

CHAPTER 14 The Data Dictionary, Scripting, and Reporting

Oracle SQL By Example, Fourth Edition by Alice Rischert. Published by Prentice Hall. Copyright © 2008 by Pearson Education, Inc.

CHAPTER OBJECTIVES

In this chapter, you will learn about:

- ▶ The Oracle Data Dictionary Views
- ▶ Writing SQL Scripts
- ▶ Creating SQL Developer Reports

The Oracle data dictionary is a set of tables and views that contains data about the database; it is also sometimes referred to as the *catalog*. Oracle uses the data dictionary internally for many purposes (for instance, to determine whether a SQL statement contains valid column and table names, to determine the privileges of an individual user, to check if a column is indexed).

Although the SQL Developer's Details tab displays a lot about an object, you will find it useful to know about and query the wealth of information available in the data dictionary views. What you learn in this chapter about

data dictionary views will add significantly to your understanding of Oracle technology and the related database concepts.

In this chapter, you will also learn about SQL Developer's built-in reporting capabilities. SQL Developer provides an extensive list of useful reports and offers the capability to create your own user-defined reports.

This chapter will expand your knowledge of the SQL*Plus and SQL Developer execution environments and their respective scripting and reporting capabilities. You will discover that you can simplify the writing of SQL statements and some database administration tasks by writing SQL scripts that generate and execute other SQL statements.

615
616

LAB 14.1 The Oracle Data Dictionary Views

LAB OBJECTIVES

After this lab, you will be able to:

- Query the Data Dictionary

The data dictionary has two distinct sets of views: the *static* data dictionary views and the *dynamic* data

dictionary views, also referred to as *dynamic performance views*, or *V\$TABLES* (“V-dollar tables”).

The Static Data Dictionary Views

The static data dictionary stores details about database objects, such as tables, indexes, and views. It also lists information about referential integrity constraints and indexed columns. Whenever a new object is added or an object is changed, data about the object is recorded in the data dictionary.

Most of the static dictionary views begin with the prefix `USER_`, `ALL_`, or `DBA_`. The `USER_` views show information belonging to the user querying the data dictionary. For example, when you log in as `STUDENT`, the views that begin with the `USER_` prefix show all the objects belonging to the `STUDENT` owner. The `ALL_` views show the same information, any information granted to the `STUDENT` user by another user, and any public objects. You'll learn how to grant and receive access rights in [Chapter 15](#), “Security.” The `DBA_` views show all objects in the entire database, but you need database administrator (DBA) privileges or the `SELECT ANY DICTIONARY` privilege to be able to query these views.

The Dynamic Data Dictionary Views

The dynamic views begin with V\$ and are typically used by a DBA to monitor the system. They are called dynamic because they are continuously updated by the background processes in the Oracle instance but never by the user. [Table 14.1](#) shows the different types of data dictionary views.

616

617

TABLE 14.1 Overview of Oracle Data Dictionary Views

PREFIX	PURPOSE
USER_	Objects that belong to the user querying
ALL_	Objects that belong to the user and objects that are accessible to the user
DBA_	All objects in the entire database, accessible only to users with DBA or SELECT ANY DICTIONARY privileges
V\$	Dynamic performance views, accessible only to users with DBA privileges or the SELECT ANY DICTIONARY privileges

The Dictionary

The collection of static and dynamic data dictionary tables and views, along with a description of each, is

listed in the view called **DICTIONARY**, also known by the synonym **DICT**. You can examine the columns of the **DICT** view by issuing the **SQL*Plus DESCRIBE** command.



A synonym is another name for a database object; for example, instead of using **DICTIONARY**, you can use the shorter synonym **DICT**. You'll learn about synonyms and their use in [Chapter 15](#).

```
SQL> DESC dict
```

Name	Null?	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

The column **TABLE_NAME** contains the name of the individual data dictionary view accessible to you, together with a brief description in the **COMMENTS** column.

For example, to find information about sequences in the database, you can query the **DICT** view. The column **TABLE_NAME** stores the names of the data dictionary views, in uppercase. The following query results in the

selection of all data dictionary views with the letters SEQ in their name.

```
SELECT table_name, comments
FROM dict
WHERE table_name LIKE '%SEQ%'
```

TABLE_NAME	COMMENTS
ALL_SEQUENCES	Description of SEQUENCES accessible to the user
DBA_SEQUENCES	Description of all SEQUENCES in the database
USER_SEQUENCES	Description of the user's own SEQUENCES
SEQ	Synonym for USER_SEQUENCES

4 rows selected.

617

618

Four different data dictionary views contain information about sequences.



If you do not have DBA access or the SELECT ANY DICTIONARY privileges, you cannot see the DBA_SEQUENCES view.

To display the columns of the ALL_SEQUENCES data dictionary view, issue the DESCRIBE command at the SQL*Plus prompt.

```
SQL> DESC ALL_SEQUENCES
```

Name	Null?	Type
SEQUENCE_OWNER	NOT NULL	VARCHAR2(30)
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

If you are unclear about the meanings of the different columns, you can query yet another view, named `DICT_COLUMNS`. It provides a definition for each column.

```
SELECT column_name, comments
FROM dict_columns
WHERE table_name = 'ALL_SEQUENCES'
```

COLUMN_NAME	COMMENTS
-----	-----
SEQUENCE_OWNER	Name of the owner of the sequence
SEQUENCE_NAME	SEQUENCE name
MIN_VALUE	Minimum value of the sequence
...	
CACHE_SIZE	Number of sequence numbers to cache
LAST_NUMBER	Last sequence number written to disk

9 rows selected.

To find out which individual sequences are in the `STUDENT` schema, query the view.

```
SELECT sequence_name
FROM seq
```

SEQUENCE_NAME

COURSE_NO_SEQ
INSTRUCTOR_ID_SEQ
SECTION_ID_SEQ
STUDENT_ID_SEQ

4 rows selected.



SQL Developer offers many useful data dictionary reports that provide detailed information about the database. You will learn about these reports in [Lab 14.2](#).

LAB 14.1 EXERCISES

If you have performed most of the exercises in the previous chapters, your results will differ from the results shown in the outputs of this chapter as you have added new objects and altered existing objects in the STUDENT schema. To bring the STUDENT schema back to its original state, run the `rebuildStudent.sql` script. This script drops the STUDENT account-related tables, re-creates the tables, and reloads the data. If you added the supplemental tables mentioned in the previous chapters, you can drop them by using the `sql_book_drop_extra_tables.sql` script. (These scripts are part of the downloaded scripts you used when you first created the STUDENT user.)

To run the script within SQL Developer, open the file. All buttons are grayed out. To execute the script, click

the Run Script icon (or press F5) to execute the script. After you select a database connection, the result will show in the Script Output tab, on the bottom of the screen.

619

620

- a)** For the USER_OBJECTS view, what information is stored in the columns CREATED, LAST_DDL_TIME, and STATUS?
- b)** Which data dictionary view lists only tables in the STUDENT schema?
- c)** Query the data dictionary view USER_TAB_COLUMNS for the GRADE table and describe the information in the columns DATA_TYPE, DATA_LENGTH, NULLABLE, and DATA_DEFAULT.
- d)** Show a list of all indexes and their columns for the ENROLLMENT table.
- e)** Write a query that displays a list of all the sequences in the STUDENT schema and the current value of each.
- f)** Execute the following two SQL statements. The first statement creates a view, and the second queries the data dictionary view called USER_VIEWS. What information is stored in

the TEXT column of USER_VIEWS? Drop the view after you answer these questions.

```
CREATE OR REPLACE VIEW my_test AS
SELECT first_name, instructor_id
FROM instructor
SELECT view_name, text
FROM user_views
WHERE view_name = 'MY_TEST'
```

- g)** Execute the following query. What do you observe?

```
SELECT constraint_name,
table_name, constraint_type
FROM user_constraints
```

- h)** What columns are listed in the data dictionary view USER_CONS_COLUMNS?

- i)** Execute the following SQL statement. Describe the result.

```
SELECT username
FROM all_users
```

- j)** Execute the following query. What do you observe about the result?

```
SELECT segment_name,
segment_type, bytes/1024
```

```
FROM user_segments
WHERE segment_name = 'ZIPCODE'
AND segment_type = 'TABLE'
```

LAB 14.1 EXERCISE ANSWERS

- a) For the USER_OBJECTS view, what information is stored in the columns CREATED, LAST_DDL_TIME, and STATUS?

ANSWER: The CREATED column shows the creation date of an object. The LAST_DDL_TIME column indicates when an object was last modified via a DDL command, such as when a column was added to a table or when a view was recompiled. The STATUS column displays whether an object is valid or invalid.

The resulting output may vary, depending on the objects in your schema.

```
SELECT column_name, comments
FROM dict_columns
WHERE table_name = 'USER_OBJECTS'
AND column_name IN ('STATUS', 'LAST_DDL_TIME',
                    'CREATED')

COLUMN_NAME      COMMENTS
-----
CREATED          Timestamp for the creation of the object
LAST_DDL_TIME    Timestamp for the last DDL change (including
                  GRANT and REVOKE) to the object
STATUS           Status of the object

3 rows selected.
```

A view may become invalid if the underlying table is modified or dropped. Other objects, such as PL/SQL procedures, packages, or functions, may become invalid if dependent objects are modified and they subsequently need to be recompiled.

If you are unclear about the meaning of a particular column, refer to the `DICT_COLUMNS` view for information.

```
SELECT column_name, comments
FROM dict_columns
WHERE table_name = 'USER_OBJECTS'
AND column_name IN ('STATUS', 'LAST_DDL_TIME',
                    'CREATED')
```

COLUMN_NAME	COMMENTS
CREATED	Timestamp for the creation of the object
LAST_DDL_TIME	Timestamp for the last DDL change (including GRANT and REVOKE) to the object
STATUS	Status of the object

3 rows selected.

b) Which data dictionary view lists only tables in the `STUDENT` schema?

ANSWER: You can find out which data dictionary table contains this information by querying the `DICT` view. The view is `USER_TABLES`.

```
SELECT table_name
       FROM user_tables
TABLE_NAME
-----
COURSE
...
ZIPCODE

10 rows selected.
```

- c) Query the data dictionary view `USER_TAB_COLUMNS` for the `GRADE` table and describe the information in the columns `DATA_TYPE`, `DATA_LENGTH`, `NULLABLE`, and `DATA_DEFAULT`.

ANSWER: The column `DATA_TYPE` shows the data type of the column, `DATA_LENGTH` displays the length of the column in bytes, and there is either a Y or an N in the column `NULLABLE`, indicating whether NULL values are allowed in the column. The column `DATA_DEFAULT` represents the default value for the column, if any.

```
SELECT table_name, column_name,
       data_type, data_length,
       nullable, data_default
```

621

```

SELECT table_name, column_name, data_type, data_length,
       nullable, data_default
FROM user_tab_columns
WHERE table_name = 'GRADE'

```

TABLE_NAME	COLUMN_NAME	DATA_TYP	DATA_LENGTH	N	DATA_
GRADE	STUDENT_ID	NUMBER	22	N	
...					
GRADE	NUMERIC_GRADE	NUMBER	22	N	0
...					
GRADE	MODIFIED_BY	VARCHAR2	30	N	
GRADE	MODIFIED_DATE	DATE	7	N	

10 rows selected.

Note the zero value in the last column, named DATA_DEFAULT. This means the column called NUMERIC_GRADE has a column default value of zero. This value is inserted into a table's row if the NUMERIC_GRADE column is not specified during an INSERT operation. For example, the following INSERT statement does not list the NUMERIC_GRADE column and, therefore, the NUMERIC_GRADE column is zero; alternatively, you can use the DEFAULT keyword discussed in [Chapter 11](#), “Insert, Update, and Delete.”

```

INSERT INTO GRADE
  (student_id, section_id,
   grade_type_code,
   grade_code_occurrence,

```



```
        created_by, created_date,  
        modified_by, modified_date)  
VALUES  
    (102, 89, 'FI',  
     2,  
     'ARISCHERT', SYSDATE,  
     'ARISCHERT', SYSDATE)
```

1 row created.

- d) Show a list of all indexes and their columns for the ENROLLMENT table.

Answer: The data dictionary view
USER_IND_COLUMNS lists the desired result.

```
SELECT index_name, table_name, column_name,  
       column_position  
FROM user_ind_columns  
WHERE table_name = 'ENROLLMENT'  
ORDER BY 1, 4
```

INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION
ENR_PK	ENROLLMENT	STUDENT_ID	1
ENR_PK	ENROLLMENT	SECTION_ID	2
ENR_SECT_FK_I	ENROLLMENT	SECTION_ID	1

3 rows selected.

The ENROLLMENT table has two indexes:
ENR_PK and ENR_SECT_FK_I. The first index,
a unique index created by the primary key
constraint, has the columns STUDENT_ID and

SECTION_ID, in that order.
COLUMN_POSITION shows the order of the columns within the index. The second index is the foreign key column SECTION_ID.

622

If you want to show just the listing of indexes, without the individual indexed column, you can query USER_INDEXES. This view also indicates whether an index is unique.

623

You can find details about function-based indexes listed in the
USER_IND_EXPRESSIONS view.

- e) Write a query that displays a list of all the sequences in the STUDENT schema and the current value of each.

ANSWER: The USER_SEQUENCES data dictionary view shows the sequence name and the current value of the sequence. The resulting output may vary, depending on the sequences in your schema.

```
SELECT sequence_name, last_number
FROM user_sequences
SEQUENCE_NAME                LAST_NUMBER
-----
COURSE_NO_SEQ                451
INSTRUCTOR_ID_SEQ            111
SECTION_ID_SEQ                157
STUDENT_ID_SEQ                400

4 rows selected.
```

f) Execute the following two SQL statements. The first statement creates a view, and the second queries the data dictionary view called `USER_VIEWS`. What information is stored in the `TEXT` column of `USER_VIEWS`? Drop the view after you answer these questions.

```
CREATE OR REPLACE VIEW my_test AS
SELECT first_name, instructor_id
FROM instructor
SELECT view_name, text
FROM user_views
WHERE view_name = 'MY_TEST'
```

ANSWER: The `TEXT` column of the `USER_VIEWS` data dictionary view stores the view's defining SQL statement.

```
VIEW_NAME  TEXT
-----
MY_TEST    SELECT first_name, instructor_id
           FROM instructor

1 row selected.
```

From [Chapter 13](#), “Indexes, Sequences, and Views,” recall the definition of a view as a stored query. The query is stored in the column named `TEXT` of `USER_VIEWS`.

OBJECT DEPENDENCIES

An object, such as a view, synonym, procedure, function, or package, may depend on other objects. For example, the view MY_TEST depends on the INSTRUCTOR table. You can find out about these dependencies in the USER_DEPENDENCIES view. The query shows that this object is a view and that it references the INSTRUCTOR table. While this is easy to determine with a simple view, some objects are more complicated, and querying this view helps identify the effect of any potential change.

```
SELECT name, type, referenced_name
FROM user_dependencies
WHERE name = 'MY_TEST'
```

NAME	TYPE	REFERENCED_NAME
MY_TEST	VIEW	INSTRUCTOR

```
1 row selected.
```

SQL Developer displays this information as part of the Dependencies tab, discussed in [Chapter 13](#).

You drop the MY_TEST view by using the DROP VIEW command.

```
DROP VIEW my_test
View dropped.
```

g) Execute the following query. What do you observe?

```
SELECT constraint_name,  
       table_name, constraint_type  
FROM user_constraints
```

ANSWER: The output shows the constraints on the various tables. The foreign key constraint is listed as constraint type R (for referential integrity constraint), the NOT NULL and CHECK constraints are shown as constraint type C, and the primary key constraints are displayed as constraint type P. The SECTION table has a unique constraint listed as constraint type U.

CONSTRAINT_NAME	TABLE_NAME	C
-----	-----	-
CRSE_CRSE_FK	COURSE	R
...		
SYS_C001441	GRADE	C
ENR_STU_FK	ENROLLMENT	R
...		
SECT_SECT2_UK	SECTION	U
...		
ZIP_PK	ZIPCODE	P
...		
ZIP_MODIFIED_BY_NNULL	ZIPCODE	C

94 rows selected.

Any constraint that is not explicitly named receives a system-assigned name (for example, SYS_C001441).

The USER_CONSTRAINTS view contains additional useful columns, particularly for referential integrity constraints. For example, query the view for the foreign key constraint called ENR_STU_FK. The next following result shows the name of the primary key constraint in the R_CONSTRAINT_NAME column. This constraint is referenced by the foreign key.

```
SELECT r_owner, r_constraint_name, delete_rule
FROM user_constraints
WHERE constraint_name = 'ENR_STU_FK'
```

R_OWNER	R_CONSTRAINT_NAME	DELETE_RU
STUDENT	STU_PK	NO ACTION

1 row selected.

You can see in the result that the delete rule on the ENR_STU_FK constraint specifies NO ACTION, which means any deletion of a student row (parent record) is restricted if dependent enrollment rows (child records with the same STUDENT_ID) exist. This is in contrast to a CASCADE, which means that if a parent record is deleted, the children are

automatically deleted. If the referential integrity constraint is ON DELETE SET NULL, you would see the value SET NULL in the DELETE_RULE column.

The referential integrity constraints prevent the creation of orphan rows (that is, enrollment records without corresponding students). Also, the parent table may not be dropped unless the foreign key constraint is dropped. To disable constraints, use the ALTER TABLE command. Alternatively, the parent table may be dropped using the DROP TABLE command with the CASCADE CONSTRAINTS clause, automatically dropping the foreign key constraints.

You can retrieve constraint information for an individual table by using the Constraints tab in SQL Developer (see [Figure 14.2](#)).

FIGURE 14.2 The Constraints tab in SQL Developer

Constraint Name	Constraint	Reference	Stoken	Delete Rule
ENR_CREATED_BY_NULL	Check	(null)	(null)	(null)
ENR_CREATED_DATE_NULL	Check	(null)	(null)	(null)
ENR_ENROLL_DATE_NULL	Check	(null)	(null)	(null)
ENR_MODIFIED_BY_NULL	Check	(null)	(null)	(null)
ENR_MODIFIED_DATE_NULL	Check	(null)	(null)	(null)
ENR_PK	Primary_Key	(null)	(null)	(null)
ENR_SECTION_ID_NULL	Check	(null)	(null)	(null)
ENR_SECT_FK	Foreign_Key	SECTION	SECT_PK	NO ACTION
ENR_STUDENT_ID_NULL	Check	(null)	(null)	(null)
ENR_STU_FK	Foreign_Key	STUDENT	STU_PK	NO ACTION

OTHER CONSTRAINT TYPES

[Table 14.2](#) lists the constraint types. In addition to the constraint types already mentioned in this chapter, it lists the view constraint with check option (V) and the view constraint with the read-only option (O).

625

626

TABLE 14.2 Constraint Types

CONSTRAINT TYPE	DESCRIPTION
R	Referential integrity constraint
C	Check constraint, including NOT NULL constraint
P	Primary key constraint
U	Unique constraint
V	View constraint with check option
O	View constraint with read-only option

DISTINGUISHING NOT NULL CONSTRAINTS FROM CHECK CONSTRAINTS

The NOT NULL constraint is listed as a check constraint, and you can distinguish this type from other

user-defined check constraints by looking at the SEARCH_CONDITION column. The following query shows the constraints of the GRADE_TYPE table. For example, the NOT NULL constraint called GRTYP_DESCRIPTION_NNULL on the DESCRIPTION column lists the NOT NULL column, with the column name in quotes (in case of case-sensitive column names), together with the words IS NOT NULL. Compare this to the check constraint GRTYP_GRADE_TYPE_CODE_LENGTH, which checks whether the length of the GRADE_TYPE_CODE column is exactly 2.

```
SELECT constraint_name, search_condition
FROM user_constraints
WHERE constraint_type = 'C'
AND table_name = 'GRADE_TYPE'
```

CONSTRAINT_NAME	SEARCH_CONDITION
GRTYP_DESCRIPTION_NNULL	"DESCRIPTION" IS NOT NULL
...	
GRTYP_GRADE_TYPE_CODE_LENGTH	LENGTH(grade_type_code)=2

7 rows selected.

h) What columns are listed in the data dictionary view USER_CONS_COLUMNS?

ANSWER: The columns are OWNER, CONSTRAINT_NAME, TABLE_NAME, COLUMN_NAME, and POSITION.

This data dictionary view shows the columns referenced in the constraint. A query against the view illustrates this on the example of the primary key constraint ENR_PK, which consists of the two columns STUDENT_ID and SECTION_ID.

```
SELECT constraint_name, column_name, position
FROM user_cons_columns
WHERE constraint_name = 'ENR_PK'
```

CONSTRAINT_NAME	COLUMN_NAME	POSITION
ENR_PK	STUDENT_ID	1
ENR_PK	SECTION_ID	2

2 rows selected.

626

- i) Execute the following SQL statement. Describe the result.

```
SELECT username
FROM all_users
```

627

ANSWER: It shows a list of all the users in the database. The resulting output may vary, depending on your database.

```
USERNAME
-----
SYS
SYSTEM
...
SCOTT
...
STUDENT

15 rows selected.
```

Note that in this example output, there are two users, named SYS and SYSTEM. The SYS user is the owner of the Oracle data dictionary. Never log in as this “super user” unless you are an experienced Oracle DBA or unless Oracle instructs you to do so. Otherwise, you might inadvertently perform actions that could adversely affect the database. The SYSTEM user has DBA privileges but does not own the data dictionary. You will learn more about these two user accounts in [Chapter 15](#).

Another useful view is the USER_USERS view. Following is a query that displays information about the current user or schema. It shows your login name and the name of the default tablespace on which any tables or indexes you create are stored, unless you explicitly specify another tablespace. It also shows when your account was created.

```
SELECT username, default_tablespace, created
FROM user_users
```

USERNAME	DEFAULT_TABLESPACE	CREATED
STUDENT	USERS	04-MAY-08

1 row selected.

j) Execute the following query. What do you observe about the result?

```
SELECT segment_name, segment_type,
bytes/1024
FROM user_segments
WHERE segment_name = 'ZIPCODE'
AND segment_type = 'TABLE'
```

ANSWER: The query displays the size of the ZIPCODE table.

SEGMENT_NAME	SEGMENT_TYPE	BYTES/1024
ZIPCODE	TABLE	64

1 row selected.

627

628

The most common segment types are tables and indexes. The USER_SEGMENT view shows the storage, in bytes, for a particular segment.

Dividing the bytes by 1024 displays the size in kilobytes (K). Your actual number of bytes may vary from the figure listed here, depending on the storage parameter chosen for the default tablespace in your individual user account.

To see a list of the different available tablespaces, you query the USER_TABLESPACES or

DBA_TABLESPACES view. You get a result similar to the following.

```
SELECT tablespace_name
FROM user_tablespaces
ORDER BY tablespace_name
TABLESPACE_NAME
-----
INDX
SYSTEM
TEMP
USERS

4 rows selected.
```

To find out how much space is available in total on each of the tablespaces, you write a SQL statement against the view USER_FREE_SPACE. The result shows you the available megabytes (MB) for each tablespace. To learn more about tablespace and space management topics, see the *Oracle Database Administrator's Guide*.

```
SELECT tablespace_name, SUM(bytes)/1024/1024
FROM user_free_space
GROUP BY tablespace_name
TABLESPACE_NAME                                SUM(BYTES)/1024/1024
-----
INDX                                            24.8125
SYSTEM                                         14.6796875
USERS                                           82.8125

3 rows selected.
```

628

Lab 14.1 Quiz

In order to test your progress, you should be able to answer the following questions.

- 1) The data dictionary contains data about the database.
☐ a) True
☐ b) False
- 2) The data dictionary view USER_OBJECTS stores information about tables, indexes, and sequences.
☐ a) True
☐ b) False
- 3) The dynamic data dictionary views are updated only by the Oracle database.
☐ a) True
☐ b) False
- 4) The ALL_TABLES data dictionary view shows all the tables in the entire database.
☐ a) True
☐ b) False

5) The OBJ view is a public synonym for the USER_OBJECTS view.

_____ a) True

_____ b) False

ANSWERS APPEAR IN APPENDIX A.

629

630

LAB 14.2 Scripting and Reporting

LAB OBJECTIVES

After this lab, you will be able to:

- ▶ Write and Execute Scripts
- ▶ Develop Interactive SQL Statements and Reports

So far, you have written and executed many SQL statements. This lab shows you how you can collect SQL statements and SQL*Plus commands into a script file for execution. You will find it useful to write SQL statements that allow user input and execute commands that create and execute other SQL statements.

In [Lab 14.1](#), you learned about the many available data dictionary views. This knowledge will help you

understand and appreciate SQL Developer's supplied data dictionary reports and help you write your own SQL Developer reports.

This lab contains many references specific to either SQL*Plus or SQL Developer. There are sometimes different ways to obtain the solution with either of the tools. As mentioned at the beginning of the book, it is useful to know how to use both tools. SQL Developer is a very useful productivity tool, but you cannot erase decades' worth of currently used SQL*Plus scripts and functionality. Furthermore, SQL*Plus continues to be the tool of choice for executing scripts from an operating system prompt.

What Is a Script?

Before you implement database changes to the production environment, you will develop the DDL and DML and then test the changes in a development environment. Then you need to perform the same changes again in the QA environment and finally in the production environment.

A *script* is essentially a collection of statements to be executed. Typically, scripts are tested and modified multiple times due to changes, retests, preproduction dry

runs, and so on. To ensure that you will not miss any of the SQL commands, it is useful to collect these statements in a script. Scripting the statements ensures that the same commands can be executed repeatedly and in the same sequence.

When you created the objects in the STUDENT schema, you executed a script containing the DDL statements to create tables, indexes, sequences, and constraints together with the DML statements to insert the data. This script also contains SELECT statements to validate the number of inserted rows.

630

631

A script is useful if you need to run a set of SQL statements at specific time periods or on an ad hoc basis. Rather than retype and remember the same statements, you can execute the script to perform the task.

A script can contains variables that allow for user input prompts or different runtime parameters.

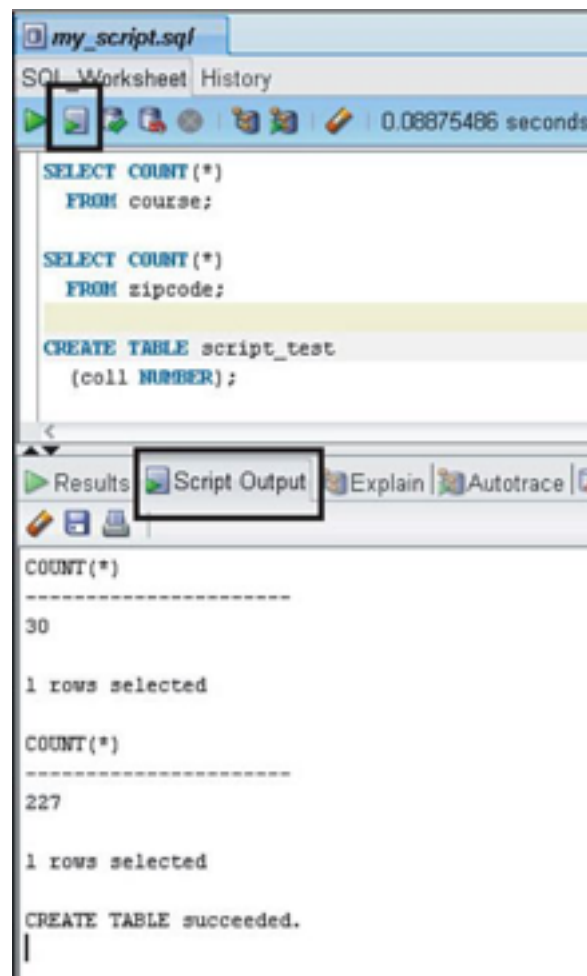
Executing a Script in SQL Developer

To execute a script, you need to load the SQL statements into the SQL Statement box. To do so, you select File, Open menu and choose the appropriate file name. After the statements are displayed in the SQL worksheet, you click the Run Script icon (or press the F5 key) to execute

the statements. Each SQL statement must end with a semicolon or a backslash on a separate line.

[Figure 14.3](#) shows the Run Script icon on the top of the screen and the result of the script execution in the Script Output tab. The my_script.sql script shown is a simple script consisting of two SELECT statements and a CREATE TABLE statement.

FIGURE 14.3 Run Script icon and Script Output tab



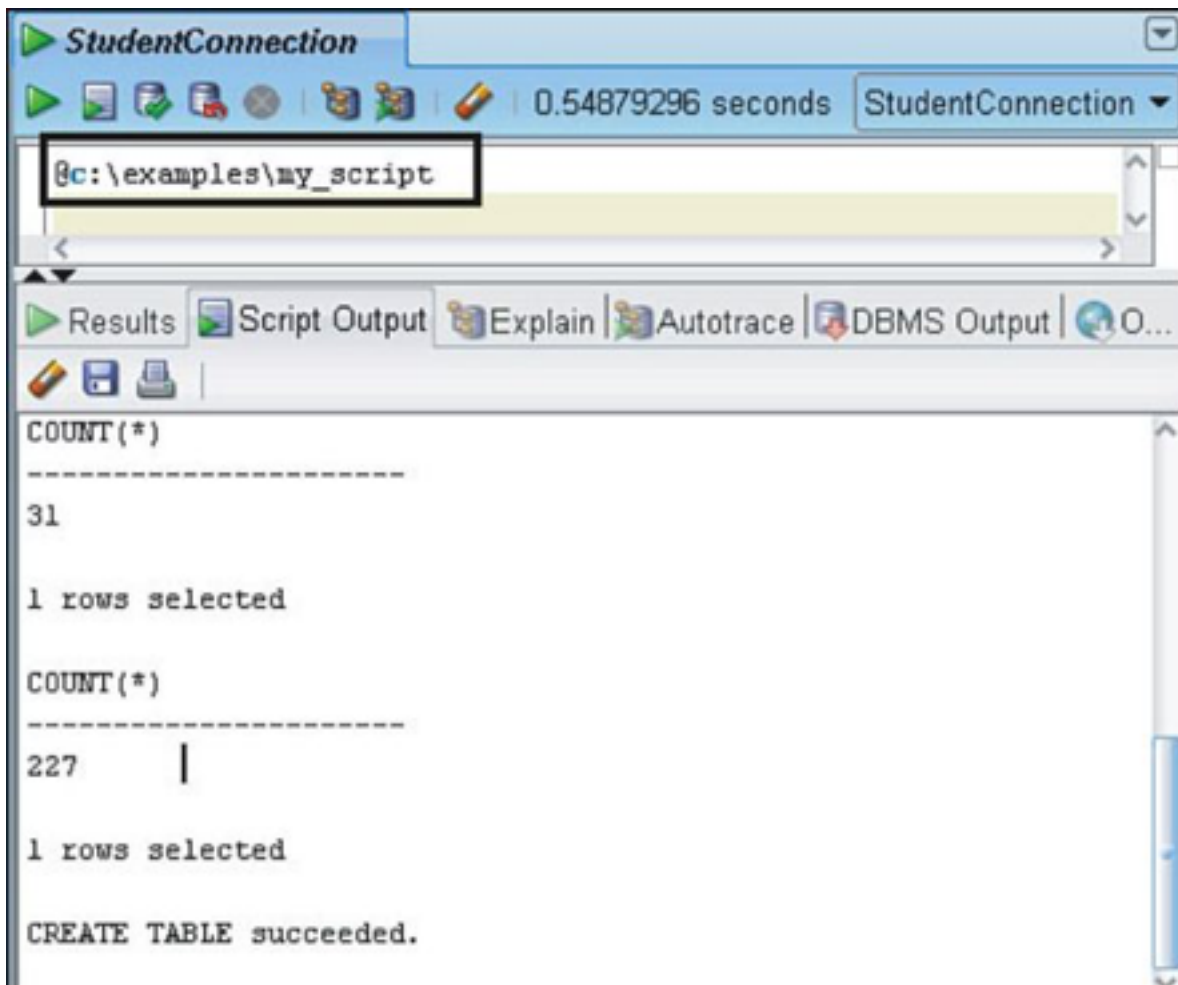
USING THE START @ COMMAND

Instead of selecting File, Open to retrieve the file and then executing the script, you can use the SQL*Plus @ or START command, as discussed in [Chapter 2](#), “SQL: The Basics.” You can use this command not only from a SQL*Plus prompt but also within SQL Developer. The @ or START command runs the script. [Figure 14.4](#) displays the use of the command in SQL Developer, along with the result. You need to include the file extension only if it is something other than .sql.



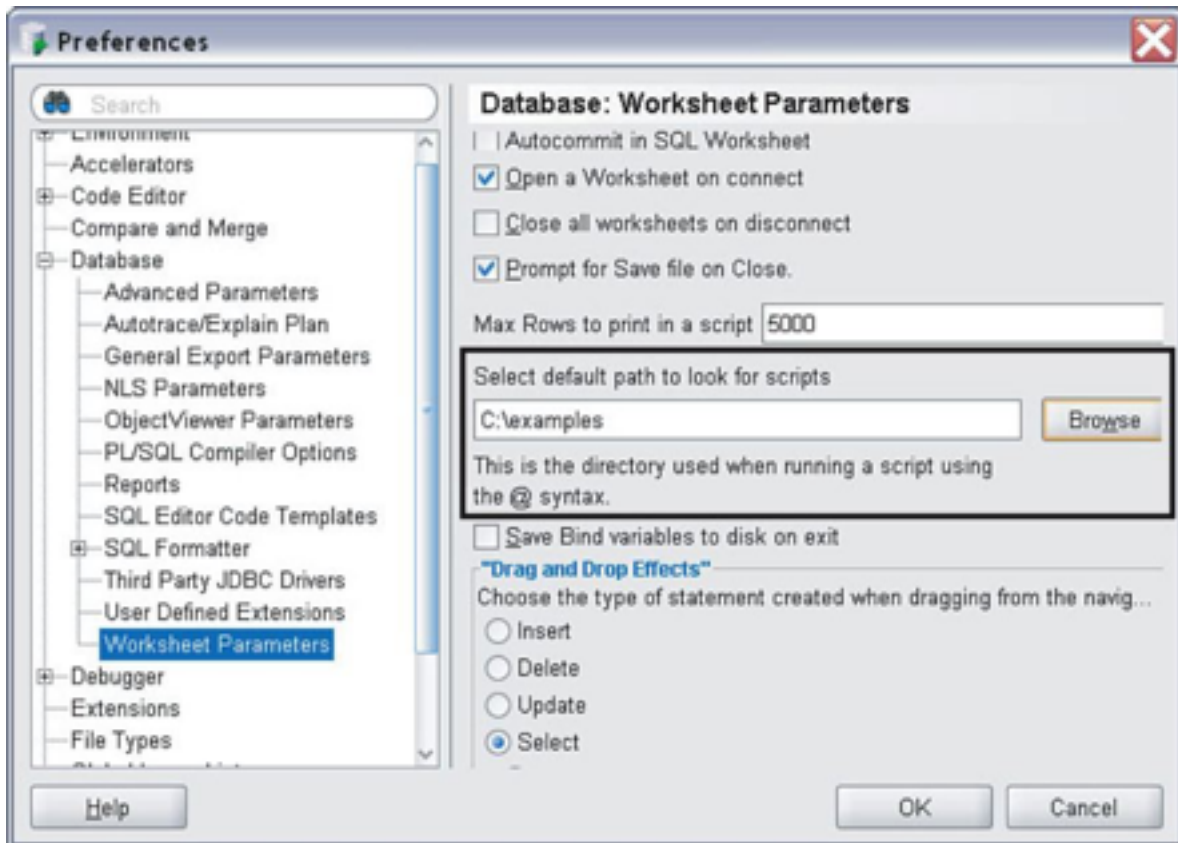
The Run Script functionality in SQL Developer executes many of the SQL*Plus commands. If a command is not supported, SQL Developer ignores the command and displays a warning message. For a list of supported SQL*Plus commands for use in SQL Developer, refer to Appendix C, “SQL*Plus Command Reference.”

FIGURE 14.4 Using the SQL*Plus @ or START command in SQL Developer



If you don't want to enter a directory path, you can set up a default directory in SQL Developer (see [Figure 14.5](#)). Then you can run the script as @my_script, and SQL Developer will find the file in the default directory.

FIGURE 14.5 Default path for scripts in SQL Developer



RUNNING MULTIPLE SQL STATEMENTS

The Run Script icon is useful not only for scripts that are saved in files but also for multiple statements listed in the SQL Worksheet. When you click the Execute Statement icon (or press F9), only the statement on which the cursor resides or the highlighted text is executed. If you run the statements with the Run Scripts

icon (or press F5), all statements in the SQL Worksheet are executed.

The result of the Run Script icon appears in the Script Output tab, which is not graphical but emulates SQL*Plus as much as possible.

Running a Script in SQL*Plus

You can run a script from the SQL*Plus prompt as follows.

```
SQL>@c:\examples\myfile
```

Alternatively, you can evoke the script from the operating system prompt, as shown next.

```
sqlplus student/learn @c:\examples  
\myfile
```

The following is the content of the myfile.sql script. It simply counts the number of courses. The file contains an EXIT command that causes SQL*Plus to return to the operating system after the script finishes executing. (In SQL Developer, the EXIT command does not end the program.)

```
SELECT COUNT (*)  
FROM course;  
EXIT
```

633

634



If your scripts contain any statements that manipulate data, keep in mind that the EXIT command automatically issues an implicit COMMIT in both SQL*Plus and SQL Developer.

If you need to run the same script or a series of scripts repeatedly, you might want to consider writing a batch file. Following is an example of a Windows batch file called `daily_batch.bat`.

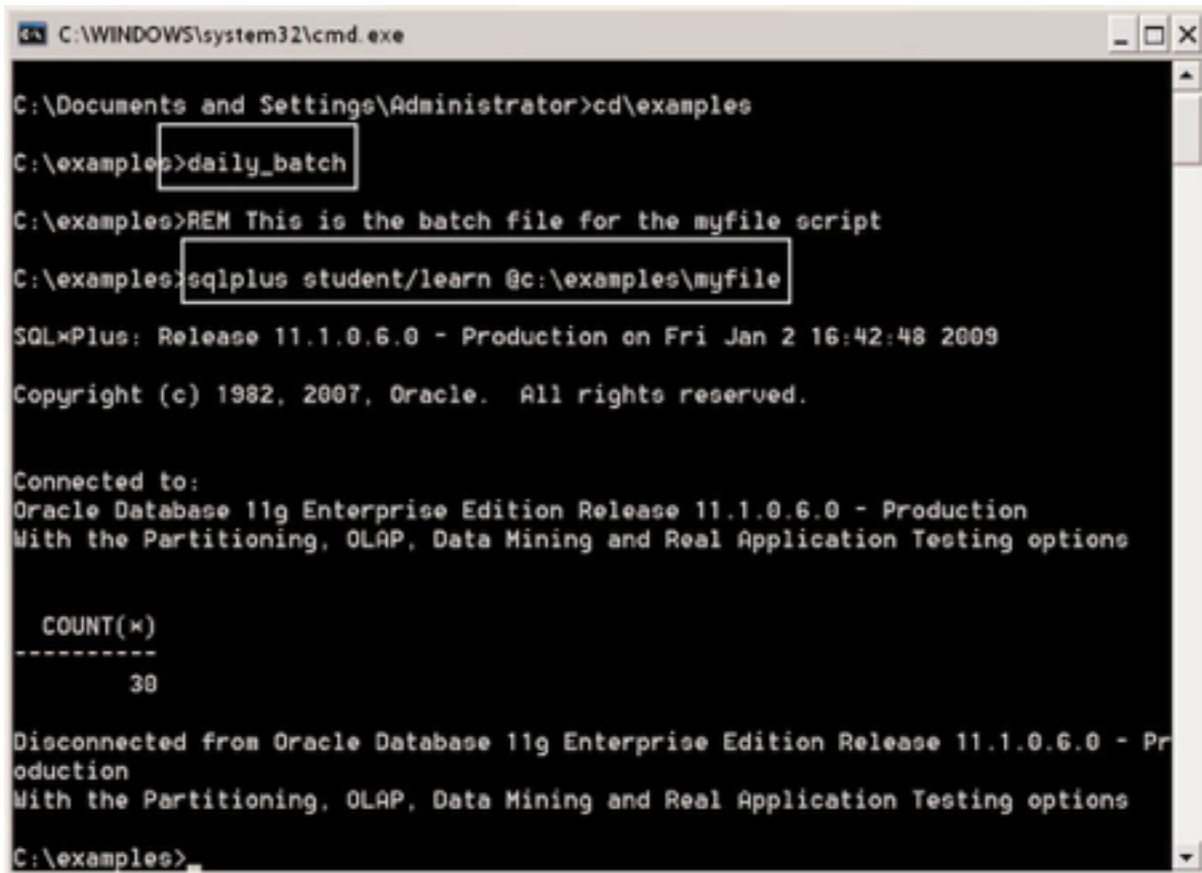
```
REM This is the batch file for the  
myfile script  
sqlplus student/learn @c:\examples  
\myfile
```

The REM command in the first line is a remark or comment statement. In the second line, SQL*Plus is invoked together with the `myfile.sql` script. The full path for the script file (`c:\examples\`) is optional here because you are already executing this in the same directory where the `myfile.sql` file is located.

[Figure 14.6](#) shows the execution of the `daily_batch.bat` file from the Windows command prompt. (To see the

Windows command prompt, choose the Windows Start menu, Run and then enter CMD.)

FIGURE 14.6 Run of a batch file invoking SQL*Plus



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>cd\examples
C:\examples>daily_batch
C:\examples>REM This is the batch file for the myfile script
C:\examples>sqlplus student/learn @c:\examples\myfile

SQL*Plus: Release 11.1.0.6.0 - Production on Fri Jan 2 16:42:48 2009
Copyright (c) 1982, 2007, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

  COUNT(*)
  -----
         30

Disconnected from Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Pr
oduction
With the Partitioning, OLAP, Data Mining and Real Application Testing options
C:\examples>
```

634

635

You can substitute the username, password, and connect string with operating system variables for additional flexibility. Consult your operating system manual regarding the creation of operating system variables.

Substitution Variables

If you find yourself executing a similar SQL statement over and over again, you can write your statement using substitution variables. In this case, part of the SQL statement is replaced with a variable. When the statement is executed in either SQL Developer or SQL*Plus, you supply the appropriate value for the variable. The substitution variable is prefixed with an arbitrary variable name and an ampersand (&) symbol. When you execute the statement, SQL Developer or SQL*Plus prompts you for a value, and the supplied value is assigned to the variable.

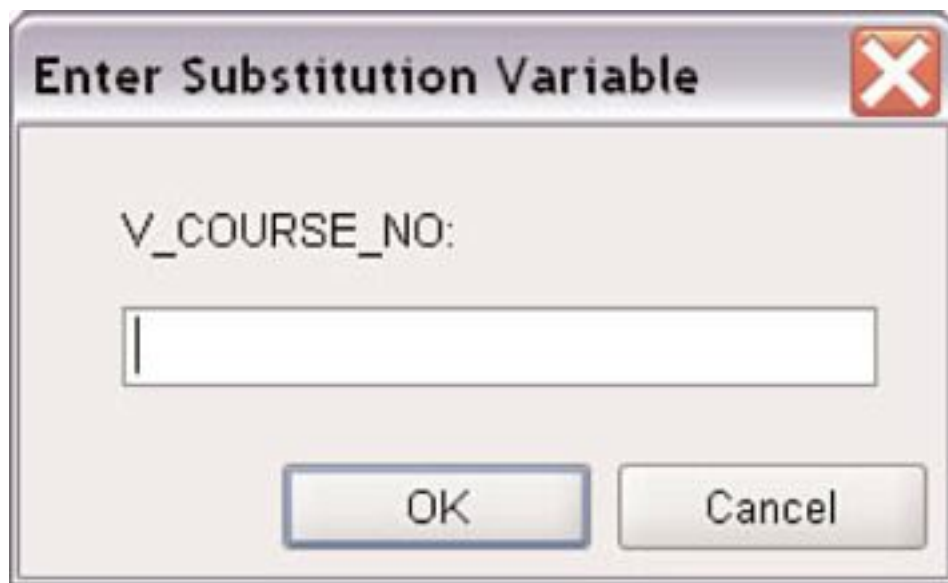
For example, the variable in the following statement is named `v_course_no`.

```
SELECT course_no, description
FROM course
WHERE course_no = &v_course_no
```

SUBSTITUTION VARIABLES IN SQL DEVELOPER

The prompt you see in SQL Developer looks similar to [Figure 14.7](#).

FIGURE 14.7 Substitution variable prompt in SQL Developer



If you enter the value 240 and click OK, this value will be assigned to the variable `v_course_no` during execution.

SUBSTITUTION VARIABLES IN SQL*PLUS

If you use the command-line version of SQL*Plus, your prompt looks as follows.

```
SQL> SELECT course_no, Description
2 FROM course
3 WHERE course_no = &v_course_no
4 /
Enter value for v_course_no: 240
old 3: WHERE course_no = &v_course_no
new 3: WHERE course_no = 240
COURSE_NO DESCRIPTION
-----
240 Java to the BASIC Language
1 row selected.
```

The text displayed after the substitution variable prompt shows the value before (old) and after the substitution of the value (new). The number 3 indicates that the substitution variable is found on line 3 of the SQL statement. You can change this default behavior with the SET VERIFY OFF SQL*Plus command, and SQL*Plus will no longer display the old and new values.

If you want to re-execute the statement in the buffer, use the forward slash (/), and you are prompted for a value for the v_course_no substitution variable each time.

USING SUBSTITUTION VARIABLES

You can use a substitution variable in any SQL statement executed within the SQL Developer and SQL*Plus environments. Substitution variables are not limited to the WHERE clause of a statement. You can also use them in the ORDER BY clause or the FROM clause to substitute a table name, you can use them in individual column expressions, or you can even substitute an entire WHERE clause.

SUPPRESSING THE USE OF SUBSTITUTION VARIABLES

There are times when you want the ampersand to be a literal ampersand rather than an indicator that a substitution variable follows. The following example illustrates this scenario.

```
UPDATE student
  SET employer = 'Soehner & Peter'
  WHERE student_id = 135
Enter value for peter:
```

SQL*Plus and SQL Developer think you want to use a substitution parameter rather than the literal ampersand. To remedy this, use the SET DEFINE command to turn the use of substitution parameters on or off.

```
SET DEFINE OFF
UPDATE student
  SET employer = 'Soehner & Peter'
  WHERE student_id = 135
1 row updated.
SET DEFINE ON
```

Issue a ROLLBACK command to undo the change of employer and set it back to the original value.

ROLLBACK

Rollback complete.

636

637

Running a Script with a Parameter in SQL*Plus

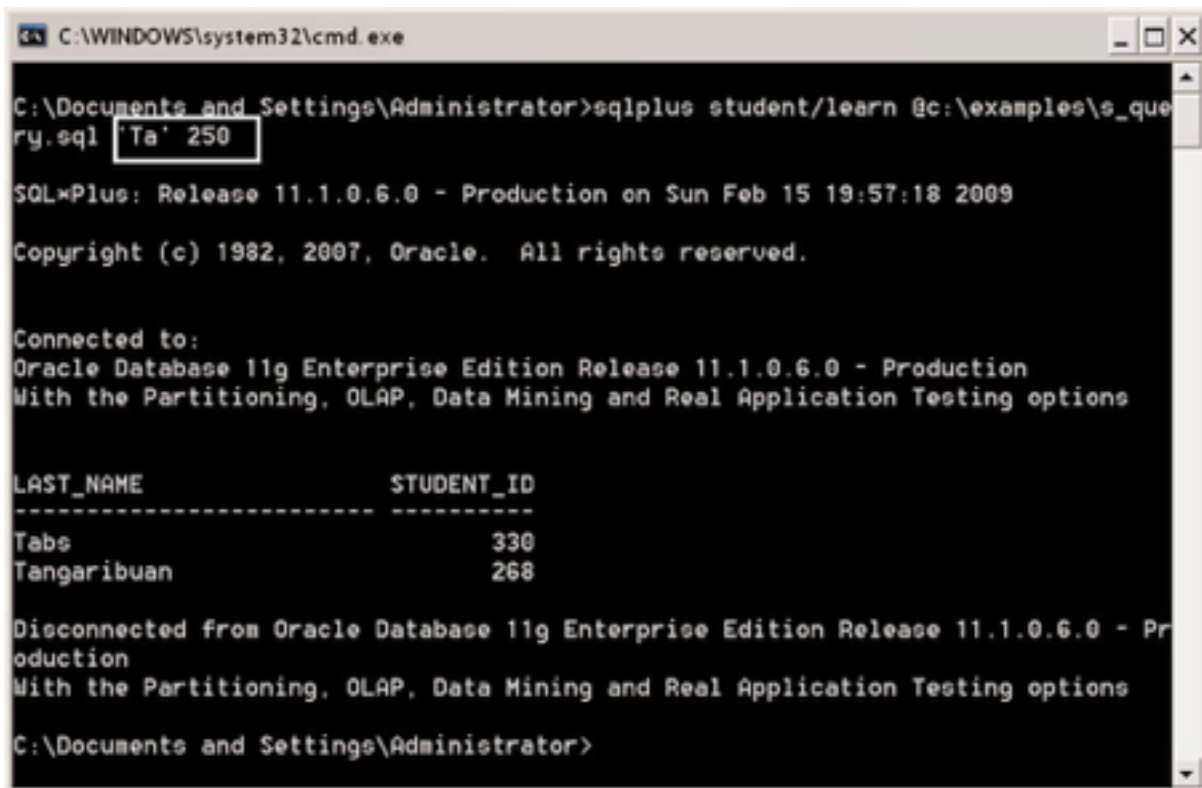
The s_query.sql file contains the following SQL statement. Like all other SQL statements in scripts, every statement must end with a semicolon or a forward slash on a separate line. Only SQL*Plus commands do not require a semicolon.

```
SET VERIFY OFF
SELECT last_name, student_id
FROM student
WHERE last_name like '&1' || '%'
AND student_id > &2;
EXIT
```

You can pass parameters (arguments) from the operating system when running a script file in SQL*Plus. This works only if your substitution variable is a numeral from 1 through 9. The &1 parameter is substituted with the first parameter passed. If you include another parameter, such as &2, you can pass a second argument, and so on.

This functionality is most useful when executed from an operating system command prompt, as shown in [Figure 14.8](#). Here the first parameter consists of 'Ta' followed by the number 250. Therefore, any students with a last name that begins with the letters Ta and a STUDENT_ID greater than 250 are displayed.

FIGURE 14.8 Running a SQL*Plus script with script arguments



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>sqlplus student/learn @c:\examples\s_query.sql 'Ta' 250

SQL*Plus: Release 11.1.0.6.0 - Production on Sun Feb 15 19:57:18 2009

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

  LAST_NAME          STUDENT_ID
  -----
  Tabs               330
  Tangaribuan        268

Disconnected from Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

C:\Documents and Settings\Administrator>
```

6.37

Common SQL*Plus Commands

The SQL*Plus commands within a script change the settings of the SQL*Plus environment. Following are some of the commonly used SQL*Plus commands. Many of these SQL*Plus commands are listed in Appendix C, and you will see them used in the lab exercises.

SPOOL

The SPOOL command, together with a file name, spools any subsequently issued SQL*Plus or SQL command to a file. The following example creates a file named temp.lst. If a file with the same name already exists, it is overwritten without warning, unless you use the CREATE syntax option. You can also optionally append a file with the APPEND syntax option.

```
SPOOL temp.lst CREATE
```

To show the name of the file you are currently spooling to, use the SPOOL command.

```
SPOOL  
currently spooling to temp.lst
```

To end the spooling and close the file, enter the following command:

SPOOL OFF

Just as with other file names, you can add a path to store the file in a directory other than your SQL*Plus default directory.

PAGESIZE

The SQL*Plus default value for PAGESIZE is 14; when you use the default, you repeat the heading every 14 lines. The PAGESIZE 0 command suppresses the column headings. If you want just the first row to have a column heading, set it to a large number, such as 50,000, the largest possible setting.

FEEDBACK

The FEEDBACK command returns the number of records returned by a query. If you are spooling to a file, you may not want to see this value. To suppress the feedback, issue either the command SET FEEDBACK 0 or SET FEEDBACK OFF.

TERMOUT

The SET TERMOUT OFF or SET TERM OFF command controls the display of output generated by the commands. The OFF setting suppresses the output

from the screen only when the command is executed from a script file.

638

639

LINESIZE

The SET LINESIZE command determines the total number of characters SQL*Plus displays in one line before beginning a new line. Setting it to 80 makes it easy to read the spooled output in a text editor.

SHOW ALL

To see the current settings of all SQL*Plus settings, use the SHOW ALL command.

Generating HTML Reports in SQL*Plus

You can generate reports in SQL*Plus by using various SQL*Plus formatting commands. These reports are formatted as plain text and fixed-character-length format. For a more formatted look, SQL*Plus allows you to generate HTML pages. The file called c:\examples\s_lname_rep.sql contains the following statements.

```
SET ECHO OFF
SET TERM OFF
SET FEEDBACK OFF
SET PAGESIZE 50000
```

```
SPOOL C:\examples
\s_lname_result.HTML
SET MARKUP HTML ON SPOOL ON
SELECT last_name, first_name,
student_id
FROM student
WHERE last_name like 'R%'
ORDER BY 1, 2;
SPOOL OFF
EXIT
```

The various SQL*Plus commands at the beginning of the script define the SQL*Plus environment settings. The SET ECHO OFF command prevents the display of each command in the script. The SET TERM OFF command prevents the display of output generated by the commands. The FEEDBACK OFF command avoids returning the number of records returned by a query.

The PAGESIZE command shows only one header at the beginning of the output. The script spools the result into the file s_lname_result.HTML in the c:\examples directory. The SET MARKUP HTML ON SPOOL ON command creates the tags for HTML format. The SQL statement shows students with a last name starting with the letter R. The SPOOL OFF command closes the file,

and the EXIT command exits SQL*Plus and returns you to the operating system command prompt.

639

The result of this script produces an HTML file that looks similar to [Figure 14.9](#) when opened in a Web browser.

640

FIGURE 14.9 HTML output from a SQL*Plus report

LAST_NAME	FIRST_NAME	STUDENT_ID
Radicola	Pierre	123
Ramesh	Simon	179
Reed	James	133
Reyes	Deborah	189
Roberts	Bharat	243
Robichaud	Barbara	212
Robles	Brian	290
Rosenberg	Rawan	361
Ross	George	275
Rothstein	Adele	265
Runyan	Jeff	128

Generating Dynamic SQL

Dynamic SQL allows you to build SQL commands at runtime. Dynamic SQL is often executed in Oracle's PL/SQL, but it can also be generated and executed in SQL*Plus, using SQL*Plus scripts. These scripts are often referred to as *SQL to generate SQL scripts* or *master/slave scripts*. By using SQL*Plus, you can automatically generate SQL statements and spool them to a file for use.

For example, say that you made some database changes to tables, causing other database objects, such as views, to become invalid. To compile the views, you can repeatedly type the ALTER VIEW command for each invalid view, or you can wait for the user to access the views and let Oracle compile them. However, it is best to compile them after the table changes to make sure there are no errors. You can do this by writing a script to generate the ALTER VIEW statement for each invalid view. The following SQL statement generates the required SQL.

```
SELECT 'ALTER VIEW ' || object_name
|| ' COMPILE;'
FROM user_objects
WHERE object_type = 'VIEW'
```

```
AND status <> 'VALID'
```

If you have any invalid views, your result might look as follows.

```
'ALTERVIEW' || OBJECT_NAME || 'COMPILE;'
-----
ALTER VIEW CAPACITY_V COMPILE;
ALTER VIEW CT_STUDENT_V COMPILE;
ALTER VIEW NJ_STUDENT_V COMPILE;
ALTER VIEW NY_STUDENT_V COMPILE;

4 rows selected.
```

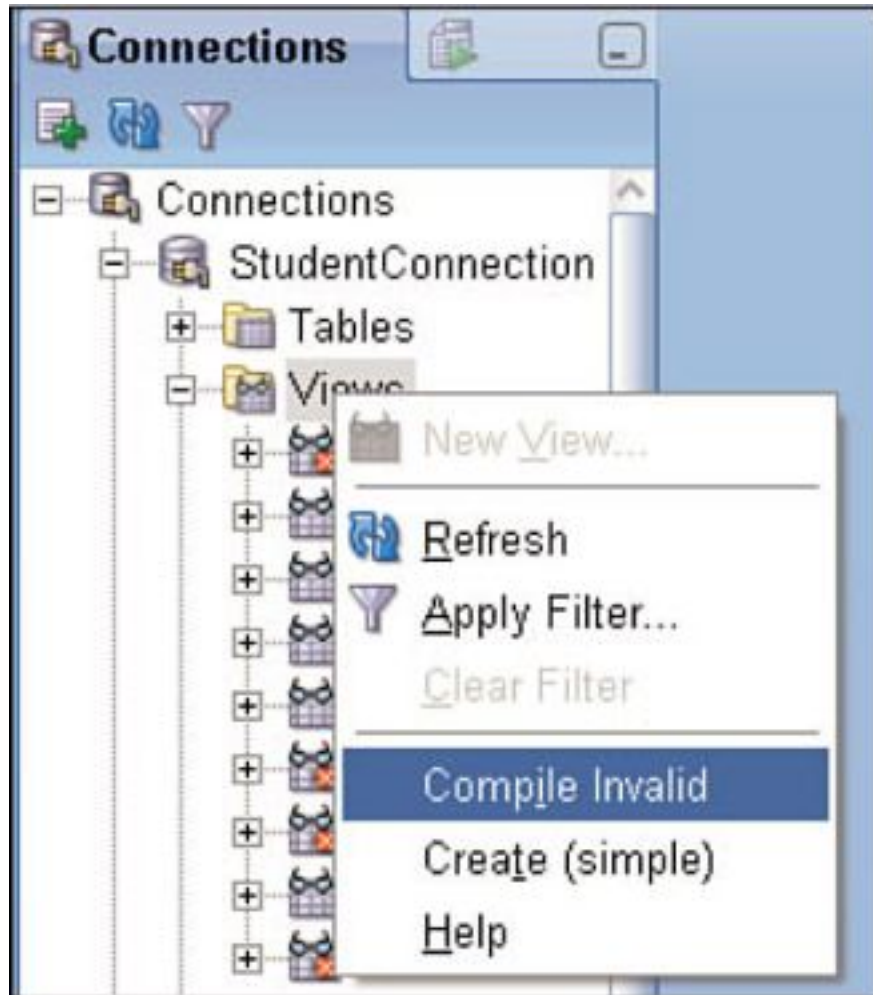
640

641

The text literal 'ALTER VIEW' is concatenated with the view name and then with the text literal 'COMPILE;'. You can save the result to a file by using the SPOOL command and then execute it to compile all the invalid views.

One of SQL Developer's useful features is that it contains a few scripts to accomplish some of these tasks for you. In the example of the invalid views, you can use the Compile Invalid menu option (see [Figure 14.10](#)), which you see when you right-click the Views node.

FIGURE 14.10 The Compile Invalid menu item



The resulting screen shows a small PL/SQL program that loops through the ALL_OBJECTS data dictionary view to find all the invalid objects for the current user account and the currently selected object type. It then executes for each row the required command to compile the object. To learn more about PL/SQL, see *Oracle PL/SQL*

by Example, 4th Edition, by Benjamin Rosenzweig and Elena Silvestrova Rakhimov (Prentice Hall 2008).

While SQL Developer includes functionality for some of these commonly performed tasks, there will be instances when you need to write your own script to solve a particular problem or automate a task. Being able to write a dynamic script will enable you to solve a given problem in an efficient and repeatable manner.

Following is a script that shows how to disable constraints using a dynamic SQL script. There are times when you want to disable constraints temporarily, such as when you must bulk load data or update large quantities of data quickly. Afterward, you enable the constraints again. For example, the following SQL statement disables the foreign key constraint on the ZIP column of the STUDENT table.

```
ALTER TABLE student DISABLE  
CONSTRAINT stu_zip_fk
```

Disabling the constraint allows child values to be entered where no corresponding parent exists. This means, for example, that you can insert or update in the STUDENT table a zip code that does not have a corresponding value in the ZIPCODE table. To perform this task for all

641

642

constraints in your schema, you start by writing the following statement.

```
SELECT 'ALTER TABLE ' || table_name
      FROM user_constraints
      WHERE constraint_type = 'R'
```

The statement generates a list of all the tables with foreign key constraints, together with the literal 'ALTER TABLE'. There are multiple rows with the same table name because a table may have multiple foreign keys.

```
'ALERTABLE' || TABLE_NAME
-----
ALTER TABLE COURSE
...
ALTER TABLE SECTION
ALTER TABLE SECTION
ALTER TABLE STUDENT

11 rows selected.
```

The following SQL statement expands the previous statement. You add the **DISABLE** clause to the statement by concatenating the text literal 'DISABLE CONSTRAINT' with the constraint name and then with another text literal containing the semicolon. The resulting output looks as follows.

```
SELECT 'ALTER TABLE ' || table_name ||  
       ' DISABLE CONSTRAINT ' || constraint_name || ';'   
FROM user_constraints  
WHERE constraint_type = 'R'  
'ALTER TABLE ' || TABLE_NAME || ' DISABLE CONSTRAINT ' || CONSTRAINT  
-----  
ALTER TABLE COURSE DISABLE CONSTRAINT CRSE_CRSE_FK;  
...  
ALTER TABLE SECTION DISABLE CONSTRAINT SECT_CRSE_FK;  
ALTER TABLE SECTION DISABLE CONSTRAINT SECT_INST_FK;  
ALTER TABLE STUDENT DISABLE CONSTRAINT STU_ZIP_FK;  
  
11 rows selected.
```

Now the syntax of the SQL statement works perfectly. You can save the SQL statement in a file named `disable_fk.sql`. Add the following SQL*Plus statements at the beginning of the file. (The double dashes indicate single-line comments.)

```
-- File Name: disable_fk.sql  
-- Purpose: Disable Foreign Key  
constraints.  
-- Created Date: Place current date  
here  
-- Author: Put your name here
```

```
SET PAGESIZE 0  
SET LINESIZE 80  
SET FEEDBACK OFF  
SET TERM OFF
```

642

643

```
SPOOL disable_fk.out
```

Add a semicolon at the end of the SQL statement and the following SQL*Plus commands afterward.

```
SPOOL OFF
SET PAGESIZE 20
SET LINESIZE 100
SET FEEDBACK ON
SET TERM ON
```

After editing the file, the disable_fk.sql script should look similar to the following. This script includes the CHR function with a value of 10, which automatically returns a new line in the result.

```
-- File Name: disable_fk.sql
-- Purpose: Disable Foreign Key
constraints.
-- Created Date: Place current date
here
-- Author: Put your name here
SET PAGESIZE 0
SET LINESIZE 80
SET FEEDBACK OFF
SET TERM OFF
SPOOL disable_fk_result.sql CREATE
SELECT 'ALTER TABLE ' || table_name
|| CHR(10) ||
```

```
' DISABLE CONSTRAINT ' ||  
constraint_name || ';'   
FROM user_constraints  
WHERE constraint_type = 'R';  
SPOOL OFF  
SET PAGESIZE 20  
SET LINESIZE 100  
SET FEEDBACK ON  
SET TERM ON
```

Executing the script `disable_fk.sql` with the `@` command results in the `disable_fk_result.sql` file, which looks like the following.

```
ALTER TABLE COURSE  
    DISABLE CONSTRAINT CRSE_CRSE_FK;  
  
...  
  
ALTER TABLE STUDENT  
    DISABLE CONSTRAINT STU_ZIP_FK;
```

The spooled file contains a list of all SQL statements necessary to disable all the foreign constraints for the user. You can execute the commands in the file by typing `@disable_fk_result.sql` at the SQL*Plus prompt.

643

644



SQL Developer does not support the SPOOL command (see Appendix C for a list of supported commands). Use SQL*Plus in the lab exercises involving the generation of dynamic SQL statements and the SPOOL command.

You should document your scripts by using comments.

Begin a single-line comment with two hyphens (--).

Begin a multiline comment with a slash and an asterisk (/*) and end it with an asterisk and a slash (*). In SQL*Plus scripts, you can also use the REMARK (REM) command to indicate a comment.

```
/* This is a multi-line
comment */
-- A single-line comment, it ends
with a line break.
REM Another single-line comment,
only used in SQL*Plus.
```

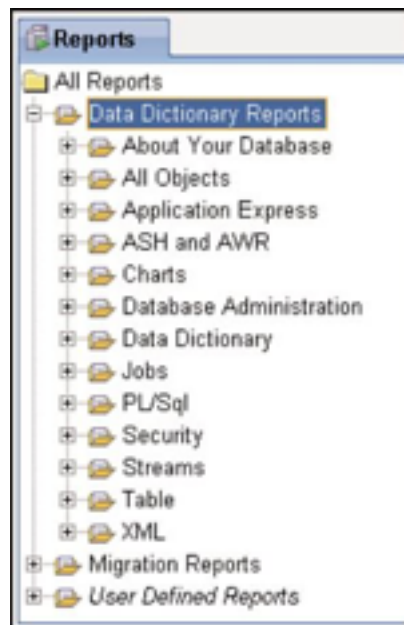
SQL Developer's Data Dictionary Reports

SQL Developer offers a number of helpful, predefined reports you can run against the data dictionary. The supplied reports cover a range of topics and illustrate a number of different presentation styles. You will find simple reports such as a list of all objects, complex

reports detailing foreign keys without indexes, and graphical reports listing the occurrences of various column types.

[Figure 14.11](#) shows the Reports navigator pane. If this tab is not visible on the left of the window, choose View, Reports. The folders in the Reports navigator pane show three sets of folders: The All Reports folder, which contains a list of Oracle-supplied reports; the Migration Reports folder, which is related to migrating third-party database to Oracle; and the User Defined Reports folder, which is where you create your own reports.

FIGURE 14.11 SQL Developer's Reports navigator pane



[Table 14.3](#) provides a brief description of the different standard reports in the All Reports folder supplied within SQL Developer. The best reports to review are the reports listed in the All Objects, Data Dictionary, Database Administration, and Table folders.

TABLE 14.3 Overview of the SQL Developer All Reports Folder

REPORT FOLDER	DESCRIPTION
About Your	Lists release information and National Language Support (NLS) parameters. Database
All Objects	Lists all objects accessible for the chosen database connection. Also lists a helpful Dependencies report for assessing change impact.
Application Express	Contains reports useful for developers of Oracle's Application Express software.
ASH and AWR	Lists information on Active Session History (ASH) and Automated Workload Repository (AWR); useful for DBAs and installations where this Oracle add-on option is licensed.
Charts	Provides a graphical report showing distribution of objects.
Database Administration	Lists many reports intended for DBAs and developers, such as information on tables, constraints, indexes, users, and sessions.
Data Dictionary	Lists the data dictionary views and data dictionary columns.

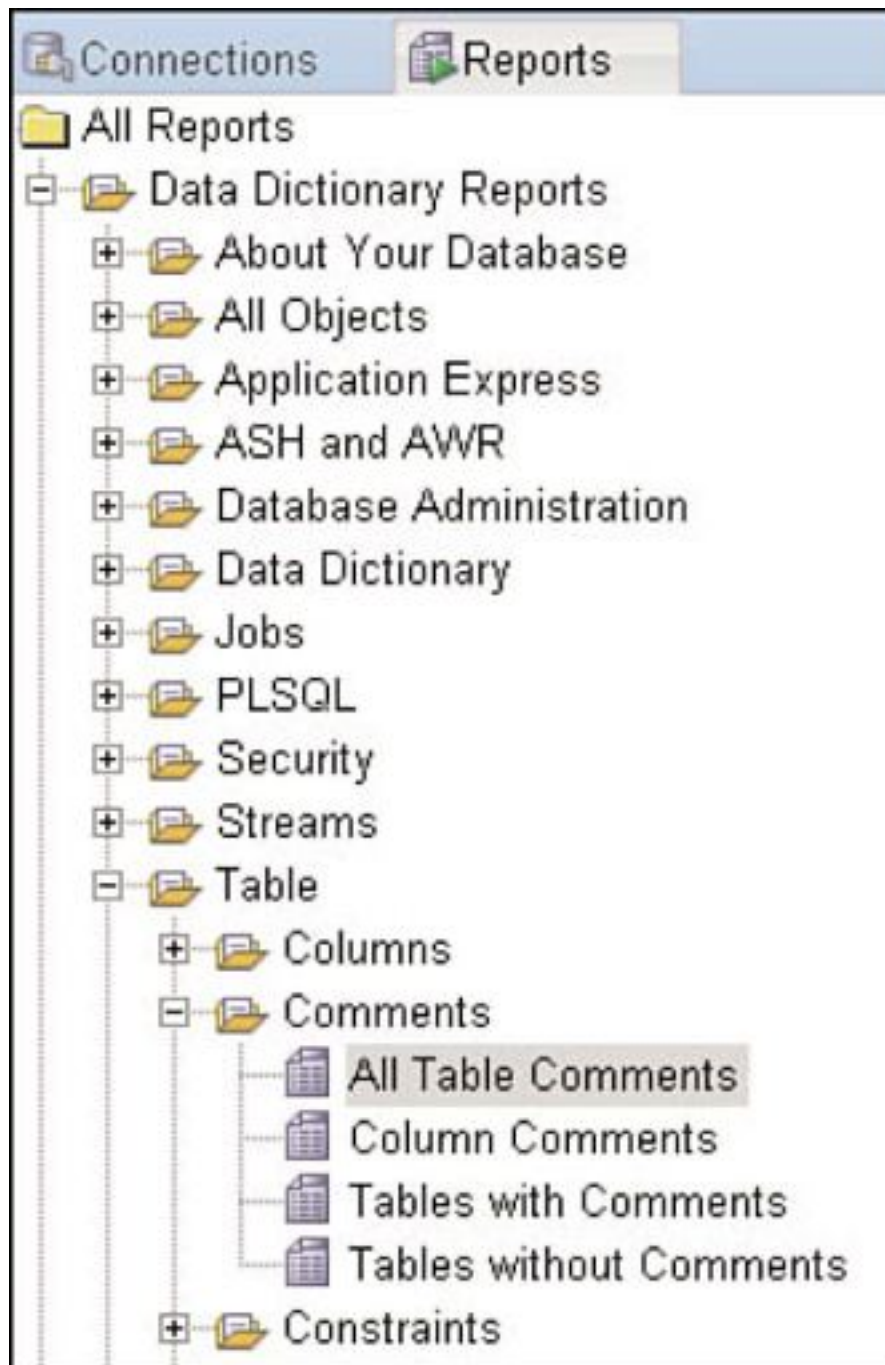
Jobs	Lists jobs running in the database; helpful for DBAs.
PL/SQL	Lists PL/SQL objects and searches the PL/SQL source code of these objects.
Security	Displays information related to grants and privileges within the database.
Streams	Lists reports related to Oracle Steams, which manages sharing of data and events in a distributed environment.
Table	Lists information related to tables, such as columns, indexes, and constraints; contains quality assurance reports to indicate any possibly missing indexes or constraints.
XML	Lists information regarding Oracle XML objects; helpful for XML developers.

Running SQL Developer Reports

When you click on one of the reports, you need to choose the connection against which you would like to run the report. For example, if you open the Table folder, you see another subfolder related to comments. This folder contains various reports displaying information about table and column comments (see [Figure 14.12](#)).

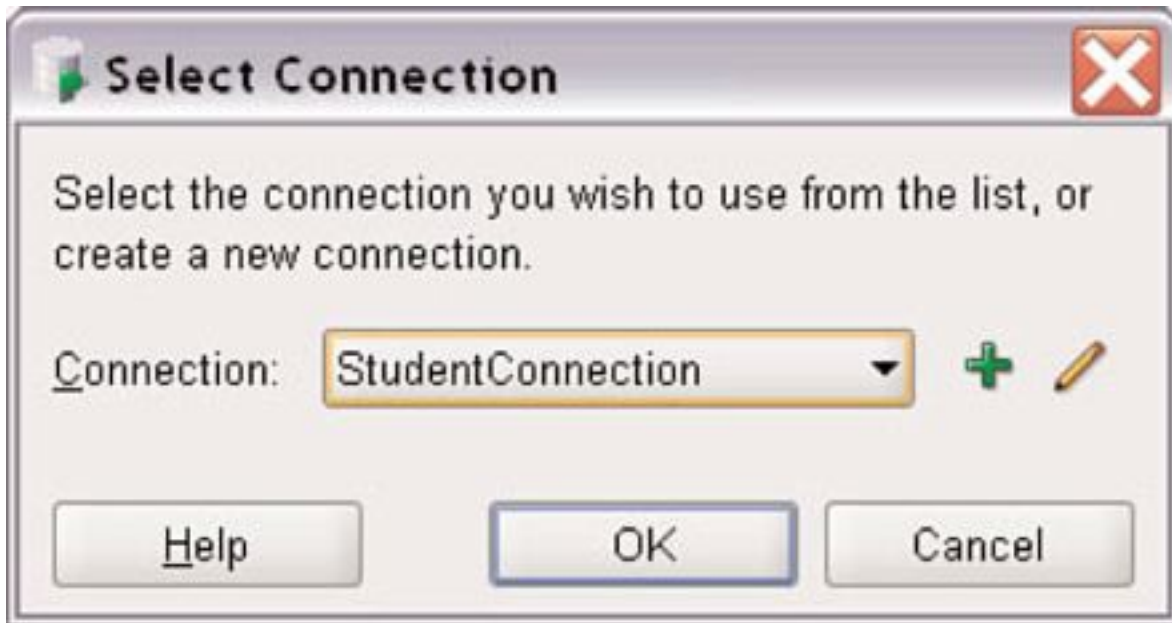
645

FIGURE 14.12 Various reports related to table and column comments



To run the report, double-click it. You are prompted for the connection (see [Figure 14.13](#)).

FIGURE 14.13 Select Connection box



Some of the reports have parameters, called bind variables, where you can enter values before the report is executed. These parameters narrow down the result. The All Table Comments report prompts you on a specific table name for which you would like to see the data; this report contains one bind value parameter (see [Figure 14.14](#)).

646

FIGURE 14.14 Bind Value dialog box

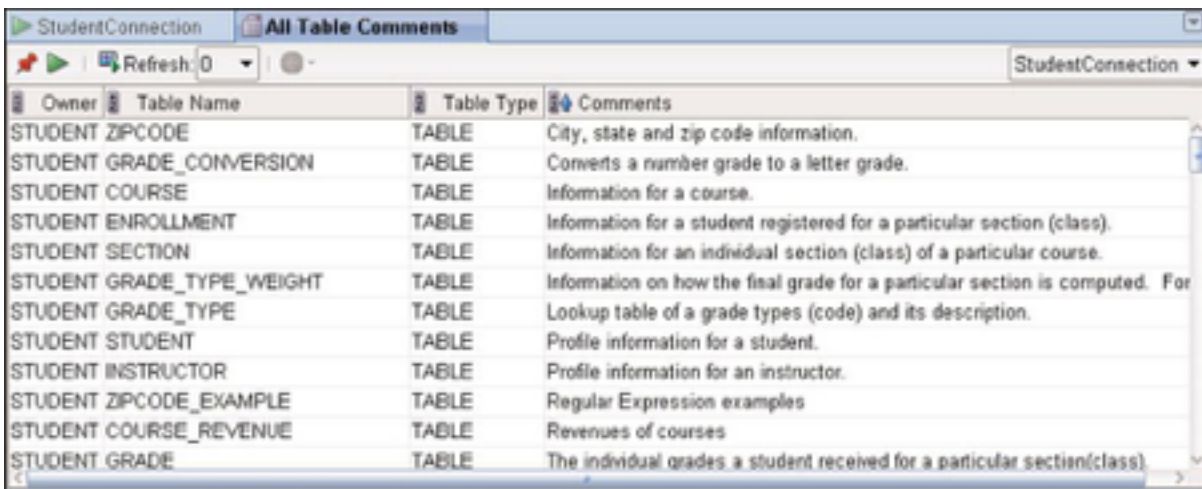


If you do not want to restrict the report for specific values, you can run the report without any bind variables by simply clicking the Apply button, which in this example will include all the tables, not just a specific table. The default value for bind variables is NULL, which indicates in SQL Developer that there is no restriction.

Bind variables are similar to the substitution variables discussed earlier. Instead of an ampersand, you see a colon before a bind variable's name.

[Figure 14.15](#) displays an excerpt of the All Table Comments report.

FIGURE 14.15 SQL Developer's All Table Comments report



Owner	Table Name	Table Type	Comments
STUDENT	ZPCODE	TABLE	City, state and zip code information.
STUDENT	GRADE_CONVERSION	TABLE	Converts a number grade to a letter grade.
STUDENT	COURSE	TABLE	Information for a course.
STUDENT	ENROLLMENT	TABLE	Information for a student registered for a particular section (class).
STUDENT	SECTION	TABLE	Information for an individual section (class) of a particular course.
STUDENT	GRADE_TYPE_WEIGHT	TABLE	Information on how the final grade for a particular section is computed. For
STUDENT	GRADE_TYPE	TABLE	Lookup table of a grade types (code) and its description.
STUDENT	STUDENT	TABLE	Profile information for a student.
STUDENT	INSTRUCTOR	TABLE	Profile information for an instructor.
STUDENT	ZPCODE_EXAMPLE	TABLE	Regular Expression examples
STUDENT	COURSE_REVENUE	TABLE	Revenues of courses
STUDENT	GRADE	TABLE	The individual grades a student received for a particular section(class).

The resulting report contains icons similar to the Results pane in the SQL Worksheet. The Push Pin icon leaves the information in place instead of replacing it with the next report. The triangle allows you to rerun the report.

647

648

Next to the triangle is the SQL Worksheet icon. You can click on this icon to copy the report's underlying SQL statement into the SQL Worksheet box; this allows you to examine the report's SQL query. The Refresh drop-down choice enables you to rerun the report at specific

intervals; this may be useful if your data changes frequently.

The Connection box to the right displays the StudentConnection, but you can choose another connection from the drop-down menu and run the report from a different user account. Depending on the underlying SQL statement, the results may vary from user account to user account because not every user may have the same access privileges and data.

The underlying SQL statement of the report shows that the report uses the SYS schema name. The SYS owner is the most privileged user in the Oracle database and the owner of the data dictionary. Many of the data dictionary views shown earlier are public synonyms or public views of these SYS owner dictionary tables. You will learn more about synonyms and the SYS user in [Chapter 15](#).

Creating User-Defined SQL Developer Reports

In the folder User Defined Reports, you can create your own reports and folders. Any new report you create will be added in this folder. When you right-click the User Defined Reports folder, the menu options shows the Add Report choice.

[Figure 14.16](#) displays the Create Report dialog box, which lists the report name and report style. This report is of Table style, which displays the data in columns and rows. This is the default report output style. You can fill in optional Description and Tooltip information. In the SQL box, you enter a query. This example shows a query against the USER_OBJECTS table that lists objects in the user's own schema. Click the Apply button to save the report.

[Figure 14.17](#) displays the report result, ordered alphabetically by the object name. All the objects are owned by the current user. The result will vary from user to user. You will most likely see a list of tables, indexes, and sequences, but the list may include views, procedures, packages, functions, synonyms, triggers, and other object types.



When writing reports, you might want to use column aliases to improve the appearance and readability of the report. In the User Objects Report shown in [Figure 14.17](#), two aliases were created: one for the formatted

CREATED column and the other for the
LAST_DDL_TIME column.

648

649

FIGURE 14.16 The Create Report Dialog screen

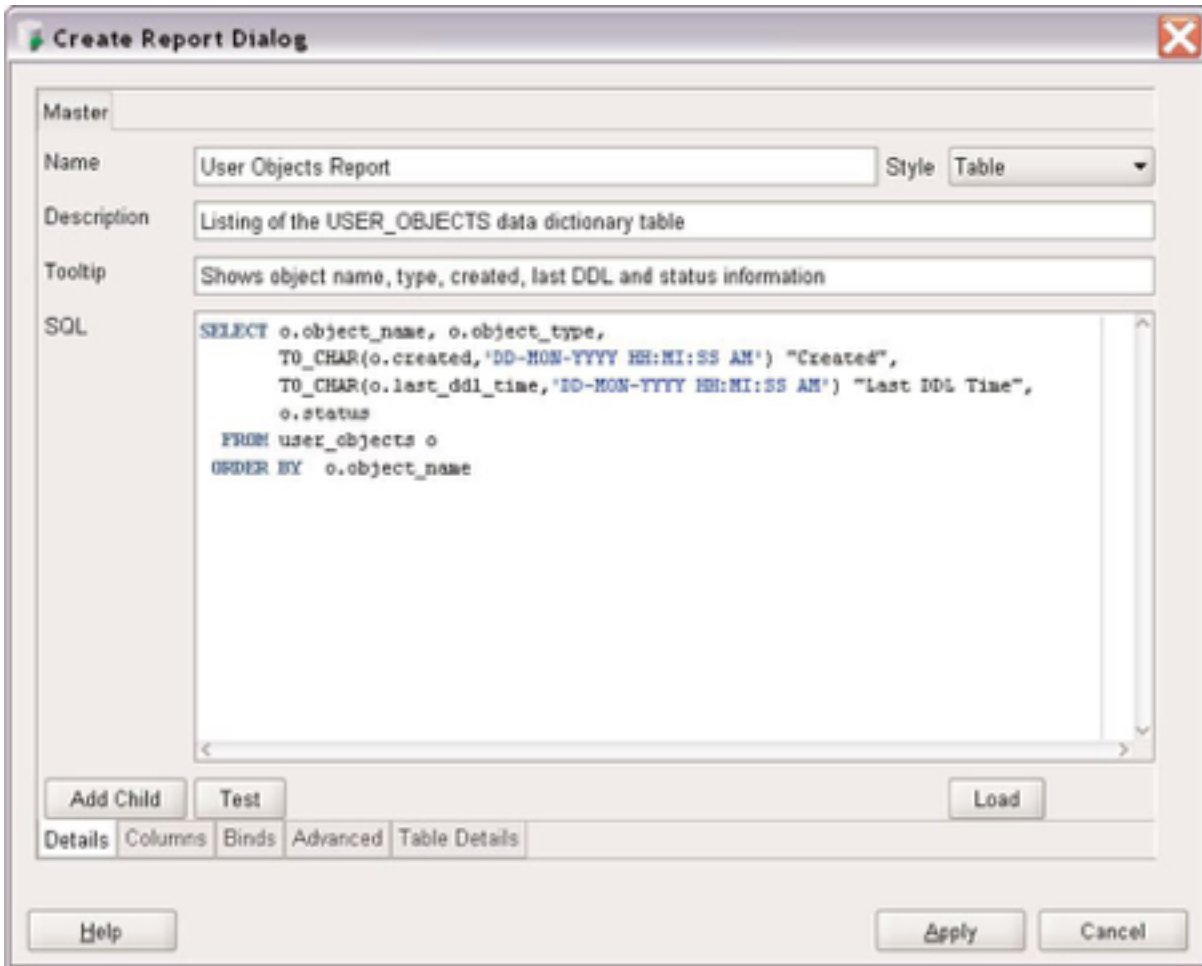


FIGURE 14.17 The report result of the user-defined User Objects report

Object Name	Object Type	Created	Last DDL Time	Status
COURSE_REVENUE	TABLE	26-OCT-2008 06:47:14 PM	26-OCT-2008 06:47:14 PM	VALID
CRSE_CRSE_FK_I	INDEX	21-DEC-2008 04:31:48 PM	21-DEC-2008 04:31:48 PM	VALID
CRSE_PK	INDEX	21-DEC-2008 04:32:16 PM	21-DEC-2008 04:32:16 PM	VALID
DATE_EXAMPLE	TABLE	26-OCT-2008 06:47:13 PM	26-OCT-2008 06:47:13 PM	VALID
DOCS	TABLE	09-NOV-2008 12:22:05 PM	09-NOV-2008 12:22:05 PM	VALID
EMPLOYEE	TABLE	26-OCT-2008 06:47:14 PM	26-OCT-2008 06:47:14 PM	VALID
EMPLOYEE_CHANGE	TABLE	26-OCT-2008 06:47:14 PM	26-OCT-2008 06:47:14 PM	VALID
ENROLLMENT	TABLE	21-DEC-2008 04:31:47 PM	21-DEC-2008 04:32:16 PM	VALID
ENR_PK	INDEX	21-DEC-2008 04:32:16 PM	21-DEC-2008 04:32:16 PM	VALID
ENR_SECT_FK_I	INDEX	21-DEC-2008 04:31:48 PM	21-DEC-2008 04:31:48 PM	VALID

Another way to create a new report is by copying and pasting an existing one into the User Defined Reports folder. You can also export the report in XML format and import it into other folders or distribute it to another user.

649

650

LAB 14.2 EXERCISES

- a) Enter all the following commands in a file named minmaxval.sql and then execute the script in SQL*Plus or SQL Developer. For the column name supply the value CAPACITY and for the table name enter the value SECTION. What do you observe in either SQL Developer or SQL*Plus?

```
ACCEPT vcol CHAR PROMPT
'Determine the minimum and
maximum value of a column.
Enter the column name: '
ACCEPT vtable CHAR PROMPT 'Enter
the corresponding table name: '
SET VERIFY OFF
SELECT MIN(&vcol), MAX(&vcol)
FROM &vtable
```

- b)** Explain each line in the following SQL script and then, in one sentence, describe the purpose of the script.

```
01 /*
02 -----
03 File name:   rows.sql
04 Purpose:
05 Created by:  H. Ashley on January 7, 2008
06 Modified by: A. Christa on September 29, 2008
07 -----
08 */
09 SET TERM OFF
10 SET PAGESIZE 0
11 SET FEEDBACK OFF
12 SPOOL temp.lst
13 SELECT 'SELECT ' || ' ' || table_name || ' ' ||
14        ', COUNT(*) ' || CHR(10) ||
15        ' FROM ' || LOWER(table_name) || ';'
16 FROM user_tables;
17 SPOOL OFF
18 SET FEEDBACK 1
19 SET PAGESIZE 20
20 SET TERM ON
21 @temp.lst
```

c) Explain when you would execute a script like the following.

```
--File: re_create_seq.sql
SET PAGESIZE 0
SET LINESIZE 100
SET FEEDBACK OFF
SPOOL re_create_seq.out
SELECT 'CREATE SEQUENCE ' ||
sequence_name ||
      ' START WITH ' || last_number ||
CHR(10) ||
      ' INCREMENT BY ' ||
increment_by ||
      DECODE(cache_size, 0, '
NOCACHE',
      ' CACHE ' || cache_size) || ';'

```

650

```
FROM seq;
SPOOL OFF
SET PAGESIZE 20
SET LINESIZE 80
SET FEEDBACK ON
SET TERM ON

```

651

d) Create a user-defined report called Student Schema Indexes in SQL Developer, using the following SQL.

```
SELECT index_name,  
       index_type,  
       table_name,  
       uniqueness  
FROM user_indexes  
ORDER BY index_name
```

LAB 14.2 EXERCISE ANSWERS

- a) Enter all the following commands in a file named minmaxval.sql and then execute the script in SQL*Plus or SQL Developer. For the column name supply the value CAPACITY and for the table name enter the value SECTION. What do you observe in either SQL Developer or SQL*Plus?

```
ACCEPT vcol CHAR PROMPT  
'Determine the minimum and  
maximum value of a column.  
Enter the column name: '  
ACCEPT vtable CHAR PROMPT 'Enter  
the corresponding table name: '  
SET VERIFY OFF  
SELECT MIN(&vcol), MAX(&vcol)  
FROM &vtable
```

ANSWER: The PROMPT and ACCEPT commands allow for user-friendly inputs and prompts.

```
Determine the minimum and maximum value of a column.  
Enter the column name: CAPACITY  
Enter the corresponding table name: SECTION
```

```
MIN(CAPACITY) MAX(CAPACITY)  
-----  
10 25
```

```
1 row selected.
```

The ACCEPT SQL*Plus command defines a variable that can then be referenced with the ampersand symbol and permits prompting for the values. The SQL*Plus ACCEPT command allows for data type checking of the entered value.

- b) Explain each line in the following SQL script and then, in one sentence, describe the purpose of the script.

```
01 /*  
02 -----  
03 File name:  zoww.sql  
04 Purpose:  
05 Created by:  N. Ashley on January 7, 2008  
06 Modified by: A. Christi on September 29, 2008  
07 -----  
08 */  
09 SET TERM OFF  
10 SET PAGESIZE 0  
11 SET FEEDBACK OFF  
12 SPOOL temp.lst  
13 SELECT 'SELECT ' || ' ' || table_name || ' ' ||  
14        ', COUNT(*) ' || CHR(10) ||  
15        ' FROM ' || LOWER(table_name) || ' '  
16 FROM user_tables;  
17 SPOOL OFF  
18 SET FEEDBACK 1  
19 SET PAGESIZE 25  
20 SET TERM ON  
21 @temp.lst
```

ANSWER: The purpose of the script is to display a list of all user-accessible tables, together with a row count for each.

The script dynamically generates the following statements and spools them to the resulting temp.lst file.

```
SELECT 'course', COUNT(*)
      FROM course;
...
SELECT 'zipcode', COUNT(*)
      FROM zipcode;
```

The temp.lst file is then executed with the @temp.lst command, and a count of all rows for each table is displayed.

```
'STUDEN  COUNT(*)
-----
student          268

1 row selected.
...
'ZIPCOD  COUNT(*)
-----
zipcode          227

1 row selected.
```


Lines 1 through 8 show a multiline comment; the comment starts with `/*` and ends with `*/`. In line 9, the command `SET TERM OFF` turns off the output to the screen. Line 10 sets the `PAGESIZE` to zero, line 11 avoids any `FEEDBACK`, and line 12 spools the result of all subsequent statements to the `temp.lst` file in the current directory. Line 13 shows an example of the literal `SELECT`, concatenated with four single quotation marks. The four single quotation marks result in a single quotation mark in the spooled file, and the table name shows between them.

652

653

USING QUOTES IN SQL

As you see in many SQL statements, single quotation marks are used to enclose a text literal.

```
SELECT last_name
FROM student
WHERE last_name = 'Smith'
```

If you want to query, insert, update, or delete a value containing a single quote, prefix the quote with another quote.

```
SELECT last_name
FROM student
```

```
WHERE last_name = 'O'Neil'
```

To replicate a single quotation mark in a dynamic SQL script, you need four quotes: two individual quotes to represent a single quotation mark and two quotes to surround this text literal.

Line 14 displays the COUNT function to count rows. The CHR(10) function issues a new line in the spooled file. The resulting concatenation is then further combined with the literal FROM in line 15, together with the table name in lowercase and a semicolon.

Line 16 shows that the query is issued against the USER_TABLES data dictionary view. Line 17 ends the spooling to the file. Lines 18, 19, and 20 reset the SQL*Plus settings to their defaults. Line 21 runs the spooled temp.lst file and automatically issues the generated SQL statements. To run this script, use SQL*Plus not SQL Developer.

- c) Explain when you would execute a script like the following.

```
--File: re_create_seq.sql
SET PAGESIZE 0
SET LINESIZE 100
SET FEEDBACK OFF
SPOOL re_create_seq.out
```

```
SELECT 'CREATE SEQUENCE ' ||
sequence_name ||
      ' START WITH ' ||
last_number || CHR(10) ||
      ' INCREMENT BY ' ||
increment_by ||
      DECODE(cache_size, 0, '
NOCACHE',
      ' CACHE' || cache_size) || ';'
FROM seq;
SPOOL OFF
SET PAGESIZE 20
SET LINESIZE 80
SET FEEDBACK ON
SET TERM ON
```

ANSWER: There are times when you need to re-create certain database objects; perhaps you need to replicate the same setup in another database or make changes to database objects without writing new scripts from scratch.

```
CREATE SEQUENCE COURSE_NO_SEQ
START WITH 454
INCREMENT BY 1 NOCACHE;
...
CREATE SEQUENCE STUDENT_ID_SEQ
START WITH 401
```

INCREMENT BY 1 NOCACHE;

653

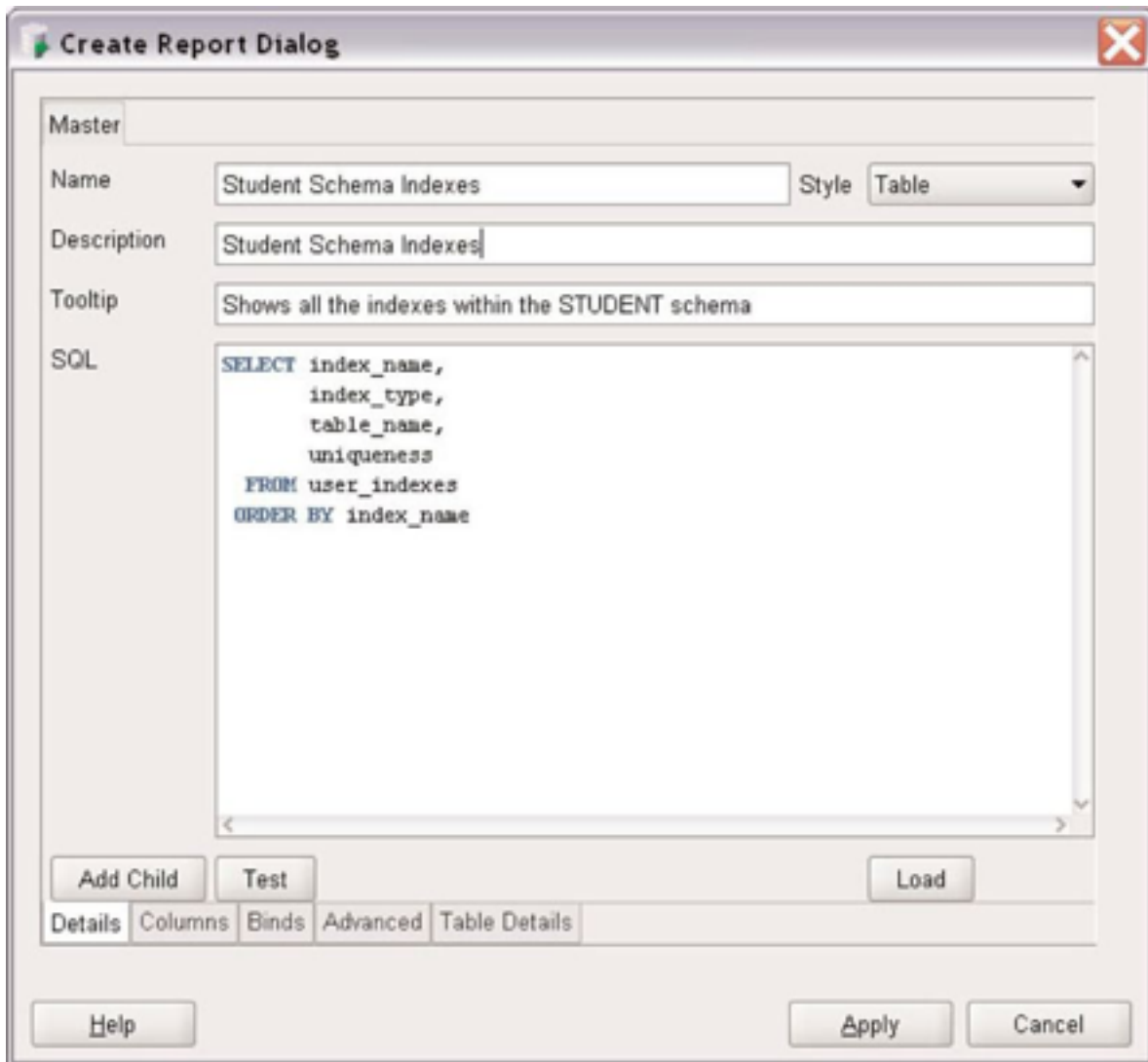
- d) Create a user-defined report called Student Schema Indexes in SQL Developer, using the following SQL.

654

```
SELECT index_name,  
       index_type,  
       table_name,  
       uniqueness  
FROM user_indexes  
ORDER BY index_name
```

ANSWER: In the User Defined Reports folder, right-click and choose the Add Report menu option. The Create Report Dialog screen appears. Fill in the Report Name, Description, Tooltip, and SQL statement boxes (see [Figure 14.18](#)). After you enter the SQL statement, you can click the Test button to see if the report works. To save the report, click the Apply button.

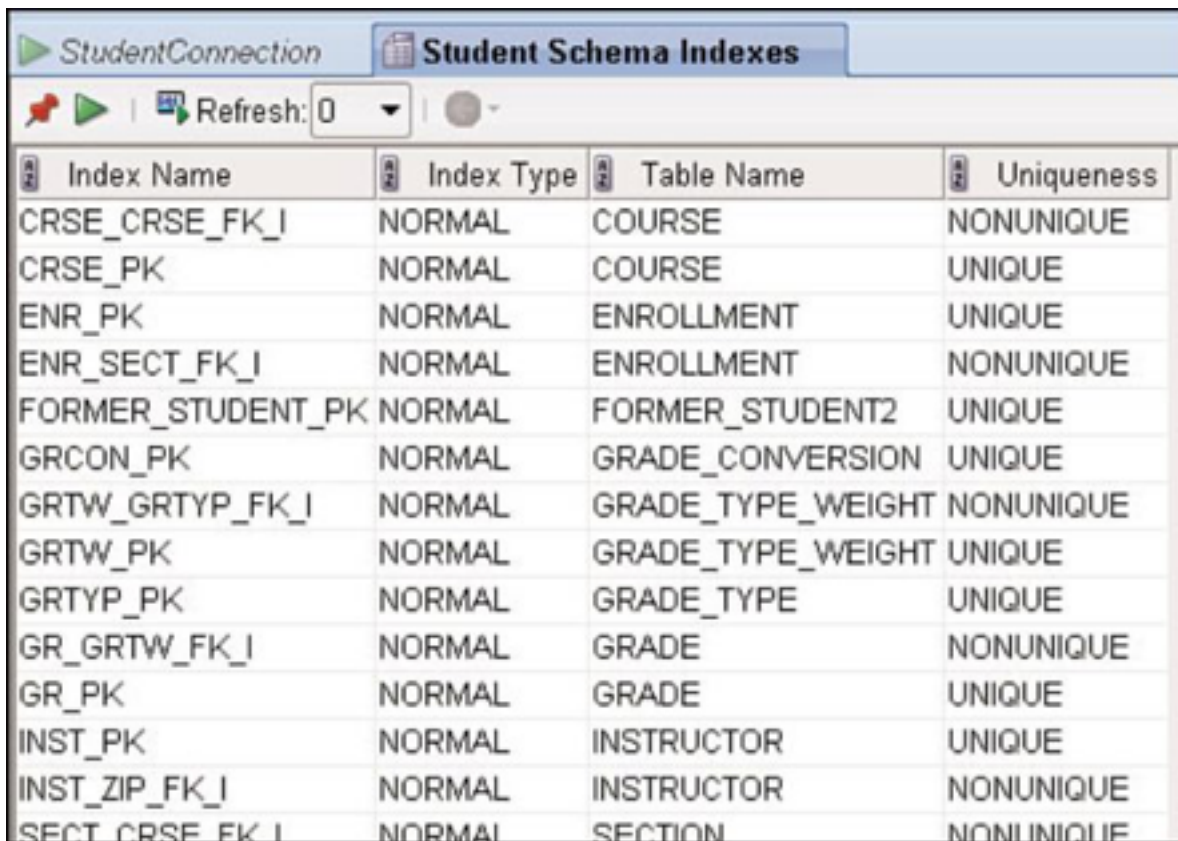
FIGURE 14.18 New report named Student Schema Indexes



[Figure 14.19](#) shows the result of the report. It displays the index name, the type of index, the table on which the index is based, and whether the index is unique.

654

FIGURE 14.19 Results of the Student Schema Indexes report



Index Name	Index Type	Table Name	Uniqueness
CRSE_CRSE_FK_I	NORMAL	COURSE	NONUNIQUE
CRSE_PK	NORMAL	COURSE	UNIQUE
ENR_PK	NORMAL	ENROLLMENT	UNIQUE
ENR_SECT_FK_I	NORMAL	ENROLLMENT	NONUNIQUE
FORMER_STUDENT_PK	NORMAL	FORMER_STUDENT2	UNIQUE
GRCON_PK	NORMAL	GRADE_CONVERSION	UNIQUE
GRTW_GRTYP_FK_I	NORMAL	GRADE_TYPE_WEIGHT	NONUNIQUE
GRTW_PK	NORMAL	GRADE_TYPE_WEIGHT	UNIQUE
GRTYP_PK	NORMAL	GRADE_TYPE	UNIQUE
GR_GRTW_FK_I	NORMAL	GRADE	NONUNIQUE
GR_PK	NORMAL	GRADE	UNIQUE
INST_PK	NORMAL	INSTRUCTOR	UNIQUE
INST_ZIP_FK_I	NORMAL	INSTRUCTOR	NONUNIQUE
SECT_CRSE_FK_I	NORMAL	SECTION	NONUNIQUE

CREATING A MASTER/CHILD REPORT IN SQL DEVELOPER

There are many choices on the Create Report Dialog screen (refer to [Figure 14.18](#)) that allow you to modify the report's layout and formatting. A very useful feature is the ability to create a master/child report, whereby

the information in the child result tab is dependent on the row of the master tab. To illustrate this capability, you will expand on the Student Schema Indexes report and add a child report to the bottom of the report. For each index in the master report, it will display the index's respective list of columns.

To modify the report, right-click the Student Schema Indexes report and choose Edit to modify the report. Click the Add Child button and enter the following SQL statement.

```
SELECT index_name, table_name,  
       column_position, column_name  
FROM user_ind_columns  
WHERE index_name = :INDEX_NAME
```

The bind variable :INDEX_NAME does not prompt you for input, but it is part of the child SQL query to link the two screens together. Fill in a report name and the description (see [Figure 14.20](#)).

655

FIGURE 14.20 Entering the child report information

The screenshot shows the 'Create Report Dialog' window. The 'Master' tab is active, showing the following fields:

- Name: Student Schema Indexes
- Style: Table
- Description: Student Schema Indexes
- Tooltip: Shows all indexes within the STUDENT schema
- SQL:

```
SELECT index_name,
       index_type,
       table_name,
       uniqueness
FROM user_indexes
ORDER BY index_name
```

Below the Master tab, there are buttons for 'Add Child', 'Test', and 'Load'. The 'Details' tab is selected, showing the 'Child1' tab with the following fields:

- Type: Child
- Style: Table
- Name: Indexed Columns
- Description: Indexed Columns
- Tooltip: (empty)
- SQL:

```
SELECT index_name, table_name,
       column_position, column_name
FROM user_ind_columns
WHERE index_name = :INDEX_NAME
```

Below the Child1 tab, there are buttons for 'Remove' and 'Load'. At the bottom of the dialog, there are buttons for 'Help', 'Apply', and 'Cancel'.

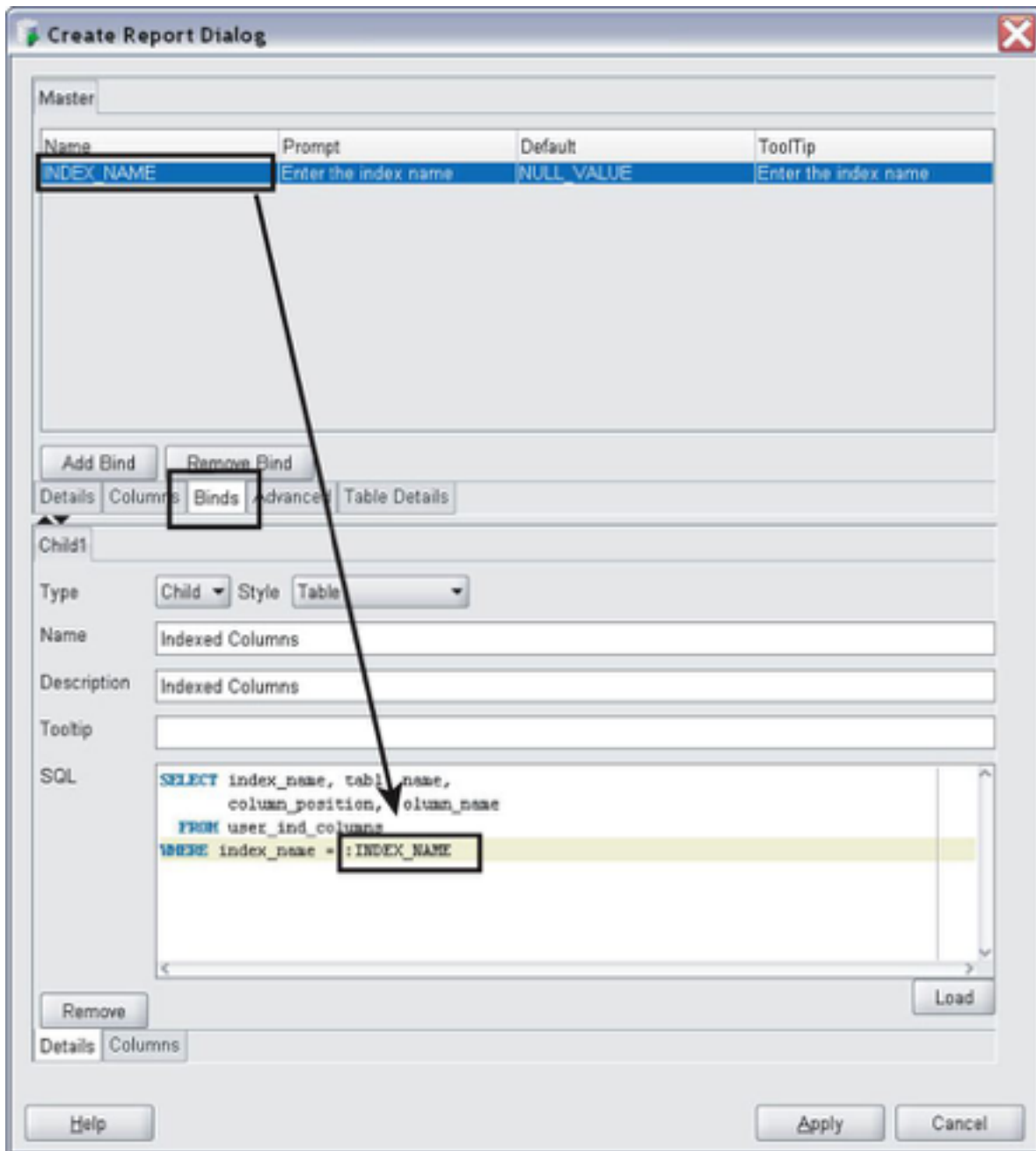
The last step is to create the bind variable
INDEX_NAME in the Binds tab (see [Figure 14.21](#)).

This establishes the link between the master report and the child report. The bind variable name in this tab and the SQL statement must agree.

656

657

FIGURE 14.21 The Binds tab



Finally, you can run the report. When you select a row in the master, the corresponding columns that make up this index are displayed in the Indexed Columns tab.

[Figure 14.22](#) shows the GR_PK index name highlighted and the corresponding four indexed columns in the defined order.

657

658

FIGURE 14.22 The master/child report for Student Schema Indexes

Index Name	Index Type	Table Name	Uniqueness
GRTW_GRTYP_FK_I	NORMAL	GRADE_TYPE_WEIGHT	NONUNIQUE
GRTW_PK	NORMAL	GRADE_TYPE_WEIGHT	UNIQUE
GRTYP_PK	NORMAL	GRADE_TYPE	UNIQUE
GR_GRTW_FK_I	NORMAL	GRADE	NONUNIQUE
GR_PK	NORMAL	GRADE	UNIQUE
INST_PK	NORMAL	INSTRUCTOR	UNIQUE

Index Name	Table Name	Column Position	Column Name
GR_PK	GRADE	1	STUDENT_ID
GR_PK	GRADE	2	SECTION_ID
GR_PK	GRADE	3	GRADE_TYPE_...
GR_PK	GRADE	4	GRADE_CODE...

You can add additional child tabs to the report. These child tabs can be linked to the master report, or the reports can be independent queries.



SQL Developer supplies many data dictionary reports, which you can review to get ideas about different presentation styles and features.

SHARING REPORTS

After you create your report, you can share the report with other users, who can then run the same report on their own, using their individual connection information and their own variables. To share the report, you need to export the report definition to an XML file on a directory where others can access this file. All users who want to run shared reports need to add a Shared Reports folder to their SQL Developer setup. You can accomplish this by selecting Tools, Preferences and then selecting Database: User Defined Extensions. In the screen that appears, add a row with the type of REPORT and add the location and name of the shared report. When you restart SQL Developer, the Reports

pane will show the Shared Reports folder containing the report.

658

659

Lab 14.2 Quiz

In order to test your progress, you should be able to answer the following questions.

- 1)** The following statements are SQL*Plus commands, not SQL commands.

```
SET FEEDBACK ON  
COL student FORMAT A20  
START
```

_____ **a)** True

_____ **b)** False

- 2)** What is the result of the following SELECT statement?

```
SELECT 'HELLO ' || CHR(10) ||  
'THERE'  
FROM dual
```

_____ **a)** HELLO THERE

_____ **b)** HELLO

_____ THERE

_____ **c)** Invalid query

3) Dynamic SQL scripts are useful for generating SQL statements.

_____ **a)** True

_____ **b)** False

4) The following SELECT statement returns a single quote.

```
SELECT ' ' ' '
FROM dual
```

_____ **a)** True

_____ **b)** False

5) Bind variables in SQL Developer allow you to prompt for user input. They can also link master/child reports.

_____ **a)** True

_____ **b)** False

ANSWERS APPEAR IN APPENDIX A.

659

660

WORKSHOP

The projects in this section are meant to prompt you to utilize all the skills you have acquired throughout this chapter. The answers to these projects can be found at

the companion Web site to this book, located at www.oraclesqlbyexample.com.

- 1) Describe the result of the following query.

```
SELECT table_name, column_name,  
       comments  
FROM   user_col_comments
```

- 2) Explain the differences between the views USER_USERS, ALL_USERS, and DBA_USERS.
- 3) What are the underlying data dictionary views for the public synonyms TABS and COLS?
- 4) Write a dynamic SQL script to drop all views in the STUDENT schema. If there are no views, create some to test your script.