department
    ( p_department_name   IN departments.department_name%TYPE,
      p_manager_id        IN departments.manager_id%TYPE )
    RETURN departments.department_id%TYPE;

END admin_pkg;
/

CREATE OR REPLACE PACKAGE BODY admin_pkg
AS
  PROCEDURE update_job
```

```
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE := NULL,
      p_min_salary  IN jobs.min_salary%TYPE := NULL,
      p_max_salary  IN jobs.max_salary%TYPE := NULL )
  IS
  BEGIN
    UPDATE jobs
    SET job_title  = NVL( p_job_title, job_title ),
        min_salary = NVL( p_min_salary, min_salary ),
        max_salary = NVL( p_max_salary, max_salary )
    WHERE job_id = p_job_id;
  END update_job;

  PROCEDURE add_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE,
      p_min_salary  IN jobs.min_salary%TYPE,
      p_max_salary  IN jobs.max_salary%TYPE )
  IS
  BEGIN
    INSERT INTO jobs ( job_id, job_title, min_salary, max_salary )
    VALUES ( p_job_id, p_job_title, p_min_salary, p_max_salary );
  END add_job;

  PROCEDURE update_department
    ( p_department_id     IN departments.department_id%TYPE,
      p_department_name   IN departments.department_name%TYPE := NULL,
      p_manager_id        IN departments.manager_id%TYPE := NULL,
      p_update_manager_id IN BOOLEAN := FALSE )
  IS
  BEGIN
    IF ( p_update_manager_id ) THEN
      UPDATE departments
      SET department_name = NVL( p_department_name, department_name ),
          manager_id = p_manager_id
      WHERE department_id = p_department_id;
    ELSE
      UPDATE departments
      SET department_name = NVL( p_department_name, department_name )
      WHERE department_id = p_department_id;
    END IF;
  END update_department;

  FUNCTION add_department
    ( p_department_name   IN departments.department_name%TYPE,
      p_manager_id        IN departments.manager_id%TYPE )
    RETURN departments.department_id%TYPE
  IS
    l_department_id departments.department_id%TYPE;
  BEGIN
    INSERT INTO departments ( department_id, department_name, manager_id )
      VALUES ( departments_sequence.NEXTVAL, p_department_name, p_manager_id )
      RETURNING department_id INTO l_department_id;
    RETURN l_department_id;
  END add_department;

END admin_pkg;
/

----------------------------------------
-- Grant privileges on admin_pkg to user
```

```
----------------------------------------

GRANT EXECUTE ON admin_pkg TO app_admin_user;
```

> **See Also:**
>
> - "Creating Installation Scripts with the Cart"
> - "Creating an Installation Script with the Database Export Wizard"

Creating Primary Installation Script create_app.sql

The primary installation script `create_app.sql` runs the other four installation scripts for the sample application in the correct order, which deploys the sample application in the deployment environment.

To create the `create_app.sql` script, enter the following text in any text editor and save the file as `create_app.sql`.

```
@schemas.sql
@objects.sql
@employees.sql
@admin.sql
```

Deploying the Sample Application

You can deploy the sample application using installation scripts.

Use the installation scripts that you created in "Creating Installation Scripts for the Sample Application".

> **Note:**
>
> For the following procedures, you need the name and password of a user who has the CREATE USER and DROP USER system privileges.

To deploy the sample application using SQL*Plus:

1. Copy the installation scripts that you created in "Creating Installation Scripts for the Sample Application" to the deployment environment.
2. In the deployment environment, connect to Oracle Database as a user with the CREATE USER and DROP USER system privileges.
3. At the `SQL>` prompt, run the primary installation script:

   ```
   @create_app.sql
   ```

   The primary installation script runs the other four installation scripts for the sample application in the correct order, thereby deploying the sample application in the deployment environment.

To deploy the sample application using SQL Developer:

1. If necessary, create a connection to the deployment environment.

   For Connection Name, enter a name that is *not* the name of the connection to the development environment.

2. Copy the installation scripts that you created in "Creating Installation Scripts for the Sample Application" to the deployment environment.

3. Connect to Oracle Database as a user with the CREATE USER and DROP USER system privileges in the deployment environment.

   A new pane appears. On its tab is the name of the connection to the deployment environment. The pane has two subpanes, Worksheet and Query Builder.

4. In the Worksheet pane, type the command for running the primary installation script:

   ```
   @create_app.sql
   ```

5. Click the icon **Run Script**.

   The primary installation script runs the other four installation scripts for the sample application in the correct order, thereby deploying the sample application in the deployment environment. The output appears in the Script Output pane, under the Worksheet pane.

   In the Connections frame, if you expand the connection to the deployment environment, and then expand the type of each object that the sample application uses, you see the objects of the sample application.

> **See Also:**
>
> - *SQL*Plus User's Guide and Reference* for more information about using scripts in SQL*Plus
> - *Oracle SQL Developer User's Guide* for more information about running scripts in SQL Developer

Checking the Validity of an Installation

After installing your application in a deployment environment, you can check its validity using SQL Developer.

- In the Connections frame:
  1. Expand the connection to the deployment environment.
  2. Examine the definitions of the new objects.
- In the Reports pane:
  1. Expand **Data Dictionary Reports**.

     A list of data dictionary reports appears.
  2. Expand **All Objects**.

     A list of objects reports appears.

3. Select **All Objects**.

   The Select Connection window appears.

4. In the Connection field, select from the menu the connection to the deployment environment.

5. Click **OK**.

6. In the Enter Bind Values window, select either `Owner` or `Object Name`.

7. Click **Apply**.

   The message `Displaying Results`shows, followed by the results.

   For each object, this report lists the Owner, Object Type, Object Name, Status (Valid or Invalid), Date Created, and Last DDL. Last DDL is the date of the last DDL operation that affected the object.

8. In the Reports pane, select **Invalid Objects**.

9. In the Enter Bind Values window, click **Apply**.

   For each object whose Status is Invalid, this report lists the Owner, Object Type, and Object Name.

> **See Also:**
>
> *Oracle SQL Developer User's Guide* for more information about SQL Developer reports

Archiving the Installation Scripts

After you verify that the installation of your application is valid, Oracle recommends that you archive your installation scripts in a source code control system.

Before doing so, add comments to each file, documenting its creation date and purpose. If you ever must deploy the same application to another environment, you can use these archived files.

> **See Also:**
>
> *Oracle Database Utilities* for information about Oracle Data Pump, which enables very high-speed movement of data and metadata from one database to another