C

# Writing Advanced Scripts



ORACLE

# Objectives

After completing this appendix, you should be able to do the following:

- Describe the type of problems that are solved by using SQL to generate SQL
- Write a script that generates a script of `DROP TABLE` statements
- Write a script that generates a script of `INSERT INTO` statements



ORACLE

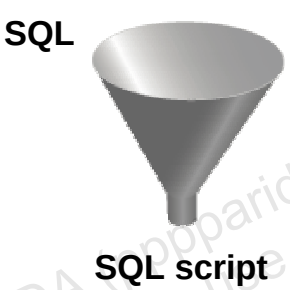


**Objectives**

In this appendix, you learn how to write a SQL script to generates a SQL script.

# Using SQL to Generate SQL

- SQL can be used to generate scripts in SQL.
- The data dictionary:
  - Is a collection of tables and views that contain database information
  - Is created and maintained by the Oracle server










## Using SQL to Generate SQL

SQL can be a powerful tool to generate other SQL statements. In most cases, this involves writing a script file. You can use SQL from SQL to:

- Avoid repetitive coding
- Access information from the data dictionary
- Drop or re-create database objects
- Generate dynamic predicates that contain run-time parameters

The examples used in this lesson involve selecting information from the data dictionary. The data dictionary is a collection of tables and views that contain information about the database. This collection is created and maintained by the Oracle server. All data dictionary tables are owned by the SYS user. Information stored in the data dictionary includes names of Oracle server users, privileges granted to users, database object names, table constraints, and audit information. There are four categories of data dictionary views. Each category has a distinct prefix that reflects its intended use.

| Prefix | Description |
|---|---|
| USER_ | Contains details of objects owned by the user |
| ALL_ | Contains details of objects to which the user has been granted access rights, in addition to objects owned by the user |
| DBA_ | Contains details of users with DBA privileges to access any object in the database |
| V$_ | Stores information about database server performance and locking; available only to the DBA |

# Creating a Basic Script

```
SELECT 'CREATE TABLE ' || table_name ||
       '_test ' || 'AS SELECT * FROM '
       || table_name ||' WHERE 1=2;'
       AS "Create Table Script"
FROM   user_tables;
```

| | Create Table Script |
|---|---|
| 1 | CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2; |
| 2 | CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2; |
| 3 | CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2; |
| 4 | CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2; |
| 5 | CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2; |
| 6 | CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2; |
| 7 | CREATE TABLE COUNTRIES_test AS SELECT * FROM COUNTRIES WHERE 1=2; |



ORACLE



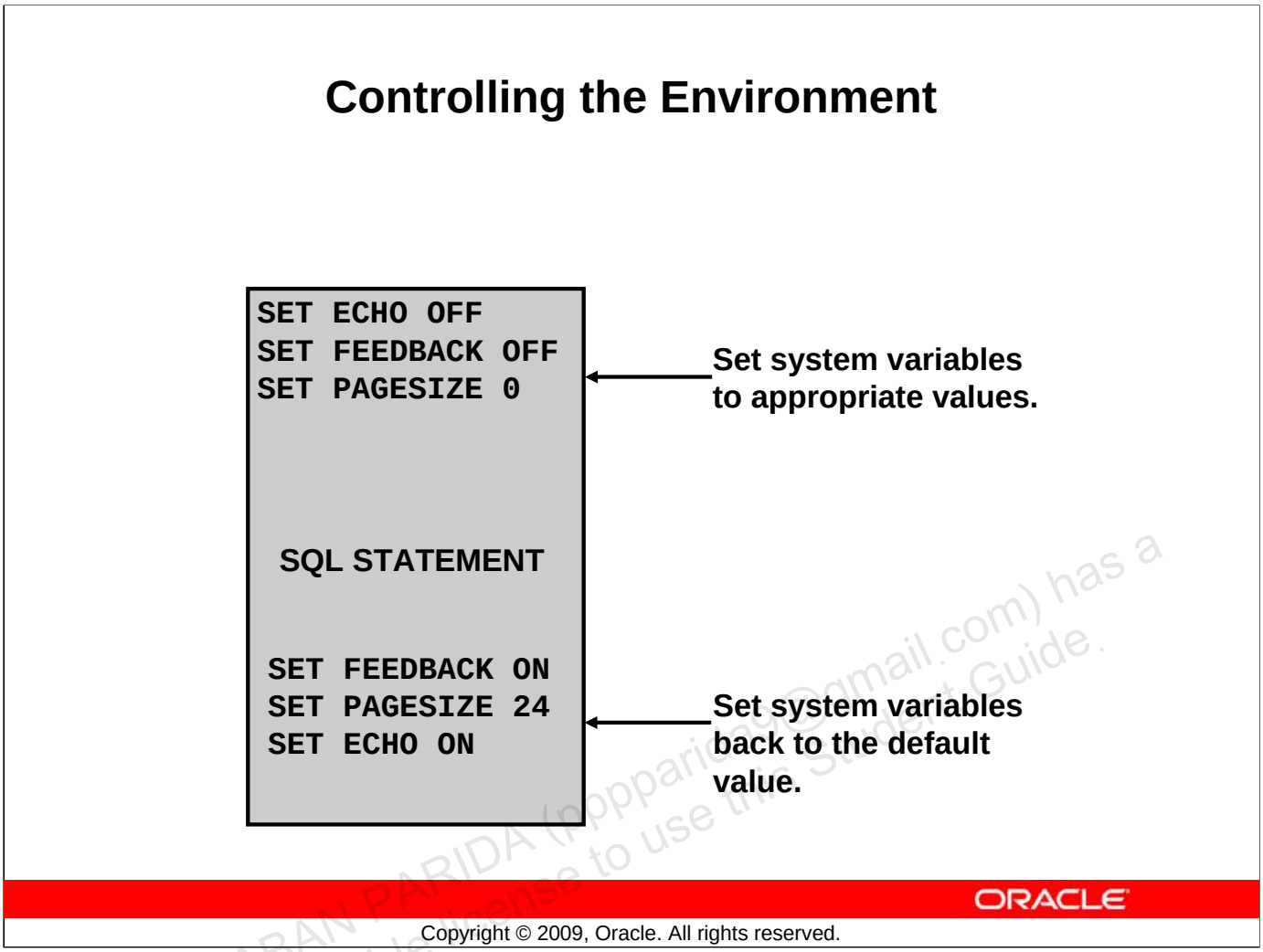
## A Basic Script

The example in the slide produces a report with CREATE TABLE statements from every table you own. Each CREATE TABLE statement produced in the report includes the syntax to create a table using the table name with a suffix of _test and having only the structure of the corresponding existing table. The old table name is obtained from the TABLE_NAME column of the data dictionary view USER_TABLES.

The next step is to enhance the report to automate the process.

**Note:** You can query the data dictionary tables to view various database objects that you own. The data dictionary views frequently used include:

- USER_TABLES: Displays description of the user's own tables
- USER_OBJECTS: Displays all the objects owned by the user
- USER_TAB_PRIVS_MADE: Displays all grants on objects owned by the user
- USER_COL_PRIVS_MADE: Displays all grants on columns of objects owned by the user








## Controlling the Environment

To execute the SQL statements that are generated, you must capture them in a file that can then be run. You must also plan to clean up the output that is generated and make sure that you suppress elements such as headings, feedback messages, top titles, and so on. In SQL Developer, you can save these statements to a script.

**Note:** Some of the SQL*Plus statements are not supported by SQL Worksheet. For the complete list of SQL*Plus statements that are supported, and not supported by SQL Worksheet, refer to the topic titled *SQL*Plus Statements Supported and Not Supported in SQL Worksheet* in the SQL Developer online Help

# The Complete Picture

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';'
FROM   user_objects
WHERE  object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```



ORACLE



## The Complete Picture

The output of the command in the slide is saved into a file called `dropem.sql` in SQL Developer. To save the output into a file in SQL Developer, you use the Save File option under the Script Output pane. The `dropem.sql` file contains the following data. This file can now be started from SQL Developer by locating the script file, loading it, and executing it.

| | 'DROPTABLE'\|\|OBJECT_NAME\|\|';' |
|---|---|
| 1 | DROP TABLE REGIONS; |
| 2 | DROP TABLE COUNTRIES; |
| 3 | DROP TABLE LOCATIONS; |
| 4 | DROP TABLE DEPARTMENTS; |
| 5 | DROP TABLE JOBS; |
| 6 | DROP TABLE EMPLOYEES; |
| 7 | DROP TABLE JOB_HISTORY; |

# Dumping the Contents of a Table to a File

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0


SELECT
 'INSERT INTO departments_test VALUES
  (' || department_id || ', ''' || department_name ||
   ''', ''' || location_id || ''');'
  AS "Insert Statements Script"
FROM   departments
/


SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```

ORACLE





## Dumping Table Contents to a File

Sometimes, it is useful to have the values for the rows of a table in a text file in the format of an `INSERT INTO VALUES` statement. This script can be run to populate the table in case the table has been dropped accidentally.

The example in the slide produces `INSERT` statements for the `DEPARTMENTS_TEST` table, captured in the `data.sql` file using the output option in SQL Developer.

The contents of the `data.sql` script file are as follows:

```
INSERT INTO departments_test VALUES
  (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
  (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
  (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
  (60, 'IT', 1400);
...
```

## Dumping the Contents of a Table to a File

| Source | Result |
|---|---|
| '''X''' | 'X' |
| '''' | ' |
| ''''\|\|department_name\|\|'''' | 'Administration' |
| ''', ''' | ',' |
| ''');' | '); |



ORACLE



### Dumping Table Contents to a File (continued)

You may have noticed the large number of single quotation marks in the previous slide. A set of four single quotation marks produces one single quotation mark in the final statement. Also remember that character and date values must be surrounded by quotation marks.

Within a string, to display one single quotation mark, you need to prefix it with another single quotation mark. For example, in the fifth example in the slide, the surrounding quotation marks are for the entire string. The second quotation mark acts as a prefix to display the third quotation mark. Thus, the result is one single quotation mark followed by the parenthesis, followed by the semicolon.

# Generating a Dynamic Predicate

```
COLUMN my_col NEW_VALUE dyn_where_clause

SELECT DECODE('&&deptno', null,
DECODE ('&&hiredate', null, ' ',
'WHERE hire_date=TO_DATE('''||'&&hiredate'',''DD-MON-YYYY'')'),
DECODE ('&&hiredate', null,
'WHERE department_id = ' || '&&deptno',
'WHERE department_id = ' || '&&deptno' ||
' AND hire_date = TO_DATE('''||'&&hiredate'',''DD-MON-YYYY''')'))
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```



ORACLE



## Generating a Dynamic Predicate

The example in the slide generates a SELECT statement that retrieves data of all employees in a department who were hired on a specific day. The script generates the WHERE clause dynamically.

**Note:** After the user variable is in place, you must use the UNDEFINE command to delete it.

The first SELECT statement prompts you to enter the department number. If you do not enter any department number, the department number is treated as null by the DECODE function, and the user is then prompted for the hire date. If you do not enter any hire date, the hire date is treated as null by the DECODE function and the dynamic WHERE clause that is generated is also a null, which causes the second SELECT statement to retrieve all rows from the EMPLOYEES table.

**Note:** The NEW_V[ALUE] variable specifies a variable to hold a column value. You can reference the variable in TTITLE commands. Use NEW_VALUE to display column values or the date in the top title. You must include the column in a BREAK command with the SKIP PAGE action. The variable name cannot contain a pound sign (#). NEW_VALUE is useful for master/detail reports in which there is a new master record for each page.

## Generating a Dynamic Predicate (continued)

**Note:** Here, the hire date must be entered in the DD-MON-YYYY format.

The SELECT statement in the slide can be interpreted as follows:

```
IF   (<<deptno>> is not entered)  THEN
     IF  (<<hiredate>> is not entered)  THEN
           return empty string
     ELSE
           return the string 'WHERE hire_date =
 TO_DATE('<<hiredate>>', 'DD-MON-YYYY')'
 ELSE
        IF (<<hiredate>> is not entered)  THEN
              return the string 'WHERE department_id = <<deptno>>
 entered'
        ELSE
              return the string 'WHERE department_id = <<deptno>>
 entered
                                                AND hire_date =
 TO_DATE(' <<hiredate>>', 'DD-MON-YYYY')'
END IF
```

The returned string becomes the value of the DYN_WHERE_CLAUSE variable, which will be used in the second SELECT statement.

**Note:** Use SQL*Plus for these examples.

When the first example in the slide is executed, the user is prompted for the values for DEPTNO and HIREDATE:

```
Enter value for deptno: 10
```

```
Enter value for hire_date: 17-sep-1987
```

The following value for MY_COL is generated:

```
MY_COL
--------------------------------------------------------------------------------
where department_id = 10 and hire_date =to_date('17-sep-1987','dd-mon-yyyy')
```

When the second example in the slide is executed, the following output is generated:

```
LAST_NAME
-------------------------
Whalen
```

# Summary

In this appendix, you should have learned that:

- You can write a SQL script to generate another SQL script
- Script files often use the data dictionary
- You can capture the output in a file



ORACLE





**Summary**

SQL can be used to generate SQL scripts. These scripts can be used to avoid repetitive coding, drop or re-create objects, get help from the data dictionary, and generate dynamic predicates that contain run-time parameters.