Banner Oracle
Introduction to Oracle and SQL Training Workbook

January 2006

Using Oracle for Banner Release 7x

SUNGARD HIGHER EDUCATION

What can we help you achieve?

SunGard Higher Education

4 Country View Road
Malvern, Pennsylvania 19355
United States of America
(800) 522 - 4827

Customer Support Center website
http://connect.sungardhe.com

Distribution Services e-mail address

distserv@sungardhe.com

Other services

In preparing and providing this publication, SunGard Higher Education is not rendering legal, accounting, or other similar professional services. SunGard Higher Education makes no claims that an institution's use of this publication or the software for which it is provided will insure compliance with applicable federal or state laws, rules, or regulations. Each organization should seek legal, accounting and other similar professional services from competent providers of the organization's own choosing.

Trademark

Notice of rights

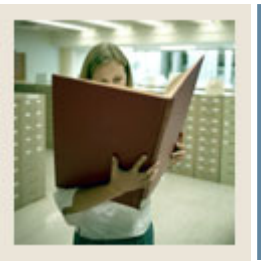


Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Section A: Introduction

Lesson: Overview

Jump to TOC

Introduction

This course introduces the user methods used to retrieve and manipulate data stored within an Oracle relational database. SQL*Plus is the application, or tool, which will be used to access the data.

Intended audience

SQL is used for all types of database activities by many types of users. However, in order for attendees to receive the optimum benefit of this training, Sungard Higher Education recommends that prospective students come from one of the following groups:

- System administrators
- Database administrators
- Security administrators
- Application programmers
- Decision support system personnel
- End users who have extensive contact with the database

Objectives

At the end of this course, participants will be able to

- Write statements to obtain information from the database
- Write statements to generate reports
- Manipulate data and process transactions
- Create and modify data tables
- Write programs in which SQL statements are enclosed within procedural statements (such as IF statements)
- Understand the various types of Oracle objects that can be stored in a database

Section contents

Section A: Introduction

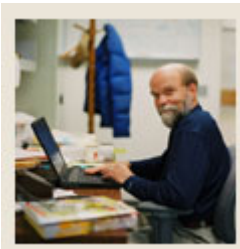
Lesson: Course Overview

Jump to TOC

Overview

This course will instruct attendees as to how SQL (Structured Query Language) is used to provide information and update data. The goal of this manual is to help developers tackle real world problems faced every day (at least those problems that can be solved with software). SQL and the rest of Oracle's products offer the potential for incredible development productivity.

This workbook is filled with sample code fragments, as well as complete application components, which can be applied immediately to a situation. Using this workbook, it is Sungard Higher Education's intent to guide participants through the analytical process to develop efficient, maintainable code to successfully support the development and maintenance of an Oracle database.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Overview

Jump to TOC

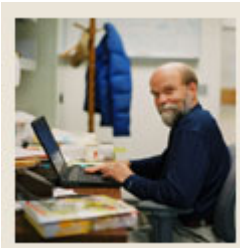
Introduction

This section provides a basic discussion of the components of Oracle SQL and SQL*Plus.

Objectives

This section will examine the following:

- Definition of a relational database
- Definition of a schema
- Object names
- Character sets
- Simple and compound symbols
- Login
- Tables
- Columns
- DUAL table
- Data Dictionary
- SQL Buffer



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Overview (Continued)

Jump to TOC

Section contents

Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: SQL Statements

Jump to TOC

Overview

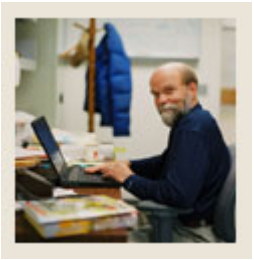
This training manual contains a list of common SQL (Structured Query Language) statements. SQL is used to create, store, modify, retrieve, and manage information in any Oracle database. This course will address how SQL is used within an Oracle database.

About SQL

SQL is a set of commands governed by the American National Standards Institution (ANSI) and the International Standards Organization (ISO). Many of the commands in this manual are a superset of the ANSI standard which will be used in the environment called SQL*Plus.

As of Oracle release 9i Version 2, Oracle has adopted the SQL99 standard of the SQL language.

For a complete listing of all SQL and SQL*Plus commands, please refer to the Oracle Database SQL Reference and SQL*Plus reference manuals. These books and more are available in the Oracle Database Documentation Library at the Oracle Technology Network website http://www.oracle.com/technology/documentation/database10gR2.html.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Oracle's Relational Database

Jump to TOC

What is a relational database?

A relational database is a collection of data items which can be accessed or reassembled in many different ways without having to reorganize the tables.

For instance, as a developer, you may be asked to create two reports:

- A report that lists a *student ID*, *Name*, Birth Date, *Social Security number*, and home address for a group of students
- A report that lists a *student ID*, *Name*, *Social security number*, and courses being taken during the current term

Although these are two different reports, they do share some things in common. You are trying to access the Student ID, Name, and Social Security number for both.

Data access

Relational databases recognize the fact that data needs to be accessed several ways in order to be valuable.

What is a schema?

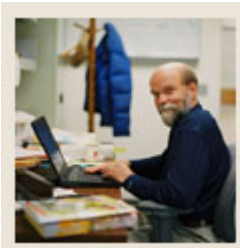
A schema is a collection of related objects. A schema is owned by a database user and has the same name as the user. Every database object that is created by a user then becomes part of the user's schema.

This will be discussed in more detail in Section J.

Oracle object names

As with any language, there are conventions used in naming objects. The following rules govern naming objects.

- Names must be 1 to 30 characters in length
- Names cannot contain quotation marks or special characters
- Names are not case sensitive**
- Each name must begin with an alphabetic character
- A name cannot be an Oracle reserved word
- A name cannot be the duplicate of another database object owned by the same user



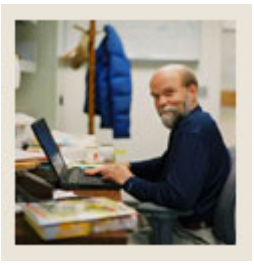
Object Naming (continued)

SQL is not a case-sensitive language. Uppercase and lowercase letters are treated the same way with the exception of literal strings (alphanumeric characters surrounded by single quotes) and character variables (variables designated with the "&" prefix).

Although SQL is case-insensitive, stick to using UPPERCASE for keywords and reserved words. It is much easier to read.

Type	Character
Letters**	A-Z, a-z
Digits	0-9
Symbols	_ (underscore)

**Objects can generally be created and referenced without regard to case. However, object names are stored in the internal data dictionary in UPPER CASE and when selecting from data dictionary views the object name must be referenced in UPPER CASE.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Oracle's Relational Database (Continued)

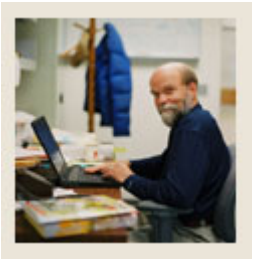
Jump to TOC

SQL Special Characters

The standard symbols exist in Oracle SQL for elementary arithmetic functions. There are other symbols that have special meaning when writing SQL statement in Oracle.

Simple and compound symbols

Symbol	Description
+	Addition symbol
-	Subtraction symbol
*	Multiplication symbol
/	Division symbol
=	Equality symbol
<	Less than symbol
>	Greater than symbol
;	Statement terminator
%	Multi-byte wild card symbol
_	Single byte wild card symbol (underscore)
<> and !=	Not equals
||	Concatenation operator
<= and >=	Relational operator
--	Single line comment indicator (dash dash)
/* and */	Multi-line comment block delimiters
&	Parameter indicator
:	Bind Variable indicator
@	Run statement in SQL*Plus
( )	Override precedence



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Login

Jump to TOC

Logging in

To retrieve and manipulate data, first connect to the database.

There are two ways to connect to the SQL*Plus environment.

Login from Unix/VMS prompt

At the operating system prompt, type SQLPLUS. Then enter a username and password. After entering a password, the system goes to the SQL prompt:

```
prompt$   SQLPLUS


SQL*Plus: Release 9.2.0.1.0 - Production on Mon Dec 19 11:55:23 2005

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.


Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 – Production

SQL>
```

Your output may differ from what is shown here, depending on which releases of the software you have installed.

Section B: Introduction to Oracle SQL and SQL*Plus

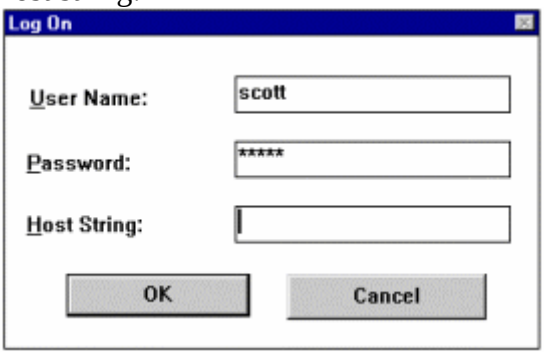
Lesson: Login (Continued)

Jump to TOC

Login from PC

The SQL*Plus program may be access via several options. It may have been created as an icon on your desktop, or go to Start → Programs → {Oracle Software – name here may vary} → Application Development → SQL*Plus.

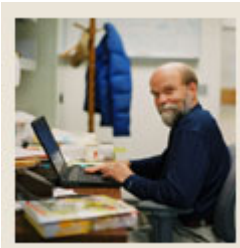
When logging in via client server, the prompt may also include a request for a database (SID) or host string:



Contact the database administrator for proper database names (SID).

Release numbers

Note the Oracle release numbers. They are normally overlooked in day to day activity, but when seeking assistance from SunGard Higher Education or the Oracle Corporation, undoubtedly these version numbers will be requested by the contact person.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: SQL Buffer

Jump to TOC

SQL buffer

SQL*Plus will store the last command in the SQL buffer. Use the following commands to edit statements stored in the buffer.

Command	Abbreviation	Purpose
APPEND *text*	a *text*	Add *text* to the end of a line
CHANGE/*old/new*	c/*old/new*	Change *old* to *new* in line
CHANGE/*text*	c/*text*	Delete *text* from a line
CLEAR BUFFER	CL BUFF	Delete lines
CLEAR SCREEN	CL SCR	Clear the screen
DEL	None	Delete current line
INPUT	I	Add one or more lines
LIST	L	List all lines in buffer
LIST *n*	l *n*	List a specific line
LIST *m n*	l *m n*	List a range of lines

Bypass the Buffer

At the SQL> prompt, type the keyword EDIT. This action will open the system editor. Save the file and exit the editor in the manner defined by the editor to return to the SQL> prompt.

If you just type “edit” at the SQL prompt, it will edit the last statement issued. This statement will be saved in a file called afiedt.buf. When you save and exit the editor, the corrected statement will appear as the new current statement.

If you save the file to another name from the editor, it will not appear as the current statement in SQL buffer. You will have to run it explicitly as outlined below.

Alternately, save the current statement from the SQL prompt by typing:

```
SQL>          Save <file_name>   { replace }
```

Use the keyword *replace* if the file already exists. The file will automatically be saved with a .sql extension.

To execute the commands specified in the file, type one of these command lines (they are synonymous). SQL*Plus assumes the file has an extension of .sql. If the files have an alternate extension, supply it with the file name.

```
SQL>          START <file_name >
SQL>          @<file_name >
```

View editor settings

In order to view the editor settings, type the keyword DEFINE at the SQL prompt.

```
SQL> DEFINE
DEFINE _CONNECT_IDENTIFIER = "TRNG" (CHAR)
DEFINE _SQLPLUS_RELEASE = "902000100" (CHAR)
DEFINE _EDITOR          = "Notepad" (CHAR)
DEFINE _O_VERSION       = "Oracle9i Enterprise Edition Release
9.2.0.6.0 - Production
JServer Release 9.2.0.6.0 - Production" (CHAR)
DEFINE _O_RELEASE       = "902000600" (CHAR)
```

New variables in Oracle 10g:

```
SQL>  define
DEFINE _DATE            = "30-DEC-05" (CHAR)
DEFINE _CONNECT_IDENTIFIER = "dw10g" (CHAR)
DEFINE _USER            = "TRAIN01" (CHAR)
DEFINE _PRIVILEGE       = "" (CHAR)
DEFINE _SQLPLUS_RELEASE = "1002000100" (CHAR)
DEFINE _EDITOR          = "Notepad" (CHAR)
DEFINE _O_VERSION       = "Oracle Database 10g Enterprise Edition Release
10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options" (CHAR)
DEFINE _O_RELEASE       = "1002000100" (CHAR)
```

Change the default editor

To change the default editor, use the keyword DEFINE_EDITOR:

```
SQL> DEFINE_EDITOR = <editor name>
```

Include the full path of the editor on Windows machines if the path is not part of your environment settings.

Setting the SQL Prompt

When working with multiple databases, it may be desirable to know which database you are connected to. This can be done in version 9i Release 2 using the new _CONNECT_IDENTIFIER variable. Issue the following command:

```
SQL> SET SQLPROMPT '&_CONNECT_IDENTIFIER > '
```

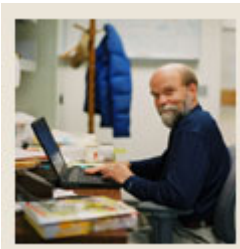
Your prompt will now look like (assuming TRNG is the database you connected to):

```
TRNG>
```

Note: if you connect to another database from the SQL prompt, it will not automatically update the prompt, you will need to reissue the SET SQLPROMPT command.

Running an SQL statement

To execute an SQL statement that you typed into the buffer, terminate it with a semicolon (;) and it will automatically run. To re-run the statement in the buffer or execute the current statement after editing, use the / (forward slash).



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Tables

Jump to TOC

Tables

A table is the basic structure to hold user information. Think of a table as a spreadsheet made up of columns and rows. Column definitions fall under three categories: character, numeric, and date data types.

Table definition

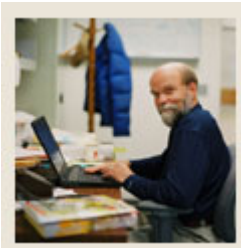
To view the definition of a table, use the DESC command (abb. for describe):

```
SQL>     DESC <table_name>
```

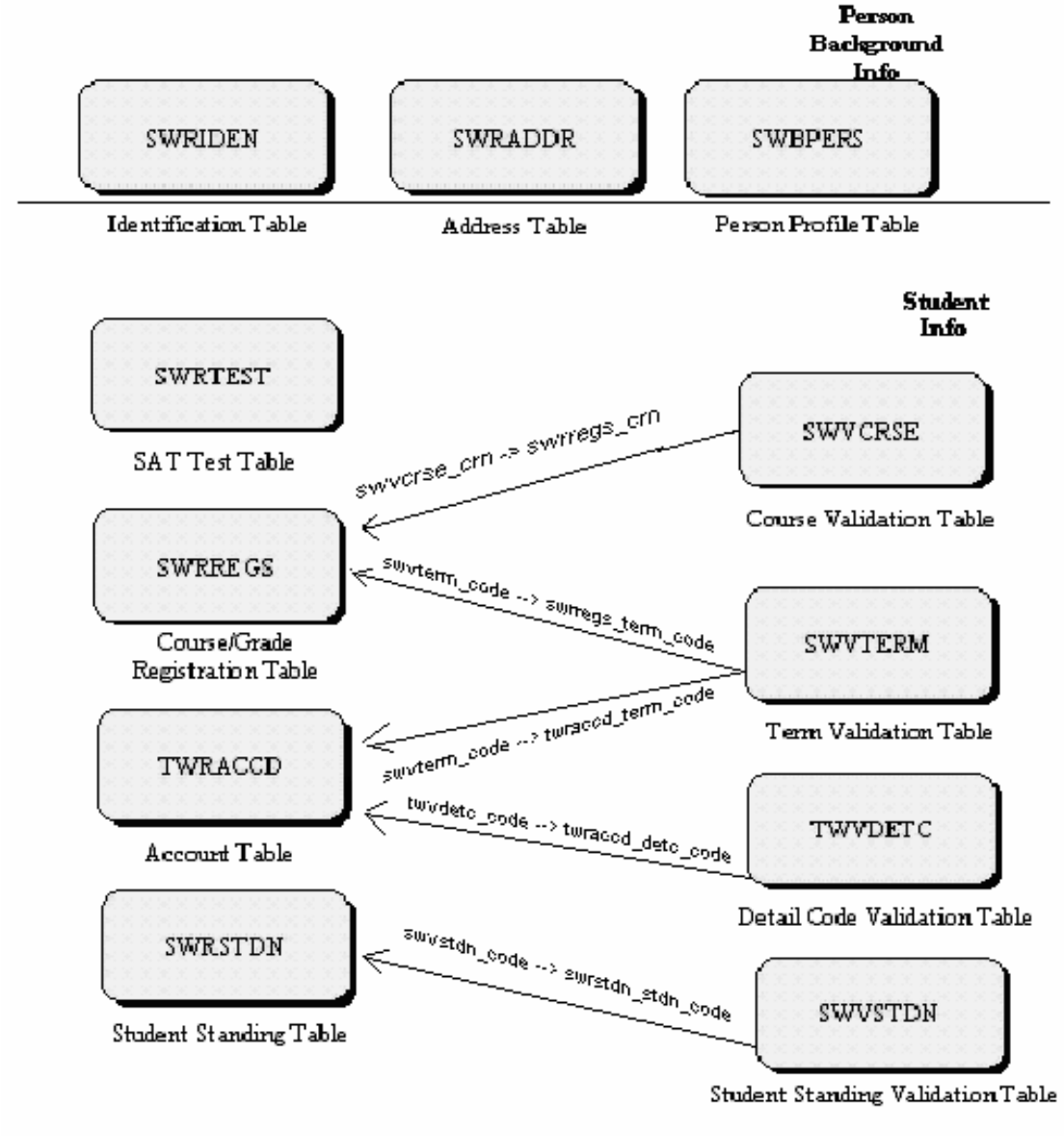
Table examples

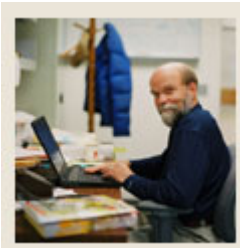
Many of the code examples and exercises in this manual use tables designed for this course. Take a moment to become familiar with these tables using the DESC command:

```
SQL>     DESC  swriden
SQL>     DESC  swbpers
SQL>     DESC  swraddr
SQL>     DESC  swvcrse
SQL>     DESC  swrregs
SQL>     DESC  swrstdn
SQL>     DESC  swrtest
SQL>     DESC  swvstdn
SQL>     DESC  swvtele
SQL>     DESC  swvterm
SQL>     DESC  twraccd
SQL>     DESC  twvdetc
```



Diagram


Person Background Info
SWRIDEN
SWRADDR
SWBPERS
Identification Table
Address Table
Person Profile Table
Student Info
SWRTEST
SAT Test Table
SWVCRSE
Course Validation Table
swvcrse_crn -> swrregs_crn
SWRREGS
Course/Grade Registration Table
swvterm_code --> swrregs_term_code
SWVTERM
Term Validation Table
swvterm_code --> twracod_term_code
TWRACCD
Account Table
twvdetc_code --> twracod_detc_code
TWVDETC
Detail Code Validation Table
SWRSTDN
Student Standing Table
swvstdn_code --> swrstdn_stdn_code
SWVSTDN
Student Standing Validation Table



Characters

The first character in the name of all Banner tables identifies the system that owns it.

The second character is system specific and identifies the application module to which the table refers.

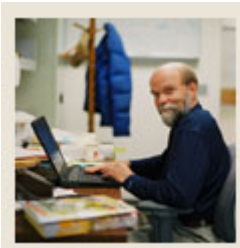
The third character identifies the type of table: B(ase), R(epeating), or V(alidation) table.

The fourth through seventh positions represent a four-character description for the table.

Example

For example: SWRIDEN

- **S**(tudent)
- **W** (client-developed object)
- **R**(epeating table)
- **IDEN**(tification)



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Columns

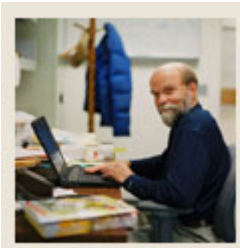
Jump to TOC

Tables are made up of columns. The number of columns in a table can range from 1 to 1000. Each column is defined using a specific data type.

The following are the most common data types:

- VARCHAR2(n)
  Variable length character string having a maximum size *n* of 4000.
- CHAR(n)+
  Fixed length character data having a maximum size *n* of 255.
- VARCHAR++
  Currently synonymous with VARCHAR2 data types.
- CLOB/BLOB/BFILE+++
  Variable length data up to 4 gigabytes.
- NUMBER(p,s)
  Numeric data type having a precision *p* and scale *s*.
- DATE
  Valid dates range from January 1, 4712 BC to December 31, 9999 AD.

+ The CHAR(n) data type is a backwards compatible data type associated with DB2 applications.
++ It is highly recommended that this data type never be used. Oracle may change its definition in a future release.
+++ Note: BFILES are pointers to external files. The files themselves should not exceed 4 Gb.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Data Dictionary

Jump to TOC

Definition

The data dictionary is a collection of views containing information concerning the database, database objects, users, and events. It is also called "meta data."

To access the views contained in the data dictionary, type the following:

```
SQL>     SELECT * FROM DICTIONARY;
```

Importance

The contents of the dictionary tables and views will be explored as they relate to the topics of discussion. It is highly recommended that users become familiar with the contents of the data dictionary. A great deal of information concerning the Banner database may be gleaned from the data dictionary.

Dictionary views

The dictionary views can be divided into three categories: ALL, USER, and DBA views.

- Dictionary view names beginning with "USER" contain information about the database objects and events owned by a schema (user)
- Dictionary view names beginning with "ALL" contain information about the database objects to which the user has access

Refer to an Oracle Server Reference manual for a complete listing of data dictionary views and their descriptions.

Selectable views

The following are data dictionary views that can be selected to learn more information about objects in the database (each have a USER_ and ALL_ equivalent):

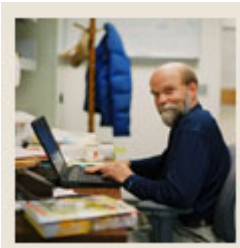
- USER_TABLES
- USER_TAB_COMMENTS
- USER_VIEWS
- USER_OBJECTS
- USER_CONSTRAINTS
- USER_TAB_COLUMNS
- USER_COL_COMMENTS
- USER_INDEXES
- USER_SOURCE
- USER_CONS_COLUMNS

List table information

To list all of the table information within your schema, type the following:

- SELECT * FROM USER_TABLES;

A printed listing of the contents of the tables used in this class may be found in Section O.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: DUAL

Jump to TOC

DUAL table

DUAL is a table automatically created by Oracle along with the data dictionary. It is in the schema of the user SYS, but is accessible to all users by the name DUAL. It has one column, DUMMY, defined to be a VARCHAR2, and it contains one row with the value of "X".

Selecting from DUAL

Selecting from DUAL is useful for computing a constant expression with the SELECT command. Because it has only one row, the constant is only returned once. Since all select statements must have a FROM, the DUAL table satisfies the syntax requirements.

The following example obtains the current system date:

```
SQL>     SELECT SYSDATE FROM DUAL;

  SYSDATE
---------
19-DEC-05
```

This examples show how to do a calculation with the dual table:

```
SQL >select (3 * 12) / 7 my_calc from dual;

  (3*12)/7
----------
5.14285714
```

Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: SQL*Plus Formatting

Jump to TOC

SQL*Plus Formatting

SQL*Plus makes some assumptions about how to display the data returned from your query. You may alter the aspects of this display by using some built in SQL*Plus commands.

This section will explore:

- Alternate column headings
- Larger line and page sizes
- Showing only one page of data at a time

Additional formatting techniques can be found in Section M – SQL*Plus reporting.

COLUMN command

Through the SQL*Plus COLUMN command, you can change the column headings and reformat the column data in your query results.

Changing column headings

SQL*Plus uses column or expression names as default column headings when displaying query results. Column names are often short and cryptic, however, and expressions can be hard to understand. You can define a more useful column heading with the HEADING clause of the COLUMN command in the format shown below:

```
COLUMN column_name HEADING column_heading
```

The new headings will remain in effect until you enter different headings, reset each column's format or exit from SQL*Plus.

EXAMPLE: Enter the following line:

```
SQL> COLUMN DETC_CODE HEADING 'Category'
```

Quotation marks

To change a column heading to two or more words, enclose the new heading in single or double quotation marks when you enter the COLUMN command. To display a column heading on more than one line, use a vertical bar (|) where you want to begin a new line.

EXAMPLE: Enter the following line:

```
SQL> COLUMN BILL_DATE HEADING 'Billing|Date'
```

Display query

Type in the rest of the column headings so that the query displays:

```
                          Billing
Term   Name               Date      Category    Amount
------ ------------------ --------- -------- ---------
200402 Erickson, Ralph    21-MAY-05 LABS           120
200402 Erickson, Ralph    21-MAY-05 DORM          1000
200501 Brown, Julie       21-MAY-05 BOOK         300.2
200501 Brown, Julie       21-MAY-05 TUIT        1500.5
200501 Brown, Julie       28-MAY-05 BKSC           700
200501 Erickson, Ralph    21-MAY-05 MEAL           900
...
```

Underline character

To change the character used to underline each column heading, set the UNDERLINE variable of the SET command to the desired character.

EXAMPLE: Enter the following:

```
SQL> SET UNDERLINE =

SQL> /
```

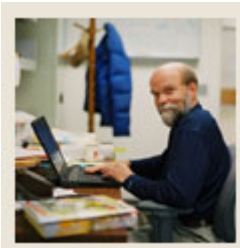
SQL*Plus displays the following results:

```
                          Billing
Term   Name               Date      Category    Amount
====== ================== ========= ======== =========
200402 Erickson, Ralph    21-MAY-05 LABS           120
200402 Erickson, Ralph    21-MAY-05 DORM          1000
200501 Brown, Julie       21-MAY-05 BOOK         300.2
200501 Brown, Julie       21-MAY-05 TUIT        1500.5
200501 Brown, Julie       28-MAY-05 BKSC           700
200501 Erickson, Ralph    21-MAY-05 MEAL           900
200501 Erickson, Ralph    21-MAY-05 CRED           400
...
```

Now change the underline character back to a dash:

```
SQL> SET UNDERLINE '-'
```

Enclose the dash in quotation marks; otherwise, SQL*Plus interprets the dash as a hyphen indicating that you wish to continue the command on another line.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Formatting Columns (Continued)

Jump to TOC

Formatting your columns

Although we now have customized our headings for the report, we also want to change the appearance of the data itself for the report.

The COLUMN command identifies the column you want to format and the model you want to use, as shown below:

```
COLUMN column_name FORMAT model
```

The format model will stay in effect until you enter a new one, reset the column's format, or exit from SQL*Plus.

Formatting a character column

The default display width for CHAR and VARCHAR2 (VARCHAR) values is the width defined for the column in the database. The default display can be changed using the COLUMN command. Use a format model consisting of the letter A (for alphanumeric), followed by a number representing the width of the column in characters.

You may notice that the Category title is truncated and that the Name column is taking up a lot of space. First, we will decrease the size of the Name column, and then increase the length of the TWRACCD_DETC_CODE column.

To set the width of the column NAME to 25 characters, enter:

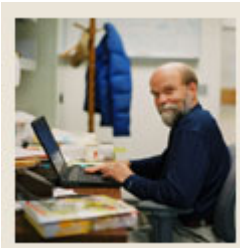
```
SQL> COLUMN NAME FORMAT A25
```

To set the width of the column TWRACCD_DETC_CODE to eight characters and rerun the current query, enter:

```
SQL> COLUMN TWRACCD_DETC_CODE FORMAT A8
SQL> /
```

SQL*Plus displays the results:

```
                              Billing
Term   Name                    Date      Category  Amount
------ -------------------- --------- -------- -------
200402 Erickson, Ralph      21-MAY-05 LABS         120
200402 Erickson, Ralph      21-MAY-05 DORM        1000
200501 Brown, Julie         21-MAY-05 BOOK       300.2
200501 Brown, Julie         21-MAY-05 TUIT      1500.5
200501 Brown, Julie         28-MAY-05 BKSC         700
200501 Erickson, Ralph      21-MAY-05 MEAL         900
200501 Erickson, Ralph      21-MAY-05 CRED         400
...
```



Formatting a number column

Use number format models to add commas, dollar signs, angle brackets (around negative values), and/or leading zeros to numbers in a given column. You can also round the values to a given number of decimal places, display minus signs to the right of negative values (instead of to the left), and display values in exponential notation.

To display AMOUNT with a dollar sign, a comma, and the number zero instead of a blank for any zero values, enter the following command:

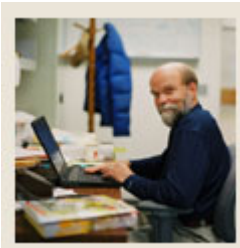
```
SQL> COLUMN AMOUNT  FORMAT $99,990.00
```

Now rerun the current query:

```
SQL> /
```

SQL*Plus displays the following output:

```
                                Billing
Term   Name                     Date      Category Amount
------ ------------------------ --------- -------- --------
200402 Erickson, Ralph          21-MAY-05 LABS      $120.00
200402 Erickson, Ralph          21-MAY-05 DORM    $1,000.00
200501 Brown, Julie             21-MAY-05 BOOK      $300.20
200501 Brown, Julie             21-MAY-05 TUIT    $1,500.50
200501 Brown, Julie             28-MAY-05 BKSC      $700.00
200501 Erickson, Ralph          21-MAY-05 MEAL      $900.00
200501 Erickson, Ralph          21-MAY-05 CRED      $400.00
200501 Erickson, Ralph          25-MAY-05 CASH      $800.00
...
```



List a column's current attributes

To list the current display attributes for a given column, use the COLUMN command followed by the column name as shown below:

```
SQL> COLUMN [column_name]
```

List all columns' current attributes

To list the current display attributes for all columns, enter the COLUMN command with no column names or clauses after it:

```
SQL> COLUMN
```

Reset a column to default values

To reset the display attributes for a column to their default values, use the CLEAR clause of the COLUMN command as shown below. Do not clear the columns now or you will undo the previous steps.

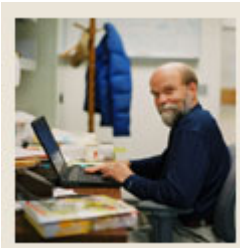
```
SQL> COLUMN [column_name] CLEAR
```

Reset all columns to default values

To reset all columns' display attributes to their default values, enter the following command:

```
SQL> CLEAR COLUMNS
```

You may wish to place the command CLEAR COLUMNS at the beginning of every command file to ensure that previously entered COLUMN commands will not affect queries you run in a given file. Also add to the file any column commands that are specific to your query.



Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Suppressing and Restoring Column Display Attributes

Jump to TOC

Suppress a column's display attributes

You can suppress and restore the display attributes you have given a specific column. To suppress a column's display attributes, enter a COLUMN command in the following form:

```
COLUMN column_name OFF
```

Restore defined attributes

The OFF clause tells SQL*Plus to use the default display attributes for the column, but does not remove the attributes you have defined through the COLUMN command. To restore the attributes you defined through COLUMN, use the ON clause:

```
COLUMN column_name ON
```

Adjusting Linesize

Linesize is the number of characters that can be presented on each line. When SQL*Plus cannot fit all columns across the screen, it will wrap them on to additional lines. There is a parameter you can set in SQL*Plus to adjust the width of data presented on the screen. However, screens are only so large, so while you may continue to increase the linesize, SQL*Plus will only show so much data on the screen.

The default linesize is 80 characters. The maximum linesize is 32767.

Other SQL tools may allow you to increase the linesize such that you can scroll back and forth without wrapping.

When sending out put to a file, you can increase the linesize beyond what SQL*Plus will show, and the data will be presented properly in the file even if it does not display on the screen properly.

```
SQL> SET LINESIZE 100
```

Adjusting Pagesize

Pagesize is the number of lines presented per logical page or screen or number of lines between repeats of column headings. Pagesize can be adjusted to show more lines of data in between column headings. By default, and based on a carry-over from older, smaller screen days, the default pagesize is only 14 lines per page. (In Oracle 10g the default pagesize has been increased to 24.) To show more data between repeats of column headings, increase the pagesize parameter.

```
SQL> SET PAGESIZE 40
```

Pagesize can also be increased substantially if the data will be written to a file. If you want to write data to a file and then load it into a spreadsheet, you may not want to repeat column headings at all. Pagesize can be set up to 50000.

Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Setting output to Pause

Jump to TOC

Adjusting Display of Data

When you select large amounts of data in a query such as the "SELECT * FROM dictionary" above, the data just scrolls by on the screen without stopping until all the data has been displayed. Screen buffers do not always store enough data for you to scroll backwards and see the beginning of the data from your query. You can adjust SQL*Plus to only display one "page" at a time (note this is the logical page defined by pagesize, above).

There are two commands that should be set when you want to pause the display of data so that it does not scroll off the screen. They are SET PAUSE ON and SET PAUSE <'string of characters'>.

The SET PAUSE ON command turns the pausing of data per logical page on. If the second command is not issued, the screen will pause waiting for you to hit the ENTER key to advance the data. One feature of the Pause command is that it will pause even before displaying the first logical page of data. Without a prompt, you may not know whether the query has data ready for you to view, or if it is still processing your request. You can add a prompt to the Pause command to let you know when data is ready to display. You issue the second SET PAUSE command and provide a string that will appear when another page of data is ready to display.

```
SQL> SET PAUSE ON
SQL> SET PAUSE 'Press any key to continue…'
SQL>   1* select * from swriden;
Press any key to continue...

SWRIDEN_PIDM SWRIDEN_I SWRIDEN_LAST_NAME         SWRIDEN_FIRST_N SWRIDEN_MI
S SWRIDEN_A
------------ --------- ------------------------ --------------- ------------
--- - ---------
SWRIDEN_USER_ID                SWRIDEN_DATA_ORIGIN
------------------------------ ------------------------------
       12340 857834585 Brown                    Julie           K
20-DEC-05
TRAIN_ORA101                   TRAINING

       12340 876536782 Brown                    Julie           K
I 20-DEC-05
TRAIN_ORA101                   TRAINING
...
```

To turn pausing off, issue the SET PAUSE OFF command.

Section B: Introduction to Oracle SQL and SQL*Plus

Lesson: Self Check

Jump to TOC

Directions

Use the information from this section to complete the following exercises.

Exercise 1

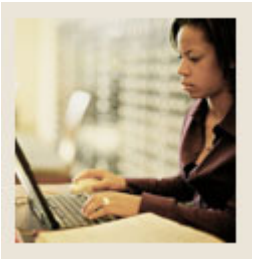
Using the login and database information provided by the instructor, make sure you can log into SQL*Plus on your computer.

Exercise 2

Issue the following statement to view the tables that you own:
SELECT * FROM USER_TABLES;
If no records are retrieved, contact the instructor.

Exercise 3

Issue the following statement to view the tables that you have permission to view (if the list is long and scrolls off the screen, consider setting PAUSE on with an appropriate pause message):
SELECT * FROM ALL_TABLES;



Section C: Introduction to the Query

Lesson: Overview

Jump to TOC

Introduction

This section provides an introduction to SQL queries.

Objectives

At the end of this section, participants will be able to write queries that perform the following:

- Select all or specific columns
- Remove duplicate rows
- Create substitute column headings

Section contents

Section C: Introduction to the Query

Lesson: Select Statements

Jump to TOC

Overview

The SELECT statement retrieves rows from one or more tables. It pulls all rows, or adds conditions so that only the information needed is retrieved.

```
SELECT <column name> <constant> FROM <table name>;
```

The purpose of a SELECT statement is to display columns and rows from one or more tables. This is also known as querying the database.

```
Examples
SQL> SELECT    swriden_last_name
       FROM    swriden;


SWRIDEN_LAST_NAME
------------------
Brown
Brown
Erickson
Erickson
Johnson
Jones
Jones-Erickson
Smith
White

SQL> SELECT    swriden_id
        FROM  swriden;

SWRIDEN_ID
----------
157834585
176536782
692568211
578549991
3539543
145672112
145672112
5829934
543853339
...
```

Section C: Introduction to the Query

Lesson: Selecting Multiple Columns

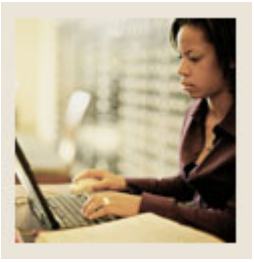
Jump to TOC

Separator

To select multiple columns, separate them by a comma.

Example

```
SQL> SELECT    swriden_last_name, swriden_first_name
       FROM    swriden;

SWRIDEN_LAST_NAME         SWRIDEN_FIRST_NAME
------------------------- ------------------
Brown                     Julie
Brown                     Julie
Erickson                  Ralph
Erickson                  Susan
Johnson                   Peter
Jones                     Sandy
Jones-Erickson            Sandy
Smith                     Robert
White                     Nancy
```



Section C: Introduction to the Query

Lesson: Selecting a Literal

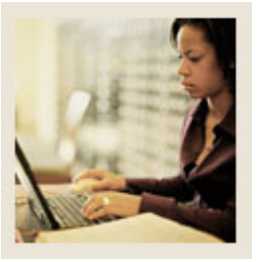
Jump to TOC

Literals

Selections do not have to be on a column from a table or database object. Select a literal if it is enclosed in quotes.

Example

```
  SQL> SELECT 'Student Name is', swriden_first_name, swriden_last_name
         FROM   swriden;

     'STUDENTNAMEIS' SWRIDEN_FIRST_NAME SWRIDEN_LAST_NAME
     --------------- ------------------ --------------------
     Student Name is Julie              Brown
     Student Name is Julie              Brown
     Student Name is Peter              Johnson
     Student Name is Robert             Smith
...
```



Section C: Introduction to the Query

Lesson: DISTINCT Clause

Jump to TOC

DISTINCT

The DISTINCT clause specifies that duplicate rows should be removed before the rows are returned. A row is considered a duplicate of another if every value for each column of the SELECT clause matches that of another row(s).

Note that distinct applies only to the columns in the select clause and not all the columns in the table.

Without DISTINCT clause

```
SQL> SELECT  swriden_last_name
       FROM  swriden;

SWRIDEN_LAST_NAME
-----------------
Brown
Brown
Erickson
Erickson
Johnson
Jones
Jones-Erickson
Smith
White
```

With DISTINCT clause

```
SQL> SELECT DISTINCT swriden_last_name
       FROM swriden;

SWRIDEN_LAST_NAME
-----------------
Brown
Erickson
Johnson
Jones
Jones-Erickson
Smith
White
```

Without DISTINCT

```
SQL> SELECT swriden_last_name, swriden_first_name
       FROM swriden;

SWRIDEN_LAST_NAME   SWRIDEN_FIRST_NAME
-----------------   ------------------
Brown               Julie
Brown               Julie
Smith               Robert
Johnson             Peter
Jones               Sandy
Jones-Erickson      Sandy
Erickson            Ralph
Erickson            Susan
White               Nancy
```

With DISTINCT

```
SQL> SELECT DISTINCT swriden_last_name, swriden_first_name
       FROM swriden;

SWRIDEN_LAST_NAME   SWRIDEN_FIRST_NAME
-----------------   ------------------
Brown               Julie
Erickson            Ralph
Erickson            Susan
Johnson             Peter
Jones               Sandy
Jones-Erickson      Sandy
Smith               Robert
White               Nancy
```

Section C: Introduction to the Query

Lesson: Selecting All Columns

Jump to TOC

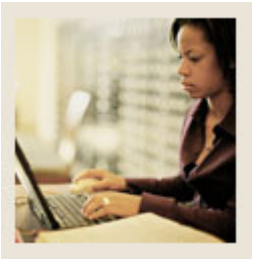
Separator

To retrieve all columns, specify all columns separated by commas, or use an asterisk (*).

```
SELECT * FROM <table name>;
```

Example

```
SQL> SELECT * FROM swriden;
```



Section C: Introduction to the Query

Lesson: Column Heading Aliases

Jump to TOC

Aliases

Aliases are used to substitute column headings in the SELECT statement. The default headings are in UPPERCASE and reflect the column name.

Example

```
SQL>     SELECT  DISTINCT swriden_pidm "ID Number",
                         swriden_last_name "Last Name",
                         swriden_first_name "First Name"
           FROM  swriden;

ID Number Last Name                 First Name
--------- ------------------------- ----------
    12340 Brown                     Julie
    12341 Smith                     Robert
    12342 Johnson                   Peter
    12343 Jones                     Sandy
    12343 Jones-Erickson            Sandy
    12344 Erickson                  Ralph
    12345 Erickson                  Susan
    12346 White                     Nancy
```

Multiple words

Enclosing the Heading Alias within double quotes allows the use of multiple-word headings. If the alias is one word, then the quotes are not necessary.

The default headings are in UPPERCASE and reflect the column name. An alias allows you to not only change the name but change the case of the heading.

Section C: Introduction to the Query

Lesson: Pseudo-columns

Jump to TOC

Pseudo-columns

A pseudo-column is a column that yields a value when selected, but is not actually a column in a table. Below are some frequently used pseudo-columns.

ROWNUM

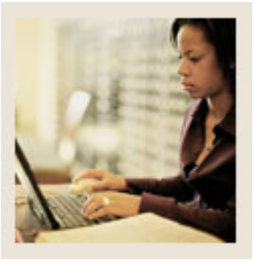
Returns a number indicating the sequence in which a row was selected from a table or set of joined rows.

```
SQL> SELECT ROWNUM, swriden_pidm, swriden_last_name
       FROM swriden;

ROWNUM    SWRIDEN_PIDM SWRIDEN_LAST_NAME
--------- ------------ -----------------
        1     12340    Brown
        2     12340    Brown
        3     12341    Smith
        4     12342    Johnson
      ...
```

ROWID

The ROWID is an internally generated and maintained binary value that identifies a row of data in a table. The information in ROWID provides the exact physical location of a row in the database. The return value of ROWID provides this value in a readable format.

```
SQL> SELECT rowid, swriden_last_name
  2    FROM swriden;

ROWID              SWRIDEN_LAST_NAME
------------------ -------------------------
AAAKIQAAGAAAabTAAA Brown
AAAKIQAAGAAAabTAAB Brown
AAAKIQAAGAAAabTAAC Smith
AAAKIQAAGAAAabTAAD Johnson
AAAKIQAAGAAAabTAAE Jones
AAAKIQAAGAAAabTAAF Jones-Erickson
AAAKIQAAGAAAabTAAG Erickson
AAAKIQAAGAAAabTAAH Erickson
AAAKIQAAGAAAabTAAI White
AAAKIQAAGAAAabTAAJ Marx
AAAKIQAAGAAAabTAAK Clifford
```



ROWID format

The ROWID format appears as follows:

OOOOOOFFFBBBBBBSSS

- OOOOOO The object number
- FFF The relative file number
- BBBBBB The block number in the file
- SSS The slot number within the block

Note: Prior to 8.x, the ROWID was something like a 16-digit string, with each digit being base-16. From 8.x on, the ROWID is something like a 20-digit string, with each digit being base-64.

SYSDATE

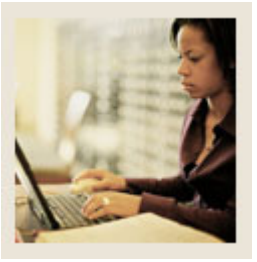
The current date and time.

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
01-APR-05
```

USER

The name of the current user.

```
SQL> SELECT USER FROM DUAL;

USER
---------
TRAIN01
```



Section C: Introduction to the Query

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

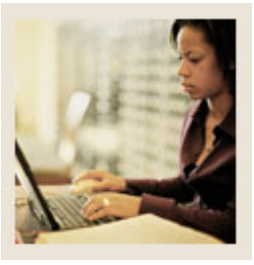
For the following exercises, use the Student subject table: **SWRREGS.** You may want to describe the table before performing the exercises.

Exercise 1

Write a query to return all the columns.

Exercise 2

Write a query to return the PIDM (Personal identification master), CRN (course number), and GPA (grade point average) for each record.



Section C: Introduction to the Query

Lesson: Self Check (Continued)

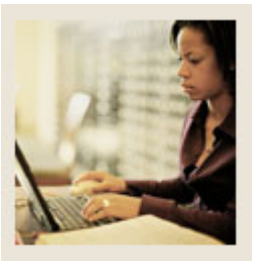
Jump to TOC

Exercise 3

Write a query to return the unique course numbers.

Exercise 4

Write a query to return the row number, row identification, and PIDM for each record.



Exercise 5

Select the system date from the dummy table DUAL.

Section D: Conditions and Operators

Lesson: Overview

Jump to TOC

Introduction

So far, we have discussed ways to query information from a particular table or database. In this section, we will discuss making queries more selective by only returning rows which meet certain criteria.

Objectives

At the end of this section, participants will be able to

- Specify all or specific rows based upon search criteria using:
  - comparison operators
  - logical operators
  - miscellaneous operators
- Use the pseudo-columns ROWNUM and ROWID to narrow searches
- Use parameters so that users are prompted for column or search information.

Section D: Conditions and Operators

Lesson: Overview (Continued)

Jump to TOC

Section contents

Section D: Conditions and Operators

Lesson: Conditions

Jump to TOC

What is a condition?

In SQL, a condition is a restriction on a query so that only rows which meet those conditions will be returned. Conditions are added through the use of the WHERE clause.

Section D: Conditions and Operators

Lesson: The WHERE Clause

Jump to TOC

Syntax

```
SELECT   select_list
  FROM   table_name
 WHERE   condition
```

WHERE Clause Components

- Column name or expression
- Comparison operator

Example

```
SQL> SELECT    swriden_last_name, swriden_first_name
       FROM    swriden
      WHERE    swriden_last_name  = 'Erickson';

SWRIDEN_LAST_NAME         SWRIDEN_FIRST_NAME
------------------------- -------------------
Erickson                  Ralph
Erickson                  Susan
```

Section D: Conditions and Operators

Lesson: Comparison Operators

Jump to TOC

Operators

Operators are used in WHERE clauses to compare values.

Operator	Function	Examples
( )	Overrides precedence	SELECT (X + Y) / (X - Y)
=	Test for equality	...WHERE last_name = ‘SMITH'
!=, ^=, <>	Test for inequality	...WHERE last_name <>‘SMITH'
>	Greater than	...WHERE sat_verbal > 450
>=	Greater than or equal to	...WHERE sat_verbal >= 450
<	Less than	...WHERE sat_math < 450
<=	Less than or equal to	...WHERE sat_math <= 450

Using ROWNUM

Take advantage of the pseudo-column ROWNUM to limit the number of rows returned. The ROWNUM is assigned to a row after it has evaluated the WHERE clause.

```
SQL> SELECT    ROWNUM, swriden_pidm, swriden_last_name
       FROM    swriden
      WHERE    ROWNUM < 5;

ROWNUM    SWRIDEN_PIDM SWRIDEN_LAST_NAME
--------- ------------ -----------------
        1     12340    Brown
        2     12340    Brown
        3     12341    Smith
        4     12342    Johnson
```

Using ROWID

Use the pseudo-column ROWID (the actual location of a row in a table) to distinguish between duplicate rows. Then, delete the extra row based on the ROWID.

```
SQL>  SELECT ROWID, swriden_pidm PIDM,
             swriden_last_name LAST_NAME,
             swriden_first_name FIRST_NAME
        FROM swriden
       WHERE swriden_change_ind IS NULL;

ROWID                  PIDM LAST_NAME           FIRST_NAME
------------------ ------- ------------------- --------------
AAAN0VAAJAAAAANAAA   12340 Brown                Julie
AAAN0VAAJAAAAANAAC   12341 Smith                Robert
AAAN0VAAJAAAAANAAD   12342 Johnson              Peter
AAAN0VAAJAAAAANAAF   12343 Jones-Erickson       Sandy
```

Section D: Conditions and Operators

Lesson: Logical Operators

Jump to TOC

Definition

Compound logical expressions are two or more expressions connected by logical operators.

Diagram 1

AND	true	false	null
True	true	false	null
False	false	false	false
Null	null	false	null

Example 1

```
SQL> SELECT    swraddr_stat_code, swraddr_zip, swraddr_pidm
       FROM    swraddr
      WHERE    swraddr_stat_code = 'PA'
        AND    swraddr_zip = '19380';

SWR SWRADDR_ZIP SWRADDR_P
--- ----------- ---------
PA  19380           12340
```

Diagram 2

OR	true	False	null
True	true	True	true
False	true	False	null
Null	true	Null	null

Example 2

```
SQL> SELECT     swriden_last_name, swriden_first_name, swriden_id
       FROM     swriden
      WHERE     swriden_last_name = 'Johnson'
         OR     swriden_id = '843853339';

SWRIDEN_LAST_NA SWRIDEN_FIRST_N SWRIDEN_I
--------------- --------------- ---------
Johnson         Peter           853954312
White           Nancy           843853339
Johnson         Jeremy          825110988
Johnson         Ian             861232200
```

Section D: Conditions and Operators

Lesson: BETWEEN Operator

Jump to TOC

BETWEEN operator

The BETWEEN condition is used to return rows containing values between two specified values (inclusive).

Example

```
SQL>  SELECT swrtest_pidm, swrtest_sat_verbal
        FROM swrtest
       WHERE swrtest_sat_verbal BETWEEN 520 AND 800;

SWRTEST_PIDM SWRTEST_SAT_VERBAL
------------ ------------------
    12340                   550
    12341                   530
    12342                   660
    12341                   590
    12343                   530
    12345                   590
    12346                   630
    12346                   520
    12346                   520
```

With logical operators

Keep in mind that logical operators may evaluate character data as well:

```
SQL>SELECT twvdetc_code DETC, twvdetc_desc
      FROM twvdetc
     WHERE twvdetc_code BETWEEN 'BOOK' AND 'CHEK';

DETC TWVDETC_DESC
---- -----------------
BOOK Book Charges
CHEK Check Payment
CASH Cash Payment
```

Section D: Conditions and Operators

Lesson: In Operator

Jump to TOC

IN operator

The IN operator is used in the WHERE clause to retrieve data which matches a value in the list provided.

Example

```
SQL> SELECT swriden_pidm, swriden_id, swriden_last_name,
            Swriden_first_name FIRST_NAME
       FROM swriden
      WHERE swriden_last_name IN ('Smith', 'Jones', 'Johnson');

      PIDM ID        LAST_NAME                 FIRST_NAME
---------- --------- ------------------------- ---------------
     12341 882993466 Smith                     Robert
     12342 853954312 Johnson                   Peter
     12343 845672112 Jones                     Sandy
     12352 825110988 Johnson                   Jeremy
     12353 861232200 Johnson                   Ian
```

Section D: Conditions and Operators

Lesson: LIKE Operator

Jump to TOC

LIKE operator

The LIKE condition is used to select information based on pattern matching. There can be more than one wildcard in a LIKE condition.

Wildcard characters

- % matches any number of characters
- _ matches a single character

Example 1

```
SQL> SELECT swvcrse_crn CRN, swvcrse_desc DESCRIPTION
       FROM swvcrse
      WHERE swvcrse_desc LIKE 'S%';

CRN      DESCRIPTION
------   --------------------
10012    Statistics
10015    Speech
10020    Swimming
```

Example 2

```
SQL> SELECT swvcrse_crn CRN, swvcrse_desc DESCRIPTION
       FROM swvcrse
      WHERE swvcrse_desc LIKE'%l_gy';

CRN      DESCRIPTION
-----    --------------------
10005    Biology
10006    Zoology
10008    Psychology
10011    Anthropology
```

Section D: Conditions and Operators

Lesson: NOT Operator

Jump to TOC

NOT operator

The NOT operator may be used to make a negative condition out of the following operators:

- NOT BETWEEN...AND...
- NOT IN (list)
- IS NOT NULL
- NOT LIKE

Example

```
SQL>SELECT     swriden_id ID, swriden_first_name FIRST_NAME,
               Swriden_last_name LAST_NAME, swriden_change_ind CHG
       FROM    swriden
      WHERE    swriden_change_ind IS NOT NULL;

ID         FIRST_NAME      LAST_NAME                 C
---------- --------------- ------------------------- -
876536782 Julie           Brown                     I
845672112 Sandy           Jones                     N
817253082 Michelle        Vaughn                    N
029348721 Frederick       Dukes                     I
029348721 Marcus          Jordan                    N
855231118 Tracy           Goode                     N
832736321 Devon           Roberson                  I
```

Section D: Conditions and Operators

Lesson: Precedence Rules

Jump to TOC

Expression evaluation

When a condition contains more than one expression, Oracle evaluates each expression according to the order of evaluation. The order of evaluation is determined by the precedence of the connecting operators.

Equal precedence

=, !=, >, >=, <, <=, IN, LIKE, IS NULL, BETWEEN...AND...

Logical operators

The logical operators are evaluated in this order:
1. NOT
2. AND
3. OR

Example 1 (Incorrect)

For example, suppose that you wish to retrieve data for a survey for all students who are married or have a birth date before '01-Jan-60'. Do not include confidential records.

```
  SQL> SELECT    *
         FROM    swbpers
        WHERE    swbpers_confid_ind = 'N'
          AND    swbpers_mrtl_code = 'M'
           OR    swbpers_birth_date < '01-JAN-60';

SWBPERS_PIDM SWBPERS_S SWBPERS_B S S S SWBPERS_A SWBPERS_USER SWBPERS_
------------ --------- --------- - - - --------- ------------ --------
       12341 682082678 12-NOV-70 M M N 10-DEC-05 TRAIN_ORA101 TRAINING
       12348 231560987 25-MAR-41 M F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12353 035341098 29-JUN-65 M M N 07-DEC-05 TRAIN_ORA101 TRAINING
       12357 430896512 27-NOV-52 M M Y 07-DEC-05 TRAIN_ORA101 TRAINING
       12359 318760932 31-DEC-54 D F Y 07-DEC-05 TRAIN_ORA101 TRAINING
```

What data are you actually retrieving?

Example 2 (correct)

The correct way:

```
  SQL> SELECT    *
         FROM    swbpers
        WHERE    swbpers_confid_ind = 'N'
          AND    (swbpers_mrtl_code = 'M'
           OR    swbpers_birth_date < '01-JAN-60');


SWBPERS_PIDM SWBPERS_S SWBPERS_B S S S SWBPERS_A SWBPERS_USER SWBPERS_
------------ --------- --------- - - - --------- ------------ --------
       12341 682082678 12-NOV-70 M M N 10-DEC-05 TRAIN_ORA101 TRAINING
       12348 231560987 25-MAR-41 M F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12353 035341098 29-JUN-65 M M N 07-DEC-05 TRAIN_ORA101 TRAINING
```

Section D: Conditions and Operators

Lesson: Parameters

Jump to TOC

Ampersand prompt

Use the ampersand (&) to prompt users for parameters. When SQL*Plus encounters an ampersand variable, the user is prompted for input.

Enter substitute variables anywhere in a SQL statement command except in the first word entered at the prompt.

Example 1

```
SQL> SELECT *
       FROM swvcrse
      WHERE swvcrse_crn = '&Course_Num';
```

Enter value for course_num: 10021

```
SWVCRSE_CRN    SWVCRSE_DESC              SWVCRSE_A
-----------    ----------------------    ---------
10021          Economics                 17-APR-05
```

Example 2

```
SQL> SELECT *
       FROM &select_table;
```

Enter value for table: SWRREGS

```
SWRREGS_ SWRREGS_C  SWRREGS_G SWRRE SWRREGS_A
-------- ---------  --------- ----- ---------
12340    200501     10001     3.2   17-APR-05
12340    200501     10007     2.1   17-APR-05
12340    200501     10015     2.8   17-APR-05
12340    200501     10017           17-APR-05
  ...
```

Section D: Conditions and Operators

Lesson: Parameters (Continued)

Jump to TOC

Parameters Data Types

Parameter data must be entered in the data type that matches the column data type. For example:

- Character – enclose in single quotes
- Date – enclose in single quotes – enter in Oracle default format ('dd-Mon-yyyy' or 'dd-Mon-yy')
- Number – no quotes

```
SQL> SELECT *
       FROM swbpers
      WHERE swbpers_birth_date > &birth_date;

   Enter value for birth_date: '01-Jan-1980'
```

Single quotes can be included in the SQL statement to avoid having to guess whether a prompt requires quotes or not.

```
SQL> SELECT *
       FROM swbpers
      WHERE swbpers_birth_date > '&birth_date';

   Enter value for birth_date: 01-Jan-1980
```

Double ampersand

Avoid repetitious variable entry for the entire session by using the double ampersand (&&). It will prompt you the first time for a value, and each subsequent use of the same variable (&&variable) will use the value you entered.

```
SQL> SELECT &&my_numb, &&my_numb + 10,
            &&my_numb + 20
       FROM DUAL;
```

Enter value for my_numb: 1

```
1        1+10       1+20
-------  --------   -------
1        11         21
```

Undefining double ampersand variables

To remove the value of a double ampersand variable, use the undefine command.

```
SQL> undefine my_numb
```

Once you undefined a variable, if you use it again with the double ampersands, it will prompt you for a value (once after each undefine).

Section D: Conditions and Operators

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

Exercise 1

Query the first five rows from the **SWRREGS** table.

Try to query rows 3 and higher from **SWRREGS**. What occurred?

Exercise 2

Using **SWRADDR**, find the city, state, and zip code for PIDM (internal identification master) 12340. Remember to describe the table first to see the names of the fields.

Exercise 3

From **SWRIDEN**, query the students who do not have the last name of 'Erickson'. Remember to only include the most current record for each student. Return all columns.

Exercise 4

Use the single ampersand in a SQL statement to prompt for a table name, displaying all columns within a table. Run this query for **SWRREGS** and **SWRIDEN.**

Exercise 5

Retrieve the first name, last name, and ID from **SWRIDEN** where the ID begins with a 0. (Hint: use the keyword LIKE)

Retrieve the first name, last name, and ID from **SWRIDEN** where the fourth character in the ID field is a 6 and the total length of the column is 7.

Exercise 6

Using the **SWRREGS** table, list the PIDM, CRN, and GPA for students who have taken golf, tennis, or swimming. (Hint: look at the validation table SWVCRSE first to find the code values for the courses.)

Section D: Conditions and Operators

Lesson: Self Check (Continued)

Jump to TOC

Note

For exercises 7 - 9, refer to the SAT test table, **SWRTEST.**

Exercise 7

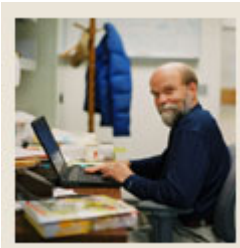
Retrieve the student records where the student achieved above 550 in both math and verbal.

Exercise 8

Retrieve the student records where the student achieved above 550 in either math or verbal.

Exercise 9

Retrieve the student records where the student took the SAT between '01-JAN-06' and '31-MAY-06' (if the two-digit year does not work, try entering the four-digit year).



Section E: Arithmetic Expressions and Functions

Lesson: Overview

Jump to TOC

Introduction

Oracle supports arithmetic expressions in SQL statements to perform various calculations on numbers:

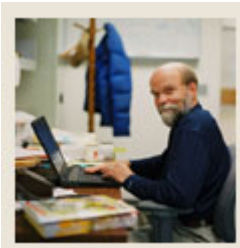
- + Add
- - Subtract
- * Multiply
- / Divide
- ( ) Override Precedence

In addition, Oracle supports a wide range of built-in functions to manipulate data.

Objectives

At the end of this section, participants will be able to

- Use arithmetic expressions to perform calculations on data
- Use the following within SQL statements:
  - Numeric functions
  - Character functions
  - Date functions
  - Conversion functions
  - Group functions.



Section E: Arithmetic Expressions and Functions

Lesson: Overview (Continued)

Jump to TOC

Section contents

Section E: Arithmetic Expressions and Functions

Lesson: Arithmetic Expressions

Jump to TOC

Purpose

Arithmetic expressions allow you to perform calculations on data.

Example 1

```
SQL> SELECT swrtest_sat_verbal SAT_ORIG,
            swrtest_sat_verbal + 100 SAT_MOD
       FROM swrtest;

SAT_ORIG       SAT_MOD
----------     --------------
550            650
530            630
660            760
590            690
530            630
370            470
```

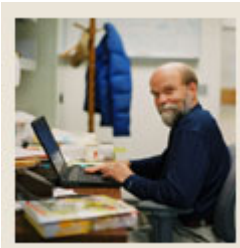
…

Example 2

```
SQL>  SELECT   swrtest_sat_verbal VERBAL, swrtest_sat_math MATH,
               swrtest_sat_verbal + swrtest_sat_math TOTAL
        FROM   swrtest;

    VERBAL       MATH      TOTAL
---------- ---------- ----------
       550        480       1030
       530        580       1110
       660        520       1180
       590        610       1200
       530        420        950
       590        620       1210
       630        590       1220
       520        460        980
                  550
       500
```

Other operators

Similarly, you can use the subtraction (-), division (/), and multiplication (*) arithmetic operators.



Section E: Arithmetic Expressions and Functions

Lesson: Order of Evaluation

Jump to TOC

Expression evaluation

The Relational Database Management System evaluates each arithmetic expression. The results are then combined in the order determined by the following precedence:

- * Multiplication
- / Division
- + Addition
- - Subtraction

Overriding precedence

To override the precedence rules, use parentheses. Oracle evaluates expressions within parentheses first.

Examples

The following examples retrieve the balance from the student account balance table. According to the precedence rules, division will occur before addition, so the following examples yield different results.

Example 1

In the first example, 100 is divided by 12 and then added to the balance.

```
SQL> SELECT    twraccd_balance BAL_ORIG,
               twraccd_balance + 100 /12 “BAL_MOD”
         FROM  twraccd;

BAL_ORIG       BAL_MOD
---------      --------------
1500.5         1508.8333
 300.2         308.53333
  -700         -691.6667
  1100         1108.3333
   500         508.33333
 -1000         -991.6667
  1200         1208.3333
    50         58.333333
   800         808.33333
   800         808.33333
 -1100         -1091.667
...
```

Example 2

In the second example, 100 is added to the balance, and then the total is divided by 12.

```
 SQL> SELECT twraccd_balance BAL_ORIG,
            (twraccd_balance + 100)/12 “BAL_MOD”
          FROM twraccd;

BAL_ORIG  BAL_MOD
--------- --------------
   1500.5        133.375
    300.2          33.35
     -700            -50
     1100            100
      500             50
    -1000            -75
     1200      108.33333
       50           12.5
      800             75
      800             75
    -1100      -83.33333
    ...
```

ABS(n)

Returns the absolute value of *n*.

```
SQL> SELECT ABS(-32) FROM DUAL;

 ABS(-32)
---------
       32
```

CEIL(n)

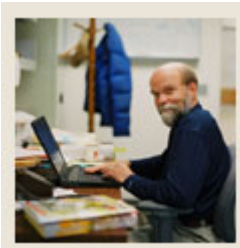
Returns smallest integer greater than or equal to *n*.

```
SQL> SELECT CEIL(12.8) FROM DUAL;
CEIL(12.8)
----------
        13

SQL> SELECT CEIL(-16.2) FROM DUAL;
CEIL(-16.2)
----------
       -16
```

FLOOR(n)

Returns largest integer equal to or less than *n*.

```
SQL> SELECT FLOOR(12.8) FROM DUAL;

FLOOR(12.8)
-----------
         12

SQL> SELECT FLOOR(-17.5) FROM DUAL;

FLOOR(-17.5)
------------
         -18
```



MOD(m, n)

Returns remainder of m divided by *n*. If *n* is negative and greater than *m*, *m* is returned.

```
SQL> SELECT MOD(5,2) FROM DUAL;

 MOD(5,2)
---------
        1
```

POWER(m, n)

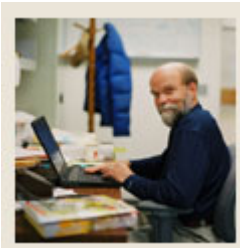
Returns *m* raised to the *n*th power. *n* must be an integer; if not, an error will be returned.

```
SQL> SELECT POWER(10,2) FROM DUAL;

POWER(10,2)
-----------
        100

SQL> SELECT POWER(-10,3) FROM DUAL;

POWER(-10,3)
------------
       -1000
```

ROUND(n [,m])

Returns *n* rounded to *m* places right of the decimal point; if *m* is omitted, then *n* is rounded to the nearest whole number.

```
SQL> SELECT ROUND(15.67,1) FROM DUAL;

ROUND(15.67,1)
--------------
          15.7

SQL> SELECT ROUND(152,-1) FROM DUAL;

ROUND(152,-1)
-------------
          150
```



SIGN(n)

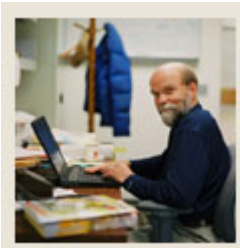
If *n* < 0, the function returns -1; if n = 0, the function returns 0; if n > 0, then the function returns 1.

```
SQL> SELECT SIGN(-15) FROM DUAL;

SIGN(-15)
---------
       -1

SQL> SELECT SIGN(15) FROM DUAL;

 SIGN(15)
---------
        1

SQL> SELECT SIGN(0) FROM DUAL;

  SIGN(0)
---------
        0
```

TRUNC(n [,m])

Returns *n* truncated to *m* decimal places. If *m* is omitted, then the decimal is removed completely. *m* can be negative to truncate *m* digits left of the decimal point.

```
SQL> SELECT TRUNC(16.99,1) FROM DUAL;

TRUNC(16.99,1)
--------------
          16.9

SQL> SELECT TRUNC(16.99,-1) FROM DUAL;

TRUNC(16.99,-1)
---------------
             10
```



ASCII(char)

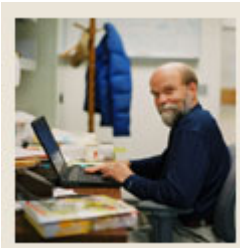
Returns the ASCII value for the given character.

```
SQL> SELECT ASCII('A') FROM DUAL;

ASCII('A')
----------
        65
```

CHR(n)

Returns the character having ASCII value *n*.

```
SQL> SELECT CHR(65) FROM DUAL;

C
-
A
```



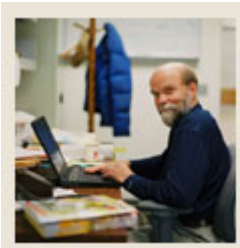
CONCAT(m, n) or ||' '||

Merges together two fields specified by *m* and *n*.

```
SQL> SELECT CONCAT(swriden_first_name, swriden_last_name) "Name"
       FROM swriden;
Name
----------------------------------------
JulieBrown
JulieBrown
RobertSmith
PeterJohnson
SandyJones
...
```

To merge two or more fields, use the pipes:

```
SQL>  SELECT   swriden_last_name||', '|| swriden_first_name "Name"
        FROM   swriden;

Name
----------------------------------------
Brown, Julie
Brown, Julie
Smith, Robert
Johnson, Peter
Jones, Sandy
...
```



INITCAP(char)

Returns *char* with the first letter of each word in uppercase and all other letters in lowercase.

```
SQL> SELECT INITCAP('BASEBALL GAME') FROM DUAL;

INITCAP('BASE
-------------
Baseball Game
```

LOWER(char)

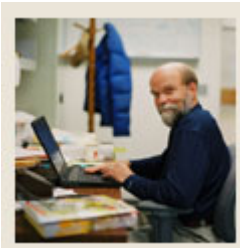
Returns *char*, with all letters forced to lowercase.

```
SQL> SELECT LOWER('BASEBALL GAME')
     FROM DUAL;

LOWER('B
-------------
baseball game
```

UPPER(char)

Returns char, with all letters forced to uppercase.

```
SQL> SELECT UPPER('big letters') "Upper Case"
       FROM DUAL;

Upper Case
-----------
BIG LETTERS


SQL> SELECT swriden_last_name LAST_NAME,
            swriden_first_name FIRST_NAME
       FROM swriden
      WHERE UPPER(swriden_last_name) = 'WHITE';

LAST_NAME                 FIRST_NAME
------------------------- ---------------
White                     Nancy
```



LPAD(char1, n [,char2])

Returns char1, left-padded to length *n* with the sequence of characters in char2; char2 defaults to blanks.

```
SQL> SELECT LPAD(swriden_pidm,8,0) "PIDM" FROM swriden;

PIDM
--------
00012340
00012340
00012341
00012342
...
```

RPAD(char1, n [,char2])

Returns char1, right-padded to length *n* with sequence of characters in char2; if char2 is omitted, right-pad with blanks.

```
SQL> SELECT RPAD(swriden_last_name,20,'.')|| swriden_id "ID Listing"
       FROM swriden
      WHERE swriden_change_ind IS NULL;

ID Listing
----------------------------
Brown...............857834585
Smith...............882993466
Johnson.............853954312
Jones-Erickson......845672112
Erickson............892568211
Erickson............878549991
White...............843853339
Marx................822874301
Clifford............862100933
Serum...............881337923
Dukes...............817253082
Dukes...............872109834
Johnson.............825110988
Johnson.............861232200
Bristow.............831603288
McNair..............855231118
Miner...............832092865
Roberson............832763321
Peterson............853084511
Jameson.............8486565
```

LTRIM(char [, set])

Removes characters from the left of char, with initial characters removed up to the first character not in the set; set defaults to ' ', which removes extra blanks.

```
SQL> SELECT LTRIM('123ABC123','123')
       FROM DUAL;

LTRIM(
------
ABC123
```

RTRIM(char [, set])

Removes characters from the right of char, with initial characters removed up to the first character not in the set; defaults to ' ', which removes extra blanks.

```
SQL> SELECT RTRIM('123ABC123','123') FROM DUAL;

RTRIM(
------
123ABC
```

SUBSTR(char, m [,n])

Returns a portion of *char*, beginning at character *m*, *n* characters long (if *n* is omitted, to the end of *char*).

```
SQL> SELECT    SUBSTR('BANNER University',1, 6) "Substring"
       FROM    DUAL;

Substring
-------------------------
BANNER

SQL> SELECT    SUBSTR('BANNER University',8) "Substring"
       FROM    DUAL;

Substring
-------------------------
University
```

LENGTH(char)

Returns the length of char.

```
SQL> SELECT LENGTH('abc') FROM DUAL;

LENGTH('ABC')
-------------
            3
```

Note: If the evaluated field is null, then no value (null) will be returned.

SOUNDEX (char)

Returns character string containing the phonetic representation of char.

```
SQL> SELECT swriden_last_name LAST_NAME
       FROM swriden
      WHERE SOUNDEX(swriden_last_name) = SOUNDEX('SMYTHE');

LAST_NAME
-------------------------
Smith
```

Section E: Arithmetic Expressions and Functions

Lesson: Regular Expressions

Jump to TOC

Regular Expressions (10g only)

In Oracle 10g functionality was added to do regular expression searching. Regular expressions allow you to search and manipulate text strings using complex patterns. There are four functions that were introduced to support regular expressions.

REGEXP_LIKE

REGEXP_LIKE (*source_string*, *pattern* [, *match_parameter*])

The *source_string* is a character-type column or literal.
The *pattern* is the regular expression string of characters you are trying to match.
The *match_parameter* are optional parameters that aid in searching:

i – specifies case insensitive matching
c – specifies case sensitive matching
n – specifies that a period (.) should match a new line character
m – specifies that the string is multiple lines

Example

Do a case insensitive search, selecting first and last names where the first name starts with a vowel.

```
SELECT swriden_first_name, swriden_last_name
  FROM swriden
  WHERE regexp_like(swriden_first_name,'^[aeiou]','i');

SWRIDEN_FIRST_N SWRIDEN_LAST_NAME
--------------- -------------------------
Ian             Johnson
Amy             Peterson
```

REGEXP_INSTR

REGEXP_INSTR(*source_string*, *pattern*,
[, *position [*, *occurrence*
[, *return_option* [, *match_parameter*]]])

The *return_option* of 0 (default) returns the position of the first character of the occurrence. A 1 returns the position of the character following the occurrence.

Section E: Arithmetic Expressions and Functions

Lesson: Regular Expressions (Continued)

Jump to TOC

REGEXP_INSTR (continued)

Example

Select first and last name and position of second vowel in first name, case insensitive, where the last name is at least 6 characters long. Do not include rows where the first name does not have 2 vowels.

```
SQL>SELECT swriden_first_name, swriden_last_name,
           REGEXP_INSTR(swriden_first_name, '[aeiou]',1,2,0,'i')
   secnd_vowel
      FROM swriden
     WHERE REGEXP_LIKE (swriden_last_name,'[[:alpha:]]{6,}','i')
       AND REGEXP_INSTR(swriden_first_name, '[aeiou]',1,2,0,'i') > 0;

   SWRIDEN_FIRST_N SWRIDEN_LAST_NAME         SECND_VOWEL
   --------------- ------------------------- -----------
   Peter           Johnson                             4
   Susan           Erickson                            4
   Stephanie       Clifford                            6
   Michelle        Vaughn                              5
   Marcus          Jordan                              5
   Jeremy          Johnson                             4
   Ian             Johnson                             2
   Brandon         Bristow                             6
   Devon           Roberson                            4
   Devon           Roberson                            4
   Jennifer        Jameson                             5
```

REGEXP_REPLACE

REGEXP_REPLACE (*source_string*, *pattern* [, *position* [, *occurrence* [, *match_parameter*]]])

Example

Add a space between each character.

```
SQL> SELECT REGEXP_REPLACE('Banner Univ.', '(.)', '\1 ') more_space
  2    FROM dual;

MORE_SPACE
------------------------
B a n n e r   U n i v .
```

REGEXP_SUBSTR

REGEXP_SUBSTR (*source_string, pattern* [, *position* [, *occurrence* [, *match_parameter*]]])

Example

For those who have PO Box numbers as their address, extract the box number.

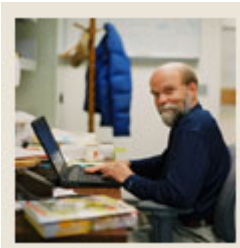
```
SELECT REGEXP_SUBSTR(swraddr_street_line1,'.*',
       REGEXP_INSTR(swraddr_street_line1, 'PO Box',1,1,1,'i') +1
       ,1,'i') box_nbr
  FROM swraddr
 WHERE swraddr_street_line1 like 'PO%';

BOX_NBR
------------------------------
1035
```

Date format element models

Date format element models for TO_CHAR and TO_DATE

This table lists the date format elements. You can use any combination of these elements as the *fmt* argument of the TO_CHAR and TO_DATE functions. *Fmt* defaults to the default DATE format, 'DD-MON-YY'.

Format element	Value returned
SCC or CC	Century; 'S' prefixes BC date with '-'
YYYY or SYYYY	Year; 'S' prefixes BC date with '-'
YYY or YY or Y	Last 3, 2, or 1 digit(s) of year. Century defaults to current
RR	Last 2 digits of year. 50 - 99 = 20th century, 00 - 49 =21st century
BC or AD	BC/AD indicator
B.C. or A.D.	BC/AD indicators with periods
Q	Quarter of year (1, 2, 3, 4)
MM	Month of year (01 - 12)
RM	Roman Numeral month (I…XII)
MONTH	Name of month padded with blanks to nine characters
MON	Name of month abbreviated (JAN, FEB, etc.)
WW or W	Week of year (1 - 52) or month(1 - 5)
DDD or DD or D	Day of year (1-366) or month (1 - 31) or week (1 - 7)
DAY	Name of day, blank-padded to 9 characters
DY	Name of day, 3 letter abbreviation
J	Julian day (days since December 31, 4713 BC)
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12	Hour of day (1 - 12)
HH24	Hour of day (1 - 23)
MI	Minute (0 - 59)
SS or SSSSS	Seconds (0 - 59) or seconds past midnight (0 - 86399)
- / , . ; :	Punctuation is reproduced in the result



Date format element models (cont.)

Format element	Value returned
"…text…"	Quoted string is reproduced in the result
Date format prefixes and suffixes	
FM	"Fill mode." Suppresses blank padding when prefixed to MONTH or DAY.
FX	"Format exact." Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model in the TO_DATE function.

Suffixes

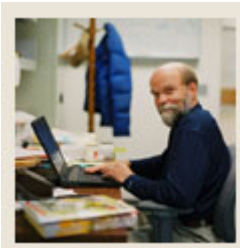
You can add these suffixes to date format elements:

- **TH** Ordinal number ("DDTH" for "4TH")
- **SP** Spelled-out number ("DDSP" for "FOUR")
- **SPTH** and **THSP** Spelled-out ordinal number ("DDSPTH" for "FOURTH")

Date format case control

The following strings specify output in uppercase, initial caps, or lower case.

Uppercase	Initial caps	Lower case
DAY	Day	day
DY	Dy	dy
MONTH	Month	month
YEAR	Year	year
AM	Am	am
PM	Pm	pm
A.M.	A.m.	a.m.
P.M.	P.m.	p.m.



Section E: Arithmetic Expressions and Functions

Lesson: Format Models (Continued)

Jump to TOC

Number format models

Number format models for TO_CHAR

Format element	Example	Function
9	'9999'	Number of "9"s determines length of returned character
0	'0999'	Prefixes value with leading or trailing zeroes
$	'$9999'	Prefixes value with a dollar sign
B	'B9999'	Returns zero values as blanks, instead of "0"
MI	'9999MI	Returns "-" after negative values
S	'S9999'	Returns "+" for positive values and "-" for negative values
PR	'999PR'	Returns negative values in <angle brackets>
D	99D99	Returns the decimal character
G	9G999	Returns the group separator
L	L999	Returns the local currency symbol
C	C999	Returns the international currency symbol
,	'9,999'	Returns comma in this position
.	'99.99'	Returns period in this position
V	'999V99'	Multiplies value by 10^n where n is "9"s after V
EEEE	'9.99EEEE'	Returns value in scientific notation
RM or rn	RN	Upper or lowercase Roman numerals
DATE	'DATE'	Returns value converted from Julian date

ADD_MONTHS(d, n)

Returns the date *d* plus *n* months. Can be negative to subtract months.

```
SQL> SELECT ADD_MONTHS('10-DEC-05',2) FROM DUAL;

ADD_MONTH
---------
10-FEB-06
```

LAST_DAY(d)

Returns the date of the last day of the month that contains *d*.

```
SQL> SELECT LAST_DAY('15-JAN-05') FROM DUAL;

LAST_DAY(
---------
31-JAN-05
```

MONTHS_ BETWEEN (d, e)

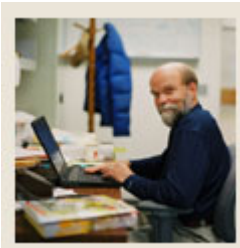
Returns the number of months between dates *d* and *e*. If *d* is less than *e*, then the result is negative.

```
SQL> SELECT MONTHS_BETWEEN('01-JAN-06', '11-OCT-05') "Months"
     FROM DUAL;

    Months
----------
2.67741935
```

NEXT_DAY (d, char)

Returns the date of the first day of the week named by *char* that is later than *d*.

```
SQL> SELECT    NEXT_DAY('05-FEB-05','FRIDAY')
                 "Pay Date"
       FROM    DUAL;

Pay Date
---------
11-FEB-05
```



TRUNC (d, [,fmt])

Returns *d* with the time portion of the date truncated to the unit specified by the format model *fmt*.

```
SQL> SELECT    TRUNC(TO_DATE('05-OCT-05'),'YEAR') "Date"
       FROM    DUAL;

Date
---------
01-JAN-05
```

ROUND (d [,fmt])

Returns *d* with the time portion of the date rounded to the unit specified by the format model *fmt*.

NOTE on ROUND: Days past the middle of a month round up to the next month, months July to Decenber round up to the next year, and dates stored with time after 12:00PM(noon) will round to the next day.

```
SQL> SELECT    ROUND(TO_DATE('05-OCT-05'),'YEAR') "Date"
       FROM    DUAL;

Date
---------
01-JAN-06
```

TRANSLATE (char, from, to)

Returns *char* with all occurrences of each character in *from* replaced by its corresponding character in *to*.

```
SQL> SELECT TRANSLATE('PERRY','P','J') FROM DUAL;

TRANS
-----
JERRY

SQL> SELECT    TRANSLATE('ABC123ABC123','B3','X0') "Translate"
       FROM    DUAL;

Translate
-----------
AXC120AXC120
```

NVL (expr1, expr2)

Fields in a database which contain NULL values actually contain no value. Because NULL in a field can give unexpected results, it is best to convert the NULL into another value such as a zero for a number field or a space for a character field.

```
SQL> SELECT   NVL(swrtest_sat_math,0) MATH,
              NVL(swrtest_sat_verbal,0) VERBAL,
              NVL(swrtest_sat_math,0) + NVL(swrtest_sat_verbal,0)
              "Tot Score",
              swrtest_sat_math + swrtest_sat_verbal "Tot No NVL"
       FROM    swrtest;

      MATH     VERBAL  Tot Score Tot No NVL
---------- ---------- ---------- ----------
       480        550       1030       1030
       580        530       1110       1110
       520        660       1180       1180
       610        590       1200       1200
       420        530        950        950
       620        590       1210       1210
       590        630       1220       1220
       460        520        980        980
       550          0        550
         0        500        500
```

COALESCE (expr1, expr2, expr3…)

Returns the first argument (beginning on the left) that is NOT NULL. It is similar to the NVL function, but it can take multiple alternate values. It accepts any number of arguments.

```
  SELECT swrtest_sat_math, swrtest_sat_verbal,
         COALESCE(swrtest_sat_math,
                  swrtest_sat_verbal,0) "COALESCE RESULT"
         FROM swrtest;

SWRTEST_SAT_MATH SWRTEST_SAT_VERBAL COALESCE RESULT
---------------- ------------------ ---------------
             480                550             480
             580                530             580
             520                660             520
             610                590             610
             420                530             420
             620                590             620
             590                630             590
             460                520             460
             550                                550
                                500             500
                                                  0
```

NULLIF(expr1, expr2)

If expr1 = expr2 then return null, else return expr1.

```
  SELECT swrtest_sat_math, swrtest_sat_verbal,
         NULLIF(swrtest_sat_verbal,530)
    FROM swrtest;

SWRTEST_SAT_MATH SWRTEST_SAT_VERBAL NULLIF(SWRTEST_SAT_VERBAL,530)
---------------- ------------------ -------------------------------
             480                550                             550
             580                530
             520                660                             660
             420                530
             620                590                             590
             550
                                500                             500
```

Section E: Arithmetic Expressions and Functions

Lesson: Conversion Functions (Continued)

Jump to TOC

DECODE(expr, search1, result1, [search2, result2,]... [default])

The DECODE function converts the retrieved value from the database (search value) of the expression to a value specified (result). If no match is found, the DECODE function returns the default value. If no default value is specified, then the default will return a null.

```
SQL> SELECT    swbpers_pidm,
     DECODE    (swbpers_sex, 'F', 'Female', 'M', 'Male',
               'Unknown') "Gender"
       FROM    swbpers;

PIDM     Gender
-------  -------
12340    Female
12341    Male
12343    Female
12344    Male
12345    Unknown
```

DECODE can be used to perform a calculation. Consider a table which stores an amount as an absolute value and an indicator to specify if the refund is a negative quantity.

```
SQL> SELECT    DECODE(refund, 'Y', amount * -1,
               amount)
       FROM    ...
```

It's easy to think of a DECODE as a simple If-Then-Else statement:

DECODE(expr, search1, result1, search2, result2,…default) can be thought of as:

If expr = search 1 then result1
Else if expr = search2 then result2
Else if expr = search3 then result3
…
Else default

Section E: Arithmetic Expressions and Functions

Lesson: Conversion Functions (Continued)

Jump to TOC

CASE

There are two types of case statements – Simple and Searched.

Simple CASE

The simple CASE is similar to DECODE. It can be used to search and replace values within a given expression. You can specify a return value for each searched value. No comparison operators can be used in the simple CASE.

This decode statement can be re-written as the simple CASE statement below:

```
SELECT swbpers_pidm,
       decode(swbpers_mrtl_code,'S','Swinging Single',
                                'M','Happily Married',
                                'W','Widowed',
                                'D','Divorced',
                                'Unknown')
  FROM swbpers;
```

Same statement as a case statement:

```
SELECT swbpers_pidm,
       CASE swbpers_mrtl_code
            WHEN 'S' THEN 'Swinging Single'
            WHEN 'M' THEN 'Happily Married'
            WHEN 'W' THEN 'Widowed'
            WHEN 'D' THEN 'Divorced'
            ELSE 'Unknown'
       END
  FROM swbpers;
```

```
Both statements return:

      SWBPERS_PIDM DECODE(SWBPERS_
      ------------ ---------------
             12340 Swinging Single
             12341 Happily Married
             12343 Swinging Single
             12344 Swinging Single
             12345 Widowed
             12346 Divorced
               ….
```

CASE (continued)

Searched Case

The searched case allows for test conditions other than equality.

```
SELECT swrtest_pidm,
       CASE
       WHEN swrtest_sat_math < 300 THEN 'Unacceptable'
       WHEN swrtest_sat_math between 301 and 500 THEN 'Needs Work'
       WHEN swrtest_sat_math > 500 THEN 'Excellent'
       ELSE 'Unknown'
       END
    FROM swrtest

SWRTEST_PIDM CASEWHENSWRT
------------ ------------
       12340 Needs Work
       12341 Excellent
       12342 Excellent
       12341 Excellent
       12343 Needs Work
       12345 Excellent
       12346 Excellent
       12346 Needs Work
       12347 Excellent
       12345 Unknown
       12344 Unknown
```

INSTR (char1, char2 [, n [,m]])

Searches *char1* beginning with its *n*th character for the *m*th occurrence of *char2* and returns the position of the character in *char1* that is the first character of this occurrence.

```
SQL> SELECT INSTR('MISSISSIPPI','SS') "STRING"
       FROM DUAL;

   STRING
---------
        3
```

Both of the following produce the same result:

```
SQL> SELECT INSTR('MISSISSIPPI','SS',5) "STRING"
       FROM DUAL;

SQL> SELECT INSTR('MISSISSIPPI','SS',1,2) "STRING"
       FROM DUAL;

   STRING
---------
        6
```

INSTR and character strings

Use INSTR to parse a character string. In the view swvtele, the name column is created by the concatenation of last_name, a comma, a space, first name, a space, and mi (middle initial). The name column in this view would be parsed as follows:

```
SQL> SELECT swvtele_name,
            INSTR(swvtele_name,',') "Comma"
       FROM swvtele;

SWVTELE_NAME                                 Comma
---------------------------------------- ----------
Brown, Julie                                      6
Smith, Robert                                     6
Johnson, Peter                                    8
Jones-Erickson, Sandy                            15
```

REPLACE (char, search_string [, replace_string])

Returns *char* with every occurrence of *search_string* replaced with *replacement_string*. If *replacement_string* is omitted, all occurrences of *search_string* are removed.

```
SQL> SELECT    REPLACE('100 Lakeshore Drive # 3',
               '#', 'No.') "Changes"
       FROM    DUAL;

Changes
-------------------------
100 Lakeshore Drive No. 3
```

TO_DATE (char [,fmt])

Converts a character value to a date value. If the *fmt* clause is omitted, then the character value must have the default format of 'DD-MON-YY'. When the format is not in the default form, you must explicitly tell Oracle the format.

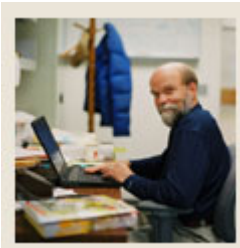
```
SQL>  SELECT to_date('12/31/2005','mm/dd/yyyy')
        FROM dual;

TO_DATE('
---------
31-DEC-05
```

TO_CHAR (d, [,fmt])

Converts number and dates into a character.

```
SQL> SELECT    TO_CHAR(SYSDATE,'DD-MON-YYYY') "DATE"
       FROM    DUAL;

DATE
---------------------------------
20-MAY-2005


SQL> SELECT    TO_CHAR(SYSDATE,'FMMonth DD, YYYY') "DATE"
       FROM    DUAL;

DATE
------------
May 20, 2005


SQL> SELECT    TO_CHAR(SYSDATE,'DD-MON-YY HH24:HH:SS') "DATE"
       FROM    DUAL;

DATE
------------------
20-MAY-05 14:02:22


SQL> SELECT    TO_CHAR(SYSDATE,'FMDay, FMMonth
                 FMDdSPTh, Syear') "DATE"
       FROM    DUAL;

DATE
--------------------------------------------------
Tuesday, May Twentieth, Nineteen Ninety-Seven
```

Note: Dates that have been converted using TO_CHAR are no longer sorted by date, but by alphanumeric order. If you are forced to use a converted date in a GROUP BY clause, it is best to convert the date to the YYYYMMDDHH24MISS format.



RR: Changing Millennia

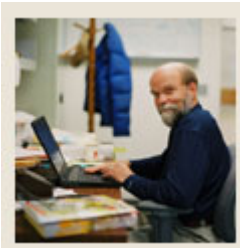
With the arrival of the 21st century, the need to refer to both the 20th and 21st centuries is necessary. The RR date format will change the pivot so that two-digit years from 50 - 99 refer to the 20th century; years from 00 - 49 refer to the 21st century. If the date is retrieved from a field where the date was inserted as a four-digit year, then changing the pivot year is not necessary.

```
SQL> SELECT to_date('01-Jan-70','dd-Mon-RR') "DATE"
       FROM dual;

DATE
------------------
01-Jan-1970


SQL> SELECT to_date('01-Jan-70','dd-Mon-YY') "DATE"
       FROM dual;

DATE
------------------
01-Jan-2070
```

TO_NUMBER (char)

Converts *char* to a numeric data type.

```
SQL> SELECT    TO_NUMBER('001') + TO_NUMBER('002') "TOTAL"
       FROM    DUAL;

TOTAL
---------
3
```



Section E: Arithmetic Expressions and Functions

Lesson: Group Functions

Jump to TOC

Group functions

Group functions are used to obtain summary information about groups of rows. All group functions, except COUNT (*), ignore null values. Use the NVL function in the argument to substitute a value of null in a group function.

AVG(n)

Returns the average value of *n*.

```
SQL> SELECT AVG(swrregs_gpa) FROM swrregs;

AVG(SWRREGS_GPA)
----------------
      2.86216216

SQL> SELECT AVG(NVL(swrregs_gpa,0)) FROM swrregs;

AVG(NVL(SWRREGS_GPA,0))
-----------------------
             2.40681818
```

COUNT ({ * | expr })

Returns the number of rows in a query. If specifying a count on a particular column, the null values are not included.

```
SQL> SELECT COUNT(*) FROM swrregs;

 COUNT(*)
---------
       44

SQL> SELECT COUNT(swrregs_gpa) FROM swrregs;

COUNT(SWRREGS_GPA)
------------------
                37
```

MAX(expr)

Returns the maximum value of expr.

```
SQL> SELECT MAX(swrregs_gpa) FROM swrregs;

MAX(SWRREGS_GPA)
----------------
               4
```

MIN(expr)

Returns the minimum value of expr.

```
SQL> SELECT MIN(swrregs_gpa) FROM swrregs;

 MIN(SWRREGS_GPA)
-----------------
                0
```

SUM(n)

Returns sum of values of *n*.

```
SQL> SELECT SUM(swrregs_gpa) FROM SWRREGS;

 SUM(SWRREGS_GPA)
-----------------
            49.5
```

Section E: Arithmetic Expressions and Functions

Lesson: Self Check

Jump to TOC

Directions

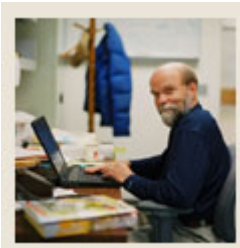
Use the information you have learned in this workbook to complete this self check activity.

Exercise 1

In the **SWRREGS** table, what is the average GPA of all the classes that PIDM 12342 took?

Exercise 2

How many records does PIDM 12343 have in the **SWRIDEN** table?

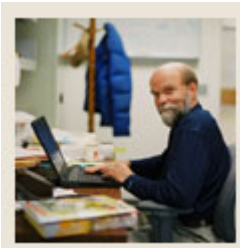


Exercise 3

Select the PIDM and the combined score of the SAT verbal and math for each record from the **SWRTEST** table.

Exercise 4

Return the first name concatenated with the last name from the **SWRIDEN** table. Return only rows where the uppercase value of the first name is 'PETER'.



Exercise 5

What is the lowest and highest SAT verbal scores for students who took the test in March or April 2006 using **SWRTEST**?

Exercise 6

Retrieve the PIDM and age (whole number) from **SWBPERS**. Use the birth date and current system date to obtain the age.

Section F: Nesting Functions

Lesson: Overview

Jump to TOC

Introduction

Nesting of functions within functions provides greater flexibility and complexity in writing SQL statements. Most functions can be nested with other functions to manipulate data. A number of commonly used nesting of functions will be provided in this section.

Note that this section contains only a very small subset of possible function combinations. There are countless combinations of functions.

Objectives

This section will examine the following commonly used nesting of functions:

- Nesting CASE
- Nesting COUNT
- Nesting DECODE
- Nesting SUBSTR
- Nesting SUM

Section Contents

Section F: Nesting Functions

Lesson: Nesting CASE

Jump to TOC

Nesting CASE with other functions

Case statements can have other functions nested within them. The following example looks at the first digit of the zip code to determine the region of the country in which the address is located.

```
  SQL> SELECT swraddr_pidm, CASE
                              WHEN substr(swraddr_zip,1,1) between '0' and '3'
                                   THEN 'Eastern'
                              WHEN substr(swraddr_zip,1,1) between '4' and '7'
                                   THEN 'Middle'
                              WHEN substr(swraddr_zip,1,1) > '7'
                                   THEN 'Western'
                              ELSE 'Foreign'
                            END as REGION
       FROM swraddr;

SWRADDR_PIDM REGION
------------ -------
       12340 Eastern
       12341 Western
       12342 Middle
       12343 Eastern
       12344 Middle
       12345 Eastern
```

Section F: Nesting Functions

Lesson: Nesting CASE (Continued)

Jump to TOC

Nesting multiple CASE statements

Case statements can be nested within themselves. This example evaluates the birth year and the sex of each person for determining a life insurance premium percentage. The older the person, the higher the life insurance premium. Since women tend to live longer than men do, their premiums are less.

```
SQL> SELECT swbpers_pidm, swbpers_birth_date, swbpers_sex,
            CASE
                WHEN to_char(swbpers_birth_date,'YYYY') < 1945
                     THEN CASE swbpers_sex
                               WHEN 'M' THEN ' 25%'
                               WHEN 'F' THEN ' 20%'
                               ELSE ' 30%'
                          END
                WHEN to_char(swbpers_birth_date,'YYYY') between 1946 and 1965
                     THEN CASE swbpers_sex
                               WHEN 'M' THEN ' 15%'
                               WHEN 'F' THEN ' 10%'
                               ELSE ' 20%'
                          END
                WHEN to_char(swbpers_birth_date,'YYYY') between 1966 and 1985
                     THEN CASE swbpers_sex
                               WHEN 'M' THEN '  5%'
                               WHEN 'F' THEN '  0%'
                               ELSE ' 10%'
                          END
                WHEN to_char(swbpers_birth_date,'YYYY') > 1985
                     THEN CASE swbpers_sex
                               WHEN 'M' THEN '  5%'
                               WHEN 'F' THEN '  0%'
                               ELSE ' 10%'
                          END
                ELSE ' 50%'
            END as INSURANCE_PREMIUM
      FROM swbpers;

SWBPERS_PIDM SWBPERS_BIR S INSU
------------ ----------- - ----
       12344 30-Oct-1973 M   5%
       12345 05-Jan-1984 F   0%
       12346 15-Feb-1961 F  10%
       12348 25-Mar-1941 F  20%
       12353 29-Jun-1965 M  15%
       12355 04-Jul-1960 F  10%
```

Section F: Nesting Functions

Lesson: Nesting COUNT

Jump to TOC

Nesting COUNT

The COUNT function can be nested with other functions to provide differing results.

COUNT with DISTINCT

There may be a need where just counting the number of records in a table is not sufficient. You may want to know how many unique values are in a table.

```
SQL> SELECT count(*) FROM swrregs;

  COUNT(*)
----------
        44


SQL> select count(DISTINCT swrregs_pidm) FROM swrregs;

COUNT(DISTINCTSWRREGS_PIDM)
---------------------------
                          9
```

These shows there are 44 total records in the SWRREGS table, but only 9 unique PIDM values. This is expected behavior for a registration table where a student is enrolled in multiple courses.

Section F: Nesting Functions

Lesson: Nesting DECODE

Jump to TOC

Nesting DECODE

Decode statements can be nested inside each other to provide additional evaluation of data.

This statement will check the confidential indicator before publishing the marital status of each person. If the person has requested confidentiality, 'Not Available' will be the result instead of their actual status:

```
SQL> SELECT swbpers_pidm,
 decode(swbpers_confid_ind,'N',decode(swbpers_mrtl_code,'S','Single',
                                                        'M','Married',
                                                        'W','Widowed',
                                                        'D','Divorced',
                                                        'Unspecified'),
                           'Y','Not Available',
                          null, 'Unknown')
    FROM swbpers;

SWBPERS_PIDM DECODE(SWBPER
------------ -------------
       12340 Not Available
       12341 Married
       12344 Single
       12350 Widowed
       12352 Divorced
       12355 Not Available
```

This could also have been written as a nested CASE statement:

```
SELECT swbpers_pidm,
       CASE swbpers_confid_ind
            WHEN 'N' THEN CASE swbpers_mrtl_code
                               WHEN 'S' THEN 'Single'
                               WHEN 'M' THEN 'Married'
                               WHEN 'W' THEN 'Widowed'
                               WHEN 'D' THEN 'Divorced'
                               ELSE          'Unspecified'
                          END
            WHEN 'Y' THEN 'Not Available'
            ELSE 'Unknown'
       END
  FROM swbpers;
```

Section F: Nesting Functions

Lesson: Nesting SUBSTR

Jump to TOC

Nesting SUBSTR

The SUBSTR function is often combined with the INSTR or LENGTH functions to parse character strings.

SUBSTR with INSTR

We can take the full name field from the SWVTELE view and break it back into its separate parts. This type of functionality is useful for conversions and loading data from third party systems.

A comma separates the last name from the first and middle names. The second space in the string, after the comma, separates the first name from the middle name.

```
SQL> SELECT swvtele_name,
            substr(swvtele_name,1,instr(swvtele_name,',') -1) "Last Name",
            substr(swvtele_name,instr(swvtele_name,',') +2,
                   (instr(swvtele_name,' ',1,2) -1) - instr(swvtele_name,',')
                   ) "First Name",
            substr(swvtele_name,instr(swvtele_name,' ',1,2) + 1) "Middle"
       FROM swvtele;

SWVTELE_NAME                 Last Name      First Name     Middle
---------------------------- -------------- -------------- ----------
Brown, Julie K               Brown          Julie          K
Smith, Robert E              Smith          Robert         E
Johnson, Peter S             Johnson        Peter          S
Jones-Erickson, Sandy J      Jones-Erickson Sandy          J
Erickson, Ralph L            Erickson       Ralph          L
Erickson, Susan T            Erickson       Susan          T
Marx, Joan Elizabeth         Marx           Joan           Elizabeth
Clifford, Stephanie Geena    Clifford       Stephanie      Geena
Serum, Tracy Paige           Serum          Tracy          Paige
Dukes, Michelle Q            Dukes          Michelle       Q
Johnson, Jeremy P            Johnson        Jeremy         P
McNair, Tracy A              McNair         Tracy          A
Miner, Christopher U         Miner          Christopher    U
Jameson, Jennifer W          Jameson        Jennifer       W
```

Section F: Nesting Functions

Lesson: Nesting SUBSTR (Continued)

Jump to TOC

SUBSTR with LENGTH

The SUBSTR function can be combined with the LENGTH function to parse part of a string where there are no clear delimiters to use in an INSTR and the length of the string may vary.

For example, credit card number lengths differ by credit card companies. A new government requirement to help protect against credit card fraud requires that credit card receipts only print the last 4 digits of the card number. You need to know the length of the card number to determine the last 4 digits.

```
SQL>  SELECT '*********' ||
        substr(credit_card_no,length(credit_card_no)-3,4) "CARD NO",
        expire_date
      FROM credit_card_table;


CARD NO              EXPIRE_DATE
-------------        -----------
*********1234        01/2001
*********4567        10/1997
*********6789        03/1999
```

Section F: Nesting Functions

Lesson: Nesting SUM

Jump to TOC

Nesting SUM

The sum function can be combined with other functions to summarize only certain data based on some criteria.

This query summarized account data based on detail code. If the code does not match in the decode statement, a zero is used so that the sum will not be affected.

```
SQL> SELECT sum(decode(twraccd_detc_code,'TUIT',twraccd_amount,0))
                "Tuition Total",
            sum(decode(twraccd_detc_code,'BOOK',twraccd_amount,0))
                "Book Total",
            sum(decode(twraccd_detc_code,'LABS',twraccd_amount,0))
                "Labs Total",
            sum(decode(twraccd_detc_code,'DORM',twraccd_amount,0))
                "Dorm Total"
       FROM twraccd;

Tuition Total Book Total Labs Total Dorm Total
------------- ---------- ---------- ----------
       6650.5     1400.2        220       1500
```

Be careful of the order you place function when using SUM:

```
SQL> select sum(round(twraccd_balance)) from twraccd;

SUM(ROUND(TWRACCD_BALANCE))
---------------------------
                       6122

SQL> select sum(twraccd_balance) from twraccd;

SUM(TWRACCD_BALANCE)
--------------------
              6121.2

SQL> select round(sum(twraccd_balance)) from twraccd;

ROUND(SUM(TWRACCD_BALANCE))
---------------------------
                       6121
```

Section F: Nesting Functions

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

Exercise 1

Using a combination of functions and the table containing address data, parse the street address field into House Number, Street Name, and Street direction.

Exercise 2

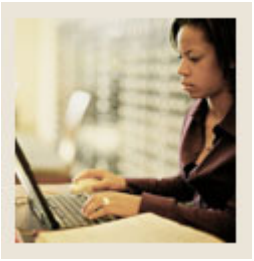
Sum the amount column in the TWRACCD table assigning positive values to Trans Type C and negative values to Trans Type P.

Section F: Nesting Functions

Lesson: Self Check (Continued)

Jump to TOC

Exercise 3 (Optional)
Adding to the SQL in Exercise 1, check for PO Box addresses and lump ‘PO Box’ into the Street Name. (Hint: Use the DECODE function).



Section G: Clauses

Lesson: Overview

Jump to TOC

Introduction

Clauses can be added to a SELECT statement to add conditions to the returned data as with the WHERE clause. In addition, clauses may be used to summarize data and change the order in which the data is returned.

Objectives

This section will examine the following clauses:

- WHERE
- ORDER BY
- GROUP BY
- HAVING

Section contents

Section G: Clauses

Lesson: The WHERE Clause

Jump to TOC

WHERE clause

As noted in Section D, the WHERE clause is comprised of one or more conditions added to a query or manipulation statements so that only certain records are selected or manipulated.

- **SELECT...**
- **FROM...**
- **WHERE...**

Without WHERE

Without the WHERE clause, all rows will be returned from the specified table:

```
SQL> SELECT * FROM SWBPERS;

SWBPERS_PIDM SWBPERS_S SWBPERS_B S S S SWBPERS_A USERID       ORIGIN
------------ --------- --------- - - - --------- ------------ ---------
       12340 585442212 02-AUG-73 S F Y 31-OCT-05 TRAIN_ORA101 TRAINING
       12341 682082678 12-NOV-70 M M N 10-DEC-05 TRAIN_ORA101 TRAINING
       12343 555444412 22-SEP-73 S F Y 05-DEC-05 TRAIN_ORA101 TRAINING
       12344 198767345 30-OCT-73 S M N 08-DEC-05 TRAIN_ORA101 TRAINING
       12345 955433412 05-JAN-84 W F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12346 643091257 15-FEB-61 D F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12348 231560987 25-MAR-41 M F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12350 340541234 01-APR-76 W F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12352 189054387 19-MAY-78 D M N 07-DEC-05 TRAIN_ORA101 TRAINING
       12353 035341098 29-JUN-65 M M N 07-DEC-05 TRAIN_ORA101 TRAINING
       12355 608321875 04-JUL-60 W F Y 07-DEC-05 TRAIN_ORA101 TRAINING
       12357 430896512 27-NOV-52 M M Y 07-DEC-05 TRAIN_ORA101 TRAINING
       12359 318760932 31-DEC-54 D F Y 07-DEC-05 TRAIN_ORA101 TRAINING
```

With WHERE

In the example below, rows are selected based on the criteria of birth date:

```
SQL> SELECT * FROM SWBPERS WHERE SWBPERS_BIRTH_DATE = '02-AUG-73';

SWBPERS_PIDM SWBPERS_S SWBPERS_B S S S SWBPERS_A USERID       ORIGIN
------------ --------- --------- - - - --------- ------------ ---------
       12340 585442212 02-AUG-73 S F Y 31-OCT-05 TRAIN_ORA101 TRAINING
```

Section G: Clauses

Lesson: ORDER BY

Jump to TOC

ORDER BY clause

The ORDER BY clause changes the order in which information is displayed.

Note: The order columns in the ORDER BY clause do not have to appear in the SELECT clause. Without specifying ascending or descending order, ascending is assumed.

- **SELECT...**
- **FROM...**
- **WHERE...**
- **ORDER BY...**

Example 1

```
SQL> SELECT  swbpers_pidm, swbpers_birth_date
       FROM  swbpers
   ORDER BY  swbpers_birth_date;

SWBPERS_PIDM SWBPERS_B
------------ ---------
       12348 25-MAR-41
       12357 27-NOV-52
       12359 31-DEC-54
       12355 04-JUL-60
       12346 15-FEB-61
       12353 29-JUN-65
       12341 12-NOV-70
       12340 02-AUG-73
       12343 22-SEP-73
       12344 30-OCT-73
       12350 01-APR-76
       12352 19-MAY-78
       12345 05-JAN-84
```

Example 2

```
SQL> SELECT  swbpers_pidm, swbpers_birth_date
       FROM  swbpers
   ORDER BY  swbpers_birth_date DESC;

SWBPERS_PIDM SWBPERS_B
------------ ---------
       12345 05-JAN-84
       12352 19-MAY-78
       12350 01-APR-76
...
```

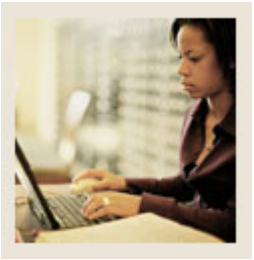
Section G: Clauses

Lesson: ORDER BY (Continued)

Jump to TOC

Example 3

```
SQL> SELECT  swriden_last_name, swriden_first_name
       FROM  swriden
   ORDER BY  swriden_last_name, swriden_first_name;

LAST_NAME            FIRST_NAME
-------------------- ------------
Brown                Julie
Brown                Julie
Erickson             Ralph
Erickson             Susan
Johnson              Peter
Jones                Sandy
Jones-Erickson       Sandy
Smith                Robert
White                Nancy
```



Section G: Clauses

Lesson: Ordering by Position

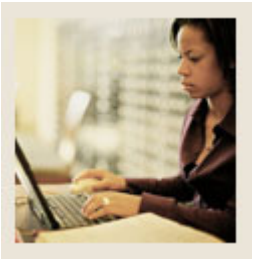
Jump to TOC

Column positions

Rather than specifying column names, refer to the columns by their position in the SELECT statement.

```
SQL> SELECT    swriden_last_name, swriden_first_name
     FROM      swriden
 ORDER BY      1,2;

SWRIDEN_LAST_NAME    SWRIDEN_FIRST_NAME
-------------------- ------------------
Brown                Julie
Brown                Julie
Erickson             Ralph
Erickson             Susan
Johnson              Peter
Jones                Sandy
Jones-Erickson       Sandy
Smith                Robert
White                Nancy
…
```

Note: Although ordering by position requires less programming effort, it should not be used for programs. Adding columns to a table, re-arranging columns in a table, or re-arranging columns in the SELECT statement may cause positional sorting to have unpredictable results.



Section G: Clauses

Lesson: GROUP BY

Jump to TOC

GROUP BY clause

Use the GROUP BY clause to group selected rows and return a single row of summary information. Oracle collects each group of rows based on the values of the expression(s) specified in the GROUP BY clause.

Restrictions

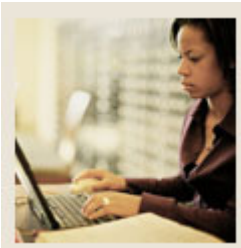
If a SELECT statement contains the GROUP BY clause, the select list can only contain these types of expressions:

- Constants
- Group functions
- Expressions identical to those in the GROUP BY clause
- Expressions involving the above expressions that evaluate to the same value for all rows in a group

Example 1

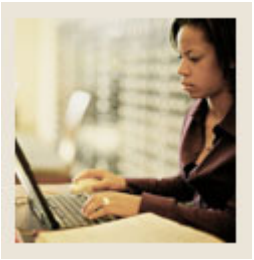
```
SQL> SELECT    swrregs_pidm, AVG(swrregs_gpa)
       FROM    swrregs
   GROUP BY    swrregs_pidm;

SWRIDEN_PIDM AVG(SWRREGS_GPA)
------------ ----------------
       12340            3.025
       12341            2.475
       12342             2.45
       12343        3.1333333
       12344                0
       12345              2.5
       12346        2.7333333
```

The SELECT statement contains both a column name and a group function. This would return an error without the GROUP BY clause, which references *swbpers_pidm* and causes *AVG(swbpers_gpa)* to average the rows associated with each *swbpers_pidm*.



Example 2

```
SQL> SELECT twraccd_term_code, twraccd_pidm, SUM(twraccd_amount)
       FROM twraccd
      GROUP BY twraccd_term_code, twraccd_pidm
      ORDER BY twraccd_term_code, twraccd_pidm;

TWRACC TWRACCD_PIDM SUM(TWRACCD_AMOUNT)
------ ------------ -------------------
200402        12344                1120
200501        12340              2500.7
200501        12341                1600
200501        12343                1900
200501        12344              2100.5
200501        12345                 300
200501        12346                1900
200502        12342                1201
200502        12344                1150
200602        12342                1350
200602        12341                1000
```



Section G: Clauses

Lesson: HAVING

Jump to TOC

HAVING clause

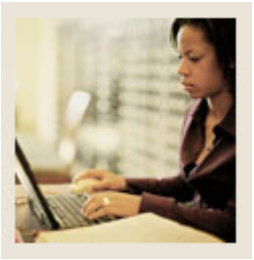
A HAVING clause places a condition on the GROUP function.

Example 1

```
SQL> SELECT  SWRIDEN_PIDM, COUNT(*)
       FROM  SWRIDEN
   GROUP BY  SWRIDEN_PIDM
     HAVING  COUNT(*) > 1;

SWRIDEN_P   COUNT(*)
--------- ---------
    12340         2
    12343         2
    12350         2
    12351         3
    12355         2
    12357         2
```

Example 2

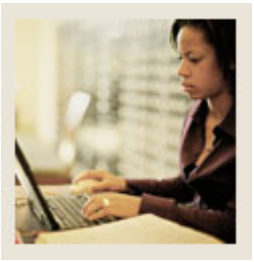
```
SQL> SELECT    swrregs_pidm, AVG(swrregs_gpa)
       FROM    swrregs
   GROUP BY    swrregs_pidm
     HAVING    AVG(swrregs_gpa) < 3.0;

SWRIDEN_P  AVG(SWRREGS_GPA)
--------- -----------------
    12341             2.475
    12342              2.45
    12344                 0
    12345               2.5
    12346         2.7333333
```



Example 3

```
SQL>   SELECT  twraccd_pidm, twraccd_term_code, SUM(twraccd_amount)
         FROM  twraccd
        WHERE  twraccd_term_code >= '200501'
     GROUP BY  twraccd_pidm, twraccd_term_code
       HAVING  SUM(twraccd_amount) > 500
     ORDER BY  twraccd_pidm, twraccd_term_code;

TWRACCD_PIDM TWRACC SUM(TWRACCD_AMOUNT)
------------ ------ --------------------
       12340 200501               2500.7
       12341 200501                 1600
       12341 200602                 1000
       12342 200502                 1201
       12342 200602                 1350
       12343 200501                 1900
       12344 200501               2100.5
       12344 200502                 1150
       12346 200501                 1900
```

Note: ORDER BY must be the last line in a WHERE clause.



Section G: Clauses

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

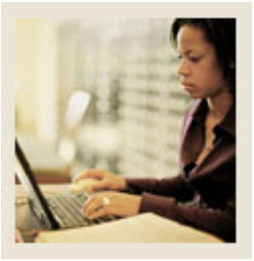
Examine the data to find out if the students have been receiving, on the average, low marks for courses. Build the query step by step.

Exercise 1

Examine the Student Grades table (**SWRREGS**) using the DESC command. You will be referring to this table for the rest of the section.

Exercise 2

Find the average GPA for each course number. Group by the course number.



Exercise 3

To easily locate the courses with particularly low averages, order your data by the average (lowest first).

Exercise 4

Reduce the list so that only courses with averages below 2.0 are returned using a **WHERE** clause. Did you receive an error? Why?

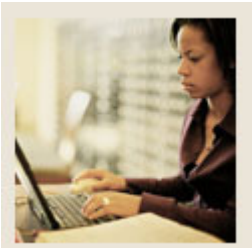
Exercise 5

Try Exercise 4 again, but put the condition in a **HAVING** clause.

Exercise 6

Your institution has changed the testing format for courses 10001 through 10006 from consecutive terms of 200602 and 200702. Examine the effects of the format change. In order to do this, select the course number, term code, and average GPA for the above courses and terms using **SWRREGS**. Group and order by course number and term code.

According to the data, has the new test format had a positive or negative effect on the GPAs?



Exercise 7

To ensure that there is enough data to make a valid conclusion, make sure at least 3 students have taken the course in a term for the row to be returned. Use a **HAVING** clause to restrict the data being returned.

Section H: Advanced Queries

Lesson: Overview

Jump to TOC

Introduction

Up to this point, we have provided examples of queries that return rows from only a single table. However, on many occasions we are concerned with retrieving data from many tables within the same query. For example, the student name, address, personal information, accounting information, registration information, and test scores are all kept in separate tables.

This section looks at how to join this information, so that data from two or more tables are retrieved with one SQL statement.

Objectives

At the end of this section, participants will be able to

- create queries to retrieve data from more than one table
- use the set operators of UNION, INTERSECT, and MINUS to combine two or more queries into one result
- create subqueries to solve complex queries.

Section H: Advanced Queries

Lesson: Overview (Continued)

Jump to TOC

Section contents

Section H: Advanced Queries

Lesson: Joins

Jump to TOC

Joins

A join is a SELECT statement that combines rows from two or more tables and/or views. Oracle performs joins whenever multiple tables appear in the FROM clause of a SELECT statement.

Joins and WHERE clause

The optional WHERE clause determines how Oracle combines rows of the tables. If the optional WHERE clause is omitted, the result is called a Cartesian Product.

For instance, if there are 10 rows in the swriden table and 5 rows in the swraddr table, the resulting join of these two tables without a WHERE condition would return 50 rows – each row in the swriden table will be joined separately to each row in the swraddr table. This type of join is rarely useful.

Equi-join

Returns rows from two or more tables based on an equality condition.

```
SELECT   select_list
  FROM   table1 [,table2] [,table3] …
 WHERE   table1.column = table2.column
  [AND   table1.column = table3.column] …
```

Example 1

```
SQL> SELECT    swriden_last_name||', '||swriden_first_name||' '||
               swriden_mi NAME,
               swraddr_street_line1||' '|| swraddr_city||', '||
               swraddr_stat_code|| ' ' || swraddr_zip ADDRESS
       FROM    swriden, swraddr
      WHERE    swriden_pidm = swraddr_pidm
        AND    swriden_change_ind IS NULL;

NAME                      ADDRESS
------------------------- ----------------------------------------
Brown, Julie K            506 BROWN STREET WEST CHESTER, PA 19380
Smith, Robert E           210 PINE STREET SAN FRANCISCO, CA 94082
Johnson, Peter S          PO BOX 1035 BROWNVILLE, KY 67233
Jones-Erickson, Sandy J 23 MARKET STREET WEST CHESTER, PA 19382
Erickson, Ralph L         18 CHESTNUT ROAD NEW ORLEANS, LA 23456
```

If a column name is ambiguous (same, exact column name in different tables), the column must be preceded by the table name (e.g. tableA.column1 = tableB.column1).

Section H: Advanced Queries

Lesson: Joins (Continued)

Jump to TOC

SQL 99 Equi-Join

Beginning with Oracle 9i Release 2, Oracle began supporting the SQL 99 or ANSI join syntax. People who have worked with other SQL databases may be more familiar with this syntax.

In the SQL 99 standard, the JOIN takes place in the FROM clause instead of the WHERE clause.

- Natural Join – Join all columns that have the same name in all tables. The columns with like names must have the same data types or you will receive an error. Can't use if join columns do not have the same name.

```
SELECT <item list>
  FROM tableA NATURAL JOIN tableB
 WHERE <conditions>
```

- Join..Using – Join two tables using the column(s) specified and not all columns that match in name/type. The column names MUST match to use this syntax.

```
SELECT <item list>
  FROM tableA JOIN tableB USING (tableA.column2 = tableB.column2)
 WHERE <conditions>
```

- Join On – Specify the column names you want to join together. Required when column names do not match exactly. Since Banner includes the table name in the column name, this is the only method that can be used for SQL 99 joins.
  - Can be equi-join or any other operator -- >, <, <>, etc.

Section H: Advanced Queries

Lesson: Joins (Continued)

Jump to TOC

```
SELECT swriden_last_name||', '||swriden_first_name
       ||' '||swriden_mi NAME,
       swraddr_street_line1||' '|| swraddr_city
       ||', '|| swraddr_stat_code ||' '|| swraddr_zip ADDRESS
  FROM swriden JOIN swraddr
    ON (swriden_pidm = swraddr_pidm
   AND  swriden_change_ind IS NULL);

NAME                    ADDRESS
----------------------- ----------------------------------------
Brown, Julie K          506 BROWN STREET WEST CHESTER, PA 19380
Smith, Robert E         210 PINE STREET SAN FRANCISCO, CA 94082
Johnson, Peter S        PO BOX 1035 BROWNVILLE KY, 67233
```

NOTE: you don't have to prefix the column names in the select clause with the table name when using SQL99 JOIN syntax.

For example, in standard Oracle SQL where both tableA and tableB had a column called column2, you would have to write the query specifying which table you want column2 to be taken from:

```
SQL> SELECT column1, tableA.column2, column3…
       FROM tableA, tableB
      WHERE tableA.column2 = tableB.column2;
```

Using SQL 99 syntax, you would write the same query as:

```
SQL> SELECT column1, column2, column3…
       FROM tableA JOIN tableB
            USING (tableA.column2 = tableB.column2);
```

Section H: Advanced Queries

Lesson: Joins (Continued)

Jump to TOC

Outer joins

An outer join returns all the rows returned by an equi-join as well as those rows from one table that do not have any rows from the other table. The table that might not contain matching data is appended with a '(+)' in the WHERE clause.

This syntax is specific to Oracle and is not available in any other version of SQL.

Example

```
SQL> SELECT    swriden_last_name||', '||swriden_first_name||' '||
               swriden_mi NAME,
               swraddr_street_line1||' '|| swraddr_city||', '||
               swraddr_stat_code   ||' '|| swraddr_zip ADDRESS
         FROM  swriden, swraddr
        WHERE  swriden_pidm = swraddr_pidm (+)
          AND  swriden_change_ind IS NULL;


NAME                      ADDRESS
------------------------  ----------------------------------------
Brown, Julie K            506 BROWN STREET WEST CHESTER PA 19380
Smith, Robert E           210 PINE STREET SAN FRANCISCO CA 94082
Johnson, Peter S          PO BOX 1035 BROWNVILLE KY 67233
Jones-Erickson, Sandy J   23 MARKET STREET WEST CHESTER PA 19382
Erickson, Ralph L         18 CHESTNUT ROAD NEW ORLEANS LA 23456
Erickson, Susan T
White, Nancy Carol
Marx, Joan Elizabeth
```

Section H: Advanced Queries

Lesson: Joins (Continued)

Jump to TOC

Outer joins (SQL 99)

The SQL 99 standard for Outer Joins is different from the Oracle version. There are several different types of Outer Joins in the SQL 99 standard.

- Left Outer Join – where the table to the LEFT of the clause is the driving table – records in the table to the right of the clause may or may not exist.

- Right Outer Join – where the table to the RIGHT of the clause is the driving table – records in the table to the left of the clause may or may not exist.

- Full Outer Join – where both tables drive – records that do not show up for either side are shown. The only way to write this type of query in Oracle prior to the SQL 99 standard was to use two separate queries and join the results using a UNION operator (discussed later).

A query that joins the SWRIDEN table to the SWRADDR table that shows SWRIDEN records without addresses would be written as:

```
SQL> SELECT   swriden_last_name||', '||swriden_first_name
              ||' '|| swriden_mi NAME,
                 swraddr_street_line1||' '|| swraddr_city
              ||' '|| swraddr_stat_code||' '|| swraddr_zip ADDRESS
       FROM   swriden LEFT OUTER JOIN swraddr
         ON   swriden_pidm = swraddr_pidm
      WHERE   swriden_change_ind IS NULL;

NAME                      ADDRESS
------------------------- ----------------------------------------
Brown, Julie K            506 BROWN STREET WEST CHESTER PA 19380
Smith, Robert E           210 PINE STREET SAN FRANCISCO CA 94082
Johnson, Peter S          PO BOX 1035 BROWNVILLE KY 67233
Jones-Erickson, Sandy J   23 MARKET STREET WEST CHESTER PA 19382
Erickson, Ralph L         18 CHESTNUT ROAD NEW ORLEANS LA 23456
Erickson, Susan T
White, Nancy Carol
```

Section H: Advanced Queries

Lesson: Joins (Continued)

Jump to TOC

Self-joins

There may be cases when you will need to join a table to itself. This is especially helpful when finding duplicates. In order to join a table to itself you must use table aliases. Below is an example of a self-join.

Example 1

Ensure that no duplicates exist in the validation table SWVTERM. Using a self-join, retrieve the records where the term coded match but the descriptions do not.

```
   SQL> SELECT A.swvterm_term_code, A.swvterm_desc,
         B.swvterm_term_code, B.swvterm_desc
         FROM swvterm A, swvterm B
         WHERE A.swvterm_term_code = B.swvterm_term_code
         AND A.swvterm_desc <> B.swvterm_desc;

SWVTER SWVTERM_DESC                    SWVTER SWVTERM_DESC
------ ------------------------------ ------ -------------------------
200705 Spring Semester 2007           200705 Summer Semester 2007
200705 Summer Semester 2007           200705 Spring Semester 2007
```

Example 2

Another example is an employee/manager relationship. A manger is also an employee, so an employee table may look like:

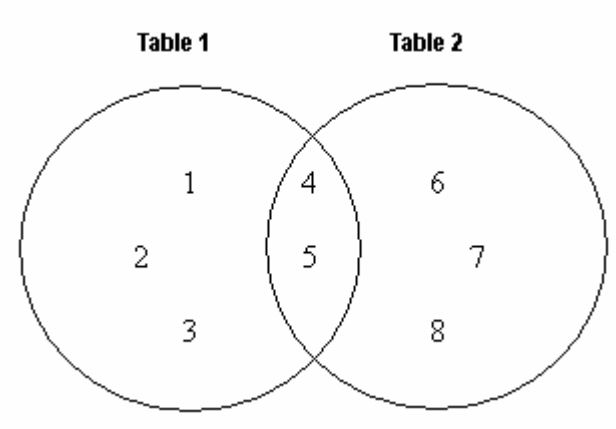
```
      EMPLOYEE
      --------------------------------
      Employee_ID
      Employee_Name
      Employee_Dept
      Employee_Manager_ID

SELECT EE.employee_id, EE.employee_name EMPLOYEE,
       MGR.employee_name MANAGER
  FROM employee EE, employee MGR
 WHERE EE.employee_manager_id = MGR.employee_id;

EMPLOYEE_ID EMPLOYEE          MANAGER
----------- --------------    ---------------------------
12345        Smith, John      Jones, Joe
34567        Benson, Mary     Carson, Cynthia
56789        Jones, Joe       Carson, Cynthia
```

Self-joins (SQL 99)

The same query above that checks for duplicate entries in the SWVTERM table would be written in SQL 99 syntax as:

```
SQL> SELECT A.swvterm_term_code, A.swvterm_desc,
            B.swvterm_term_code, B.swvterm_desc
       FROM swvterm A JOIN swvterm B
         ON (A.swvterm_term_code = B.swvterm_term_code
        AND A.swvterm_desc <> B.swvterm_desc);


SWVTER SWVTERM_DESC                SWVTER SWVTERM_DESC
------ ------------------------- ------ -------------------------
200705 Spring Semester 2007       200705 Summer Semester 2007
200705 Summer Semester 2007       200705 Spring Semester 2007
```

Diagram



UNION

Returns all distinct rows for both select statements.

- 1, 2, 3, 4, 5, 6, 7, 8

UNION ALL

Returns all rows for both select statements, regardless of duplicates.

- 1, 2, 3**, 4, 5, 4, 5,** 6, 7, 8

INTERSECT

Returns only rows returned by both of the queries.

- 4, 5

MINUS

Returns all rows returned by the preceding query that were not present in the second.

- 1, 2, 3

Section H: Advanced Queries

Lesson: Joins - Union, Union All, Intersect, Minus (Continued)

Jump to TOC

Examples

Union & Union All

Union returns all distinct rows returned by both of the two queries. Union All returns all the rows of the two queries, including duplicates.

Union

In this example, select SWRIDEN records without SWRADDR records and UNION with SWRADDR records without SWRIDEN records

```
SQL> SELECT swriden_pidm, swriden_last_name,
            swraddr_pidm, swraddr_atyp_code
       FROM swriden, swraddr
      WHERE swriden_pidm = swraddr_pidm (+)
        AND swriden_change_ind is null
  UNION
     SELECT swriden_pidm, swriden_last_name,
            swraddr_pidm, swraddr_atyp_code
       FROM swriden, swraddr
      WHERE swriden_pidm (+) = swraddr_pidm
        AND swriden_change_ind (+) is null;

SWRIDEN_PIDM SWRIDEN_LAST_NAME         SWRADDR_PIDM SW
------------ ------------------------- ------------ --
       12340 Brown                            12340 MA
       12341 Smith                            12341 PR
       12342 Johnson                          12342 PR
       12343 Jones-Erickson                   12343 P1
       12344 Erickson                         12344 MA
       12345 Erickson                         12345 PR
       12346 White
       12347 Marx                             12347 P1
       12348 Clifford                         12348 P1
       12349 Serum                            12349 P1
       12350 Dukes                            12350 MA
       12351 Dukes
       12352 Johnson                          12352 PR
       12353 Johnson
       12354 Bristow
       12355 McNair                           12355 P1
       12356 Miner                            12356 P1
       12357 Roberson
       12358 Jameson                          12358 MA
       12359 Peterson
                                              12999 P1
```

Section H: Advanced Queries

Lesson: Joins - Union, Union All, Intersect, Minus (Continued)

Jump to TOC

Union All

```
  SQL> SELECT swriden_pidm, swriden_last_name,
              swraddr_pidm, swraddr_atyp_code
         FROM swriden, swraddr
        WHERE swriden_pidm = swraddr_pidm (+)
          AND swriden_change_ind is null
    UNION ALL
       SELECT swriden_pidm, swriden_last_name,
              swraddr_pidm, swraddr_atyp_code
         FROM swriden, swraddr
        WHERE swriden_pidm (+) = swraddr_pidm
          AND swriden_change_ind (+) is null;

SWRIDEN_PIDM SWRIDEN_LAST_NAME         SWRADDR_PIDM SW
------------ ------------------------- ------------ --
       12340 Brown                             12340 MA
       12341 Smith                             12341 PR
       12342 Johnson                           12342 PR
       12343 Jones-Erickson                    12343 P1
       12344 Erickson                          12344 MA
       12345 Erickson                          12345 PR
       12346 White
       12347 Marx                              12347 P1
       12348 Clifford                          12348 P1
       12349 Serum                             12349 P1
       12350 Dukes                             12350 MA
       12351 Dukes
       12352 Johnson                           12352 PR
       12353 Johnson
       12354 Bristow
       12355 McNair                            12355 P1
       12356 Miner                             12356 P1
       12357 Roberson
       12358 Jameson                           12358 MA
       12359 Peterson
       12340 Brown                             12340 MA
       12341 Smith                             12341 PR
       12342 Johnson                           12342 PR
       12343 Jones-Erickson                    12343 P1
       12344 Erickson                          12344 MA
       12345 Erickson                          12345 PR
       12347 Marx                              12347 P1
       12348 Clifford                          12348 P1
       12349 Serum                             12349 P1
       12350 Dukes                             12350 MA
       12352 Johnson                           12352 PR
       12355 McNair                            12355 P1
       12356 Miner                             12356 P1
       12358 Jameson                           12358 MA
                                               12999 P1
```

Section H: Advanced Queries

Lesson: Joins - Union, Union All, Intersect, Minus (Continued)

Jump to TOC

NOTE: The following intersect and minus examples refer to two imaginary tables, SWRIDEN1 and SWRIDEN2, which contain the following data:

SWRIDEN1

```
SQL> SELECT swriden1_last_name, swriden1_first_name
       FROM swriden1
      WHERE swriden1_change_ind IS NULL;

SWRIDEN1_LAST_NAME          SWRIDEN1_FIRST_NAME
--------------------------- ------------------
Jones-Erickson              Sandy
White                       Nancy
Marx                        Joan
```

SWRIDEN2

```
SQL> SELECT swriden2_last_name, swriden2_first_name
       FROM swriden2
      WHERE swriden2_change_ind IS NULL;

SWRIDEN2_LAST_NAME          SWRIDEN2_FIRST_NAME
--------------------------- --------------------
Smith                       Robert
Johnson                     Peter
White                       Nancy
Marx                        Joan
```

Intersect

Intersect returns only rows returned by both of the queries.

```
SQL>   SELECT swriden1_last_name, swriden1_first_name
         FROM swriden1
        WHERE swriden1_change_ind IS NULL
    INTERSECT
       SELECT swriden2_last_name, swriden2_first_name
         FROM swriden2
        WHERE swriden2_change_ind IS NULL;


SWRIDEN1_LAST_NAME        SWRIDEN1_FIRST_NAME
------------------------- --------------------
Marx                      Joan
White                     Nancy
```

Minus

Minus returns all rows returned by the first query that are not present in the second.

```
SQL>   SELECT swriden1_last_name, swriden1_first_name
         FROM swriden1
        WHERE swriden1_change_ind IS NULL
        MINUS
       SELECT swriden2_last_name, swriden2_first_name
         FROM swriden2
        WHERE swriden2_change_ind IS NULL;

SWRIDEN1_LAST_NAME  SWRIDEN1_FIRST_NAME
------------------  -------------------
Jones-Erickson      Sandy
```

Section H: Advanced Queries

Lesson: Subqueries

Jump to TOC

Subquery

A subquery is a form of SELECT command that appears inside another SQL statement. A subquery is sometimes called a nested query. The statement containing a subquery is called the parent statement.

Example

```
  SQL>  SELECT swriden_last_name||' '||swriden_first_name "NAME",
                swvcrse_desc "COURSE", swrregs_gpa "GPA"
           FROM swvcrse, swriden, swrregs
          WHERE swrregs_pidm = swriden_pidm
            AND swrregs_crn = swvcrse_crn
            AND swrregs_gpa > =
               (SELECT  AVG(swrregs_gpa)
                  FROM  swrregs)
            AND swriden_change_ind IS NULL
       ORDER BY swvcrse_desc, swriden_last_name,
                swriden_first_name, swriden_mi;

NAME                         COURSE                              GPA
---------------------------- ------------------------------ ----------
Erickson Susan               Algebra                            3.9
Johnson Peter                Anthropology                       3.3
Brown Julie                  Biology                              3
Brown Julie                  Biology                            3.1
Brown Julie                  Calculus                           3.2
Erickson Susan               Calculus                             3
Johnson Peter                Calculus                           2.9
Jones-Erickson Sandy         Calculus                             4
Johnson Peter                European History                   3.4
Serum Tracy                  Philosophy                         3.1
Smith Robert                 Photography                        3.1
White Nancy                  Photography                        3.6
...
```

Limitations

The subquery in the previous example returns only one row for each row evaluated in the parent query. If the following statement were issued, without the AVG function, an error would occur.

```
SQL>  SELECT swriden_last_name||' '||swriden_first_name "NAME",
             swvcrse_desc "COURSE", swrregs_gpa "GPA"
        FROM swvcrse, swriden, swrregs
       WHERE swrregs_pidm = swriden_pidm
         AND swrregs_crn = swvcrse_crn
         AND swrregs_gpa > =
               (SELECT swrregs_gpa
                   FROM swrregs
                  WHERE swrregs_gpa > 2)
         AND swriden_change_ind IS NULL
    ORDER BY swvcrse_desc, swriden_last_name,
            swriden_first_name;

ERROR:
ORA-01427: single-row subquery returned more than one row
```

The SQL statement fails because the parent query is expecting only one row to be returned from the subquery.

Section H: Advanced Queries

Lesson: Subqueries Returning Multiple Values

Jump to TOC

Multiple row comparisons

To evaluate comparisons that return more than a single row, use the following:

- ANY
- ALL
- IN / NOT IN
- EXISTS / NOT EXISTS

ANY

The example below would calculate the average GPA for each course, and then select the people whose GPAs in a course are below any of the course averages.

```
SQL> SELECT swriden_last_name, swriden_first_name,
            swvcrse_desc "COURSE", swrregs_gpa "GPA"
       FROM swvcrse, swriden, swrregs
      WHERE swrregs_pidm = swriden_pidm
        AND swrregs_crn = swvcrse_crn
        AND swrregs_gpa < ANY  (SELECT AVG(swrregs_gpa)
                                  FROM swrregs
                              GROUP BY swrregs_crn)
        AND swriden_change_ind IS NULL;
```

ALL

The example below would take the average GPA for each course, and then select the people whose GPAs in a course are below all of the course averages.

```
SQL> SELECT    swriden_last_name, swriden_first_name,
               Swvcrse_desc "COURSE", swrregs_gpa "GPA"
       FROM    swvcrse, swriden, swrregs
      WHERE    swrregs_pidm = swriden_pidm
        AND    swrregs_crn = swvcrse_crn
        AND    swrregs_gpa < ALL     (SELECT    AVG(swrregs_gpa)
                                        FROM    swrregs
                                    GROUP BY    swrregs_crn)
        AND    swriden_change_ind IS NULL;
```

Section H: Advanced Queries

Lesson: Subqueries Returning Multiple Values (Continued)

Jump to TOC

IN

Use the IN operator to evaluate equality to any member of the test.

```
SQL>   SELECT swriden_last_name, swriden_first_name,
              twraccd_detc_code, twraccd_amount, twraccd_balance
         FROM swriden, twraccd
        WHERE swriden_pidm = twraccd_pidm
          AND swriden_change_ind IS NULL
          AND twraccd_pidm IN (SELECT swrstdn_pidm
                                 FROM swrstdn
                                WHERE swrstdn_stdn_code = 'SS')
     ORDER BY swriden_last_name, swriden_first_name;

SWRIDEN_LAST_NAME SWRIDEN_FIRST_NAME TWRA TWRACCD_A TWRACCD_B
----------------- ------------------ ---- --------- ---------
Erickson          Ralph              TUIT       750       750
Erickson          Ralph              BOOK       400       400
Erickson          Ralph              LABS       120       120
Erickson          Ralph              MEAL       900       900
Erickson          Ralph              DORM      1000      1000
Erickson          Ralph              CASH       800      -800
Erickson          Ralph              CRED     400.5    -400.5
```

EXISTS

Evaluates to TRUE if the subquery returns a row:

```
SQL> SELECT swriden_last_name, swriden_first_name
       FROM swriden
      WHERE swriden_change_ind IS NULL
        AND EXISTS (SELECT 'X' FROM swraddr
                      WHERE swraddr_pidm=swriden_pidm);
```

NOT EXISTS

Evaluates to TRUE if the subquery returns no rows:

```
SQL> SELECT swriden_last_name, swriden_first_name
       FROM swriden
      WHERE swriden_change_ind IS NULL
        AND NOT EXISTS (SELECT 'X' FROM swraddr
                          WHERE swraddr_pidm=swriden_pidm);
```

Section H: Advanced Queries

Lesson: Nested Subqueries

Jump to TOC

Nesting

Nesting is the act of putting several subqueries in serial:

```
SQL>     SELECT swriden_last_name, swriden_first_name
           FROM swriden
          WHERE swriden_change_ind IS NULL
            AND swriden_pidm IN
                (SELECT swbpers_pidm
                   FROM swbpers
                  WHERE swbpers_mrtl_code = 'S'
                    AND swbpers_pidm IN
                        (SELECT twraccd_pidm
                           FROM twraccd
                          WHERE twraccd_term_code = '200501'));

LAST_NAME            FIRST_NAME
-------------------- ---------------
Brown                Julie
Jones-Erickson       Sandy
Erickson             Ralph
```

Section H: Advanced Queries

Lesson: Correlated Subqueries

Jump to TOC

Correlated subqueries

A correlated subquery is a SELECT statement inside the WHERE clause of a SQL statement which is correlated (or makes reference) to one or more columns in the enclosing SQL statement.

In the preceding subquery examples, each subquery was executed once, and the resulting value was used by the WHERE clause of the main query. You can also compose a subquery that is executed repeatedly, once for each candidate row considered for selection by the main query.

Correlated subqueries can also contain tables used by the main query. If this is the case, the main query should define an alias in order to make references.

Example

```
SQL> SELECT    swriden_last_name||' '|| swriden_first_name "NAME",
               swvcrse_desc "COURSE", swrregs_gpa "GPA"
         FROM  swvcrse, swriden, swrregs a
        WHERE  a.swrregs_pidm = swriden_pidm
          AND  a.swrregs_crn = swvcrse_crn
          AND  swriden_change_ind IS NULL
          AND  swrregs_gpa < (SELECT AVG(swrregs_gpa)
                                FROM swrregs b
                               WHERE b.swrregs_crn = a.swrregs_crn);

NAME                COURSE               GPA
------------------  ---------------      ------
Brown Julie         Biology              2
Brown Julie         Speech               2.8
Smith Robert        Photography          3.1
Johnson Peter       Zoology              2.3
```

Section H: Advanced Queries

Lesson: Correlated Subqueries (Continued)

Jump to TOC

Find most recent row

If the swrstdn (student standing) table contains both current and historical data, then a correlated subquery can find the most recent row:

```
SQL> SELECT    swriden_first_name, swriden_last_name,
               swrstdn_stdn_code STANDING, swvstdn_desc "DESC"
         FROM  swriden, swrstdn a, swvstdn
        WHERE  swriden_pidm = a.swrstdn_pidm
          AND  a.swrstdn_stdn_code = swvstdn.swvstdn_code
          AND  swriden_change_ind IS NULL
          AND  a.swrstdn_activity_date =
                          (SELECT MAX(swrstdn_activity_date)
                             FROM swrstdn b
                            WHERE b.swrstdn_pidm = a.swrstdn_pidm);
```

Performance

The answer to joining tables and performance (or not joining tables) is the correlated subquery. Correlated subqueries are most often used when a column evaluation in a WHERE clause is not necessary in the select list.

Correlated subqueries require more system resources than regular subqueries, because the subquery is executed once for each row in the main query. Regular subqueries are executed only once, and the result used for all rows in the main query. For this reason, use correlated queries sparingly.

Joins can often used to produce the same results as subqueries. In some cases the join will be faster; in others, the subquery will be faster. Trying both approaches and comparing their performance is recommended.

Section H: Advanced Queries

Lesson: Dynamic SQL

Jump to TOC

Definition

Dynamic SQL is SQL that creates executable SQL statements.

Usage

This type of SQL is useful when you want to create many similar SQL statements with different values without having to prompt for values using parameters. You can send the output of dynamically created statements to a file that can be executed later.

Example

```
SQL> SELECT 'SELECT * FROM ' || table_name || ';'
       FROM user_tables
      ORDER BY table_name;

'SELECT*FROM'||TABLE_NAME||';'
------------------------------------------
SELECT * FROM HIGH_MATH;
SELECT * FROM HIGH_VERBAL;
SELECT * FROM SWBPERS;
SELECT * FROM SWRADDR;
SELECT * FROM SWRIDEN;
SELECT * FROM SWRREGS;
SELECT * FROM SWRSTDN;
SELECT * FROM SWRTEST;
SELECT * FROM SWVCRSE;
SELECT * FROM SWVSTDN;
SELECT * FROM SWVTERM;
SELECT * FROM TEMP;
SELECT * FROM TWRACCD;
SELECT * FROM TWVDETC;

14 rows selected.
```

Don't forget to include the semi colon (;) in your statement creation so that SQL*Plus knows when each statement ends.

Section H: Advanced Queries

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

Exercise 1

Select the ID and combined SAT scores from the **SWRIDEN** and **SWRTEST** tables, using an equi-join. Join by PIDM.

Exercise 2

Create a report that contains the same information as above (using the same tables), but also include students who have not taken the SAT test. Use an outer join.

Exercise 3

Return the PIDM(s) of the students who are in the **SWRIDEN** table but not in the **SWRREGS** table, using the keyword **MINUS**.

Exercise 4

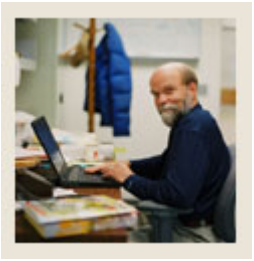
Find the person who has the highest SAT VERBAL score in the SWRTEST table. Show the PIDM, Name, and test score for that person. (Hint: Use a subquery to determine the highest score.)

Exercise 5

Create a query that will select the ID, Full Name, most recent TERM, and classes registered for that term from the **SWRIDEN, SWVCRSE** and **SWRREGS** tables. (Hint: Use a correlated subquery to obtain the most recent TERM_CODE for each person.)

Exercise 6

Create a SQL statement to create dynamic SQL that will issue a query against all of your tables, returning the number of records per table that belong to Julie Brown (Hint: Get Julie's PIDM first). Execute the SQL that you create to make sure it works correctly.



Section I: Insert, Update and Delete

Lesson: Overview

Jump to TOC

Introduction

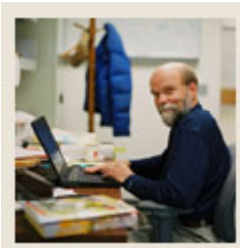
Throughout the previous sections, we have discussed retrieving data from the database. This section covers data manipulation: inserting, removing, and updating data within a table. More commonly, end users perform these tasks through a different software application, such as an Oracle Forms application.

Through a Forms application (such as Banner), a user can make changes to a few rows of data, but larger changes or mass updates require SQL*Plus. For example, what if 10,000 rows in the swriden table contained an invalid change indicator? An end user could query rows with the faulty indicator through a form and make the corrections. However, making corrections one at a time would certainly be tedious.

Objectives

At the end of this section, participants will be able to write statements which

- insert new records into a table
- update existing records in a table
- remove records from a table
- control data manipulation through the use of transactions to save and undo changes to data.

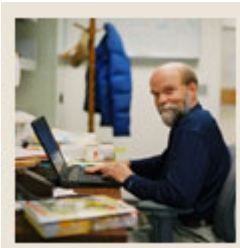


Section I: Insert, Update and Delete

Lesson: Overview (Continued)

Jump to TOC

Section contents



Section I: Insert, Update and Delete

Lesson: Insert

Jump to TOC

Purpose

Add a row to a table or a view's base table.

```
INSERT INTO table [view]([column1] [,column2] … )
     VALUES (expr1 [,expr2] …);

SQL> INSERT INTO swvcrse (swvcrse_crn, swvcrse_desc,
                          Swvcrse_activity_date)
            VALUES  (10025, 'ENGLISH', SYSDATE);
```

Abbreviated version

If all the columns of a table are being inserted in a table, the columns can be omitted from the statement. The above statement might be abbreviated:

```
SQL> INSERT INTO swvcrse
           VALUES  (10025, 'ENGLISH', SYSDATE);
```

Note: Omitting column references is a great shortcut when doing hands-on manipulations, but should never be used in a stored procedure. If the definition of a table changes in the future, the stored statements without column references will become invalid.

Insert via subquery

Inserts can be accomplished using a subquery from another table. The same number of rows returned from the subquery will be inserted into the table.

In the following example, we want to insert students into the SWRSTDN table for those that have an average course GPA of 3.5 or higher (honor students, so they will be marked with 'HS').

```
SQL> INSERT INTO    swrstdn (swrstdn_pidm, swrstdn_stdn_code,
                             Swrstdn_stdn_date, swrstdn_activity_date)
     SELECT     swrregs_pidm,
                'HS',
                '01-JAN-06',
                SYSDATE
       FROM     swrregs
    GROUP BY    swrregs_pidm
      HAVING    AVG(swrregs_gpa) >3.5;
```

Section I: Insert, Update and Delete

Lesson: Multi-table Insert

Jump to TOC

Multi-Table Inserts

Date may be inserted into multiple tables at one time using a special version of the INSERT statement. There are conditional and unconditional muli-table inserts.

NOTE: The order of the tables into which Oracle inserts data is not determinate. Therefore, before issuing a multi-table insert statement, especially in parent/child relationships, you should defer any constraints and disable any triggers that depend on a particular table order for the multi-table insert operation.

If using a sequence in a multi-table insert, supply the sequence information in the Values clause and not the Select Subquery.

Unconditional Multi-Table Inserts

The following syntax can be used to perform an unconditional multi-table insert. Records will be inserted into all tables specified.

```
INSERT ALL
         INTO table1 [view] ([column1] [,column2] …
       VALUES (expr1 [,expr2] …)
         INTO table2 [view] ([column1] [,column2] …
       VALUES (expr1 [,expr2] …)
       . . .
       SELECT <column_name> <constant> . . .
         FROM <table_name>;
```

Section I: Insert, Update and Delete

Lesson: Multi-table Insert (Continued)

Jump to TOC

Conditional Multi-Table Inserts

To insert data into multiple tables based on certain conditions, the following syntax is used:

```
INSERT ALL | FIRST
    WHEN  condition1 THEN
           INTO    table1 [view] ([column1] [,column2] …
           VALUES  (expr1 [,expr2]…)
    WHEN  condition2 THEN
           INTO  table2 [view] ([column1] [,column2] …
         VALUES  (expr1 [,expr2] …)
    [ELSE
          INTO   table99 [view] ([column1] [,column2] …
        VALUES   (expr1 [,expr2] …)]
    SELECT  <column_name> <constant>
   FROM  <table_name>;
```

If the keyword ALL is used, all the WHEN conditions will be evaluated for each row returned from the Select statement.

If the keyword FIRST is used, the WHEN conditions will be evaluated until a true condition is found. After the first true condition is found, the remaining WHEN statements are skipped.

The ELSE condition is optional and if the row does not meet any of the WHEN conditions, the ELSE clause will be executed. If no ELSE clause is supplied and the row does not match any of the WHEN conditions, no action is taken on that row.

Section I: Insert, Update and Delete

Lesson: Insert – Default in Values

Jump to TOC

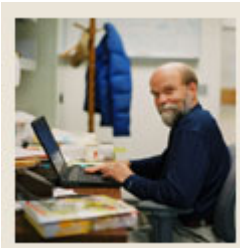
DEFAULT in Values

When issuing an INSERT INTO and VALUES clause, the keyword DEFAULT may be used to specify that Oracle should use the default value assigned for the column, if defined.

```
INSERT INTO table 1 VALUES (1, DEFAULT);
```

If no default value has been defined for the column, null inserted.

They keyword DEFAULT is available only in the VALUES clause and will work with standard and multi-table inserts.



Section I: Insert, Update and Delete

Lesson: Update

Jump to TOC

Purpose

Change existing column values within a table or in a view's base table.

```
UPDATE table [view]
   SET column = expr [,column = expr] […]
[WHERE condition ];

SQL> UPDATE swrtest
       SET swrtest_sat_math = 490
     WHERE swrtest_pidm = 12340
       AND swrtest_test_date='01-MAR-05';
```

Subqueries

Subqueries may also be used in the update WHERE condition.

```
SQL>  UPDATE swrstdn
        SET swrstdn_stdn_code = 'HS'
      WHERE swrstdn_pidm in
           (  SELECT swrregs_pidm
                FROM swrregs
             GROUP BY swrregs_pidm
               HAVING AVG (swrregs_gpa) > 3.5);
```

DEFAULT in Values

When issuing an UPDATE, the keyword DEFAULT may be used to specify that Oracle should use the default value assigned for the column, if defined.

```
UPDATE table1
   SET column1 = DEFAULT
 WHERE condition;
```

If no default value has been defined for the column, null is used.

Section I: Insert, Update and Delete

Lesson: Merge

Jump to TOC

Purpose

The MERGE statement acts like a combination insert and update statement. If the ON condition is met indicating a matching record exists, the UPDATE path is followed, otherwise the INSERT path is followed.

```
MERGE INTO table1
  USING table2 | view
  ON (table1.column1 = table2.column1 …)
WHEN MATCHED THEN
   UPDATE SET table1.column3 = {expr1} …
WHEN NOT MATCHED THEN
   INSERT (table1.column1, table1.column2 …)
   VALUES (table2.column1, {expr1}  …);
```

Restrictions

You cannot upate a column referenced in the ON clause. Both INSERT and UPDATE statements are required.

Section I: Insert, Update and Delete

Lesson: Merge (Continued)

Jump to TOC

Merge in Oracle 10G

Oracle (10g) has made some enhancements to the merge statement.

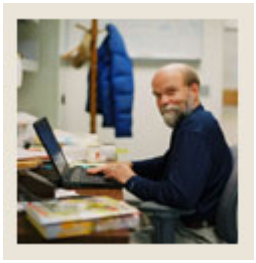
1. There can be just an insert or just and update statement – both are no longer required
2. A WHERE clause may be added to an insert or update to restrict certain rows from being updated or inserted.
3. The join of source to target (ON Clause) can be eliminated for a merge with only an INSERT statement, allowing you to insert all rows from the source into the target:

```
MERGE INTO swriden_history
USING swriden
ON (1 = 0)
WHEN NOT MATCHED THEN
INSERT
VALUES (swriden.swriden_pidm, swriden.swriden_id,
      swriden.swride_last_name,
     Swriden.swriden_first_name, swriden.swriden_mi,
      swriden.swriden_change_ind)
WHERE swriden_change_ind is null;
```

4. DELETE clause available with the WHEN MATCHED or UPDATE clause (not available on the WHEN NOT MATCHED or INSERT clause). DELETE must have a WHERE clause.

```
MERGE INTO table1
  USING table2 | view
  ON (table1.column1 = table2.column1 …)
WHEN MATCHED THEN
   UPDATE SET table1.column3 = {expr1} …
   DELETE WHERE (table1.columnX = expr1 …)
WHEN NOT MATCHED THEN
   INSERT (table1.column1, table1.column2 …)
   VALUES (table2.column1, {expr1}  …);
```

5.



Section I: Insert, Update and Delete

Lesson: Delete

Jump to TOC

Delete

Deleting rows from a table is quite simple; in fact, it is too simple. Consider the following syntax:

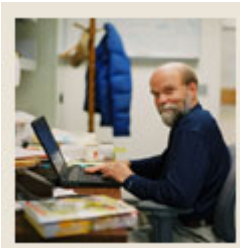
```
DELETE   [FROM]     <table>
[WHERE   <column_name> condition];

SQL>      DELETE    FROM swrtest
           WHERE    swrtest_pidm = 12341
             AND    swrtest_sat_verbal = 530
             AND    swrtest_sat_math = 580;
```

Deleting entire tables

What will occur if the delete statement is run without the optional WHERE clause, or replacing <table> with swriden?

```
SQL> DELETE swriden;

12 rows deleted.

SQL> SELECT * FROM swriden;

no rows selected
```

A good way to make sure you are deleting the right number of rows is to write a select statement with the same WHERE criteria as the delete statement. If the select statement returns the correct data, change the SELECT portion of the statement to DELETE and use the same WHERE.



Section I: Insert, Update and Delete

Lesson: Transactions

Jump to TOC

What is a transaction?

A transaction is defined as a change to the database since the last COMMIT.

Commit

Makes permanent the changes made to the database. While working in SQL, changes can be viewed after a command is run on the database. For instance, if a person was deleted from the swriden table while in SQL*Plus, you could view these changes. This is called an implied commit. This does not mean that the change has been made permanent to the database.

To make the change permanent, use the COMMIT command.

```
SQL> DELETE    FROM  swriden
               WHERE swriden_last_name = 'JONES';

SQL> COMMIT;
```

Note: Oracle recommends that every transaction end explicitly with a COMMIT before disconnection from Oracle. If a program terminates abnormally, the last uncommitted transaction is rolled back. **A normal exit from most Oracle applications causes the current transaction to be committed.**

Rollback

Use ROLLBACK to undo work within the current transaction.

```
ROLLBACK [TO SAVEPOINT <savepoint>];

SQL> DELETE     FROM  swriden
                WHERE  swriden_last_name = 'JONES';

SQL> ROLLBACK;
```

You must issue a ROLLBACK before you issue a COMMIT, before you exit SQL*Plus, or before you issue a DDL statement (CREATE/ALTER/DROP discussed later).

Savepoint

Identifies a point in the current transaction to which you can later roll back.

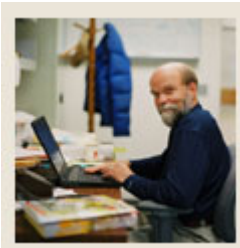
```
SAVEPOINT <savepoint>
```

```
SQL>     DELETE FROM     swriden
              WHERE      swriden_last_name = 'JONES';

SQL>     SAVEPOINT sp1;

SQL>     DELETE FROM     swriden
               WHERE     swriden_last_name = 'SMITH';

SQL>     SAVEPOINT sp2;

SQL>     ROLLBACK TO SAVEPOINT sp1;
```

At this point the SMITH delete has been "rolled back" or the data restored. If you were to commit at this point, the only rows deleted woul be the JONES records. You may do additional processing or roll back the entire transaction.

If you want to rollback not to just one savepoint, but roll back the entire transaction, issue the ROLLBACK statement without any savepoint parameter. All statements at all savepoints will be rolled back.

Re-using a savepoint name moves the savepoint to the new location. If you re-use a savepoint name you cannot roll back to the old location.



Section I: Insert, Update and Delete

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

Exercise 1

Insert a new student in the **SWRIDEN** table using your own name, PIDM 2045 and ID 432G. Do not use a middle name. (HINT: Make sure you are providing data to all required columns – check for NOT NULL columns)

Exercise 2

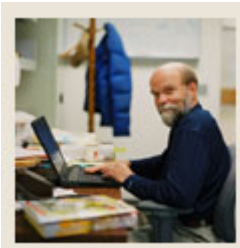
Add a new student profile record in **SWBPERS** for the new student added in Exercise 1, using the following information:

Activity Date	Current system date
Social Security Number	124-62-8747
Birth Date	Unknown (leave null)
Marital Code	Unknown (leave null)
Sex	Female
Confidential Indicator	Y

Note: Check the description of the table for column size constraints.

Exercise 3

Create a savepoint named SP1.



Exercise 4

Insert another row into the **SWRIDEN** table, but prompt the operator for each variable except for the activity date.

Exercise 5

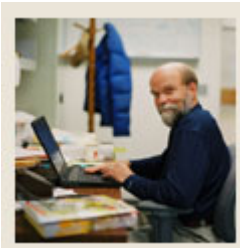
Update the new student profile record created in Exercise 2 so that the Social Security number is 635-56-1525 and the marital code is 'S' (**SWBPERS)**.

Exercise 6

Roll back to savepoint SP1.

Exercise 7

Commit your changes.



Exercise 8

Delete the student profile record created in Exercise 2 (**SWBPERS)**.

Exercise 9

Delete the **SWRIDEN** record you created in Exercise 1.

Exercise 10

Commit your changes.

Section J: Creating and Maintaining Tables and Indexes

Lesson: Overview

Jump to TOC

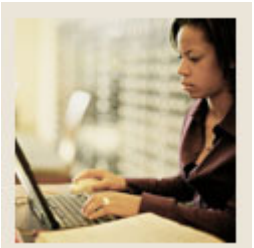
Introduction

In the previous sections, we have both retrieved and manipulated data from tables. A table is a database object within a schema. In this section, we will discuss how to create and maintain tables and get introduced to other schema objects that can be created.

Objectives

At the end of this section, participants will be able to create, maintain, and secure:

- Tables
- Indexes



Section J: Creating and Maintaining Tables and Indexes

Lesson: Overview (Continued)

Jump to TOC

Section contents

Section J: Creating and Maintaining Tables and Indexes

Lesson: Schemas

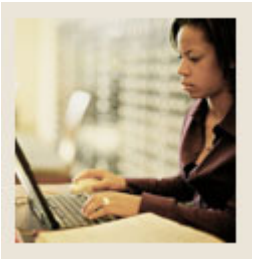
Jump to TOC

Schemas

Schemas may contain the following types of objects:

- Tables
- Views
- Clusters+
- Database links+
- Stand-alone stored functions and procedures
- Indexes
- Packages
- Database triggers+
- Sequences
- Snapshots+
- Profiles+
- Roles+
- Rollback segments+
- Tablespaces+

+ These objects are not discussed in this manual; refer to Oracle's SQL language reference manual.



Section J: Creating and Maintaining Tables and Indexes

Lesson: Data Definition Language Commands

Jump to TOC

DDL commands

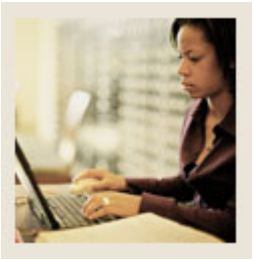
Data Definition Language (DDL) commands allow you to

- create, alter, and drop objects
- grant and revoke privileges and roles
- establish auditing options
- add comments to the data dictionary.

Implicit COMMITs

Oracle implicitly commits the current transaction before and after every Data Definition Language statement. Consideration to database manipulations should be considered before using DDL statements.

For a complete listing of DDL commands, see the Oracle SQL Language Reference Manual.



Section J: Creating and Maintaining Tables and Indexes

Lesson: Creating a Table

Jump to TOC

Naming a table or view

- Must begin with a letter, A-Z or a-z
- May contain letters, numbers, and the special character _ (underscore). The characters $ and # are also legal, but their use is discouraged
- Case-insensitive; e.g., grades, GRADES, and GrAdEs are all the same table
- May be up to 30 characters in length
- May not duplicate the name of another table or view under the same schema
- May not duplicate an Oracle reserved word

Name	Valid?
NATION	Yes
1CONTINENT	No; doesn't begin with a letter.
NORTH_AMERICA	Yes
UPDATE	No; Oracle reserved word.
TABLE1	Yes; but poor design for naming conventions.

Naming a column

Column names follow the same rules as those for table names. Columns with exactly the same name in separate tables can be ambiguous, however. For instance, joining two tables both containing "pidm" columns requires the use of TableName.ColumnName notation.

To avoid this ambiguity, Banner has appended the TableName to the front of each column name. For example, columns in the SWRIDEN table appear as:

- swriden_pidm
- swriden_last_name
- swriden_first_name

…

Not only does this naming convention remove the ambiguity of the column's table, but it makes for very readable code. This is one of several conventions found within a Banner table's design. As you begin to explore your system's tables and their structure, you will begin to recognize many other standards and conventions.

Note: Give your tables and columns meaningful names. You have up to 30 characters - why not use them?

Section J: Creating and Maintaining Tables and Indexes

Lesson: Creating a Table (Continued)

Jump to TOC

Data type

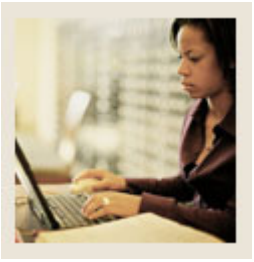
- VARCHAR2(n)
  Variable length character string with a maximum length, *n*, of 4000 bytes.
- CLOB/BLOB/BFILE
  Variable length character string containing up to 4 gigabytes, or $2^{31} - 1$ bytes.
- NUMBER(p,s)
  Numeric data type having a precision *p* and scale *s*.
- DATE
  Valid dates range from Jan 1, 4712 BC to Dec 31, 9999 AD.

Syntax

```
CREATE TABLE [schema.]table
( { column data type  [DEFAULT expr] [column_constraint] |
table_constraint }
[ , { column data type  [DEFAULT expr] [column_constraint]
table_constraint}   ] … )
                    [ AS subquery ]
```

Example

```
SQL>  CREATE TABLE hobbies
   (hobbies_pidm              NUMBER(8) NOT NULL,
    hobbies_type_code         VARCHAR2(10) DEFAULT 'UK',
    hobbies_desc              VARCHAR2(100) DEFAULT 'UNKNOWN',
    hobbies_how_long          NUMBER(3),
    hobbies_yearly_cost       NUMBER(11,2),
    hobbies_solo_group        VARCHAR2(2),
    hobbies_activity_date     DATE DEFAULT SYSDATE);
```



Section J: Creating and Maintaining Tables and Indexes

Lesson: Altering a Table

Jump to TOC

Methods

A table can be altered via any of the following methods:

- Add a column
- Redefine a column (data type, size, default value, change name)
- Add an integrity constraint
- Enable, disable, or drop an integrity constraint or trigger
- Rename table

Syntax

```
ALTER TABLE [schema.]table
{ [ ADD ( { column_element | column_constraint}
  [, column_element | column_constraint}  ] …) ]
  [ MODIFY  ( column_element [, column_element] …) ]
  [ DROP drop_clause ] …
  [ ENABLE enable_clause ] …
  [ DISABLE  disable_clause ] …
  [ RENAME TO new_table_name ]
  [ RENAME COLUMN old_column_name TO new_column_name ]
```

ALTER TABLE

```
  SQL> ALTER TABLE twraccd
               ADD (twraccd_effective_date DATE);
```

OR

```
  SQL> ALTER TABLE twraccd
               ADD (twraccd_effective_date DATE DEFAULT SYSDATE);
```

Removing a column

```
  SQL> ALTER TABLE <table_name> DROP COLUMN <column_name>;
```

Example:

```
       ALTER TABLE twraccd
        DROP COLUMN twraccd_effective_date;
```

Mark Column Unused

You can also mark a column as unused. This removes it from the describe command and from any select, insert or update statement. Essentially, it is no longer available for any use.

ALTER TABLE <tablename> SET UNUSED COLUMN <column_name>;

The difference between marking a column unused and dropping it is that an unused column still retains the disk space associated with it. Even though the disk space is still considered in use, you cannot re-instate the column. You can, however, remove the associated disk space by issuing the following command:

ALTER TABLE <tablename> DROP UNUSED COLUMNS <column_names>;

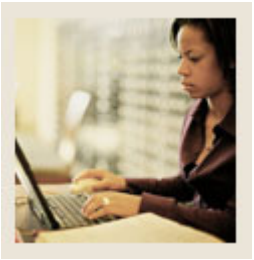
This method may be used for very large tables where you may want to remove a column from use, but not lock the table for the lengthy period it would take to remove the associated storage. The storage removal can be scheduled for the evening or any time where the table has little or no use.

Redefining a column

```
SQL>     ALTER TABLE swriden
              MODIFY (last_name VARCHAR2 (30));
Table altered.
```

OR

```
SQL>     ALTER TABLE swriden
              MODIFY (last_name VARCHAR2 (30) DEFAULT 'NONE');
Table altered.
```



Section J: Creating and Maintaining Tables and Indexes

Lesson: Constraints

Jump to TOC

Purpose

Constraints define the conditions under which data is valid.

The table_constraint syntax is a part of the table definition. An integrity constraint defined with this syntax can impose rules on any columns in the table. The table constraint syntax may appear in a CREATE TABLE or ALTER TABLE statement. The syntax can define any type of integrity constraint except a NOT NULL constraint.

The column_constraint syntax is part of a column definition. In most cases, an integrity constraint defined with this syntax can only impose rules on the column in which it is defined.

Column_constraint syntax that appears in a CREATE TABLE statement can define any type of integrity constraint. Column_constraint syntax that appears in an ALTER TABLE statement can only define or remove a NOT NULL constraint. To modify an integrity constraint, you must drop the constraint and redefine it.

NOT NULL constraint

The NOT NULL constraint specifies that a column cannot contain a null value. If you do not specify this constraint, the default is NULL.

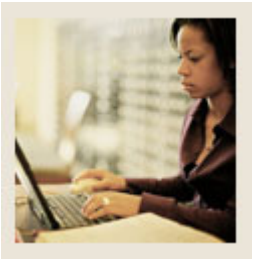
```
SQL> ALTER TABLE hobbies
        MODIFY (hobbies_type_code  NOT NULL);
```

CHECK constraint

The CHECK constraint explicitly defines a condition. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to NULL).

Syntax: `CONSTRAINT constraint_name CHECK (condition)`

```
SQL> ALTER TABLE twraccd
  ADD CONSTRAINT check_trans_type
           CHECK (twraccd_trans_type IN ('C', 'P'));

SQL> ALTER TABLE twraccd
  ADD CONSTRAINT check_amount
           CHECK (twraccd_amount <> 0);
```



PRIMARY KEY constraint

A PRIMARY KEY constraint designates a column or combination of columns as the table's primary key. To satisfy a PRIMARY KEY constraint, both of these conditions must be true:

- No primary key value can appear in more than one row in the table
- No column that is part of the primary key can contain a null

A table can have only one primary key.

```
SQL>     ALTER TABLE swvterm
           ADD CONSTRAINT PK_swvterm PRIMARY KEY
                  (swvterm_term_code);
```

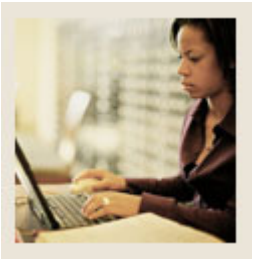
If the column(s) used to create the primary key had previously been defined without the NOT NULL constraint, Oracle will automatically assign NOT NULL constraints to those columns.

UNIQUE constraint

The UNIQUE constraint designates a column or combination of columns as a unique key. To satisfy the condition, no two rows in the table can have the same value for the unique key. You cannot designate the same column or combination of columns as both a unique key and the primary key. Although you can have only one primary key for a table, a table can have several unique keys.

```
SQL>     ALTER TABLE twvdetc
           ADD CONSTRAINT unq_detc_code UNIQUE
                   (twvdetc_code);
```

Unique constraints differ from primary key constraints in that they allow NULL values in the column(s) defined in the constraint. Because of this difference, you may have multiple rows with NULL values in columns that are part of a UNIQUE constraint.



Section J: Creating and Maintaining Tables and Indexes

Lesson: Referential Integrity Constraints

Jump to TOC

Purpose

A referential integrity constraint designates a column or combination of columns as a foreign key, and establishes a relationship between that foreign key and a specified primary or unique key called the referenced key. In this relationship, the table containing the foreign key is called the child table, and the table containing the referenced key is called the parent table.

Conditions

To satisfy a referential integrity constraint, the following conditions must be met:

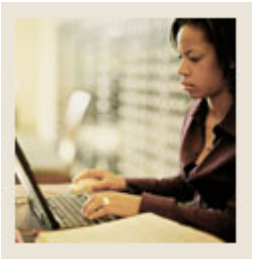
- The child and parent tables must be in the same database
- The value of the row's foreign key must appear as a referenced key value in one of the parent table's rows. The row in the child table is said to depend on the referenced key in the parent table

Keywords

A referential integrity constraint is defined in the child table. A referential integrity constraint definition can include any of these keywords:

- **Foreign key**
  Identifies the column or combination of columns in the child table that makes up the foreign key. Only use this keyword when defining a foreign key with a table constraint clause.
- **References**
  Identifies the parent table and the column or combination of columns that make up the referenced key. If you only identify the parent table and omit the column names, the foreign key automatically references to the primary key of the parent table. The referenced key columns must be of the same number and data types as the foreign key columns.
- **On delete cascade**
  Allows deletion of referenced key values in a parent table that have dependent rows in a child table. This causes Oracle to automatically delete dependent rows from the child table to maintain referential integrity. If you omit this option, Oracle forbids deletion of referenced key values in the parent table that have dependent rows in the child table.

  WARNING: ***NEVER*** create a table using this integrity constraint unless it is directly applicable to your business rules and applications.



Section J: Creating and Maintaining Tables and Indexes

Lesson: Referential Integrity Constraints (Continued)

Jump to TOC

Defined constraints

Before defining a referential integrity constraint in the child table, the referenced UNIQUE or PRIMARY KEY constraint on the parent table must already be defined. Also, the parent table must be in your own schema or you must have REFERENCES privileges on the columns of the referenced key in the parent table. You cannot define a referential integrity constraint in a CREATE table statement that contains an AS clause. Instead, create the table without the constraint and add the constrain using the ALTER TABLE statement.

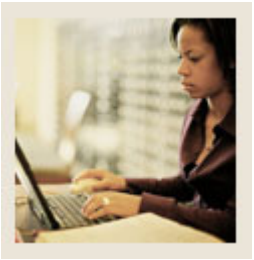
Note: You can define multiple foreign keys in a table. Also, a single column can be part of more that one foreign key.

```
SQL>    ALTER TABLE twraccd
          ADD CONSTRAINT FK1_twraccd_INV_swvterm_KEY
          FOREIGN KEY (twraccd_term_code)
          REFERENCES swvterm (swvterm_term_code);
```

Orphan Records

Adding a referential integrity constraint will fail if there are child records already in the table without parent records. Either:

- Add parent record OR
- Remove orphan child records



Section J: Creating and Maintaining Tables and Indexes

Lesson: Truncate

Jump to TOC

Purpose

TRUNCATE can be used to quickly remove all rows from a table.

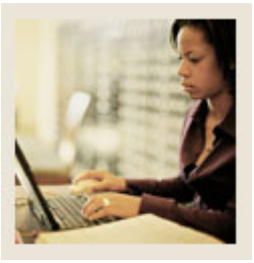
Removing all rows with the TRUNCATE command is faster than removing them with the DELETE command. No rollback information is created; thus the rows are permanently removed.

Syntax

```
TRUNCATE TABLE [schema.]<table>;
```

Example

```
SQL> TRUNCATE TABLE  old_scores;
```



Section J: Creating and Maintaining Tables and Indexes

Lesson: Indexes

Jump to TOC

Purpose

Indexes are database structures that boost the performance of queries. Indexes are used in conjunction with table columns. An index associates each distinct value of a column with the rows in a table that contain that value. The column that has the index is called the key column.

Uniqueness

Some indexes can be used to enforce uniqueness among the values in a column. Such an index is called a unique index. If a unique index is created, no two rows in the table may contain the same value in the indexed column.

A query that references an indexed column in its WHERE clause can use the index. When a query uses an index, Oracle searches the index for all the values that meet the condition specified by the WHERE clause. If the query selects only the indexed column, the query can read the indexed column values directly from the index rather than from the table.

ROWIDs

For each value, the index also identifies the locations, or ROWIDs, of rows in the table having that value. If the query selects data in addition to the indexed value, Oracle finds the rows in the table based on the ROWIDs. Searching by ROWID is the fastest way for Oracle to locate a single row.

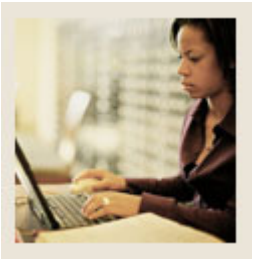
When to use indexes

Indexes improve the performance of queries that select a small percentage of rows from a table. As a general guideline, you should use indexes for queries that select less than 20% or 25% of table rows.

Be selective in the number of indexes created on a table as they can slow performance.

Full table scans

If a query does not use an index, Oracle must perform a full table scan, reading all rows of a table sequentially. Oracle examines each row to determine whether it meets the criteria of the query's WHERE clause. Indexed queries can be considerably faster than finding the row with a full table scan; however, a query that selects more than 20% or 25% of a table's rows may be performed faster by a full table scan than by an indexed query.



Choosing columns to index

- Index columns that are used frequently in WHERE clauses
- Index columns whose MAX and MIN values are selected frequently
- Index columns that are used frequently to join tables in SQL statements
- Index columns with high selectivity. Selectivity is high if few rows have the same value in the key column. Unique indexes are the most selective and the most effective in optimizing query performance
- Do not index columns with few distinct values.** Such columns have low selectivity
- Do not index columns in small tables. If a table uses fewer than 5 data blocks, a full table scan may return rows faster than an indexed query. You can determine how many data blocks a table uses by examining the ROWIDs of the table's rows. For example, this query returns the number of blocks used by the swrtest table:

```
SQL> SELECT COUNT (DISTINCT(SUBSTR(ROWID,1,8)||SUBSTR(ROWID,15,4)))
       FROM swrtest;
```

Do not index columns that are frequently modified using the UPDATE, INSERT, and DELETE statements. These statements not only update the rows in the table; they must also update the index as well.

**** Histograms**

There are times when an index on a column with low cardinality (few distinct values) is necessary and desirable. Such columns might contain True/False or Yes/No or other such values where Oracle assumes an even distribution of records to each value.

For example, if you know your data has significantly more Yes values than No values, and you frequently select rows with the value No, you may want to index the column. To ensure the index on that column is used, it must also contain a histogram which tells Oracle that the distribution of No values is much lower than Yes values, which makes the index more cost effective.

This topic is covered more extensively in the advanced Oracle courses like the DBA courses or SQL Tuning courses.

Syntax

To create an index after you have created the table:

```
CREATE [UNIQUE] INDEX [schema.]index
     ON [schema.]table (column, [column,] [column,] …)
```

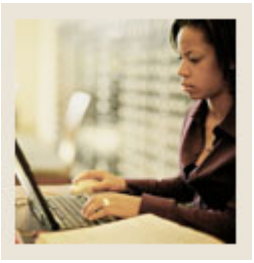
Primary key and Unique indexes can be created at the same time as you create a table:

```
CREATE TABLE new_table (
     KEY_VALUE      VARCHAR2 (50) NOT NULL,
     INDICATOR      CHAR(1) NOT NULL,
     DESCRIPTION    VARCHAR2(400),
     ACTIVITY_DATE  DATE DEFAULT SYSDATE
                     CONSTRAINT nn_activ_date NOT NULL,
     CONSTRAINT PK_NEW_TABLE PRIMARY KEY (KEY_VALUE));
```

OR

```
CREATE TABLE mytable
 (id NUMBER
     PRIMARY KEY USING INDEX
     (CREATE INDEX pk_myind ON mytable(id)),
  identifier  NUMBER,
  information VARCHAR2(30),
  CONSTRAINT mytableunique
      UNIQUE (identifier)
      USING INDEX (CREATE INDEX mytableidx ON
                   mytable(identifier)));
```

When creating Primary Key or Unique constraints, Oracle will automatically create a unique index for the columns in the Primary Key or Unique constraints even if not explicitly specified.



Section J: Creating and Maintaining Tables and Indexes

Lesson: Concatenated Indexes

Jump to TOC

Purpose

An index can be made up of more than one column. Such an index is called a concatenated index.

Concatenated indexes are useful in providing selectivity. Sometimes two columns with low selectivity can be combined to produce a concatenated index with high selectivity. If all the selected columns are included in a concatenated index, the query can be satisfied entirely by an index search and avoid access to the table altogether. The columns that make up a concatenated index are referred to as the concatenated key.

SQL statements and concatenated indexes

Whether or not a SQL statement uses a concatenated index is determined by the column contained in the WHERE clause of the SQL statement and the order of the columns in the CREATE INDEX statement. A query can only use a concatenated index if it references a leading portion of the index in the WHERE clause. The leading portion of a concatenated index refers to the first column specified in the CREATE index statement.

Columns

An index can contain a maximum of 16 columns. You can create several indexes on different columns (or different combinations of columns) in the same table. Oracle imposes no limits on the number of indexes you can create on a single table.

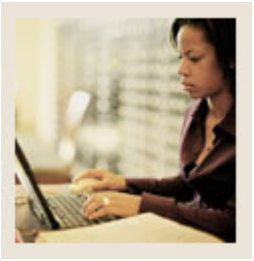
Syntax

```
CREATE [UNIQUE] INDEX [schema.]index
ON [schema.]table (column, [column,] [column,] …)
```

Ordering columns in concatenated indexes

If only one column of the concatenated index is used frequently in WHERE clauses, place that column first in the create INDEX statement.
If more than one column is used frequently in WHERE clauses, place the most selective column first in the CREATE INDEX statement.

```
SQL> CREATE INDEX pidm_term_index
           ON twraccd (twraccd_pidm, twraccd_term_code);

SQL> CREATE UNIQUE INDEX swriden_key_index
                 ON swriden (swriden_pidm, swriden_id,
                             swriden_last_name, first_name,
                             swriden_mi, swriden_change_ind);
```



Section J: Creating and Maintaining Tables and Indexes

Lesson: Comments and Data Dictionary Views

Jump to TOC

Object Comments

Comments can be placed on tables, views and columns in the database to provide internal documentation on what these objects are used for, and also aid developers and report writers in determining which table or columns to use to manipulate or retrieve data.

To add a comment to a table:

```
SQL> comment on table <table_name> is '<enter your comment here>';
```

To add a comment on a column:

```
SQL> comment on column <table_name>.<column_name> is '<enter your
comment here>';
```

Data Dictionary Views

There are several sets of data dictionary views in the database that will help you find information about an object.

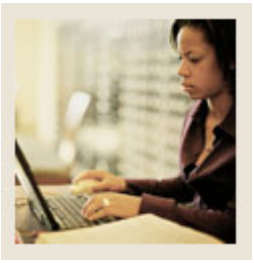
USER_% - information on objects you OWN
ALL_% - information on objects you own PLUS objects owned by other people that you can see
DBA_% - information about all objects in the database (access restricted to users with high level privileges like DBAs).

Typical views include

▪ USER_TABLES	▪ USER_TAB_COLUMNS
▪ USER_INDEXES	▪ USER_IND_COLUMNS
▪ USER_CONSTRAINTS	▪ USER_CONS_COLUMNS
▪ USER_TAB_COMMENTS	▪ USER_COL_COMMENTS

To find other USER_ views in the data dictionary:

```
SQL> select view_name from all_views where view_name like 'USER%';
```



Section J: Creating and Maintaining Tables and Indexes

Lesson: Self Check

Jump to TOC

Exercise 1

To make data retrieval faster, create an index for PIDM on **SWRIDEN**.

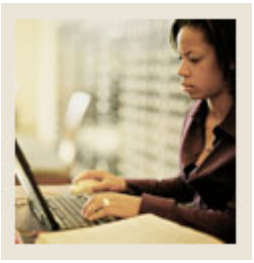
Exercise 2

Create a relationship between the validation table T**WVDETC** and the repeating table **TWRACCD.** TWVDETC should have the primary key on TWVDETC_CODE and TWRACCD should have the foreign key on TWRACCD_DETC_CODE.

Exercise 3

Create a table called **TEMP_XX** (where XX is your user number) with the following structure:

MYNUMBER	NUMBER (8)
TEXT	VARCHAR2 (30)
MYDATE	DATE
MESSAGE	VARCHAR2 (50)



Section J: Creating and Maintaining Tables and Indexes

Lesson: Self Check (Continued)

Jump to TOC

Exercise 4

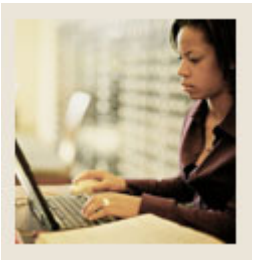
Add a column called SWRADDR_COUNTRY_CODE, type VARCHAR2(10) NOT NULL to the swraddr table.

What happens when you try to add a NOT NULL column to an existing table? How might you fix it?

Exercise 5

Add a constraint on the SWRREGS table to the SWVCRSE table on CRN. What happens? Correct the problem and try again.

Why will the constraint still not create? How would you fix this problem (Hint: Use a minus query to identify CRN records that do not match)?

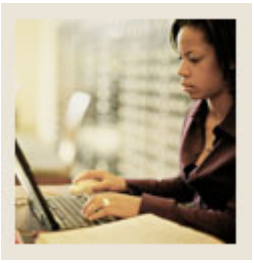


Exercise 6

Add a table comment for the SWRIDEN, SWRADDR, and SWBPERS tables.

Exercise 7

Locate your new indexes and constraints in the user_indexes, user_constraints, and user_cons_columns data dictionary views.



Section K: Creating and Maintaining Other Database Objects

Lesson: Overview

Jump to TOC

Introduction

In the previous sections, we learned how to create and maintain tables and indexes. These are only two of the many types of Objects that make up a database. In this section we will explore other objects and discuss security on objects.

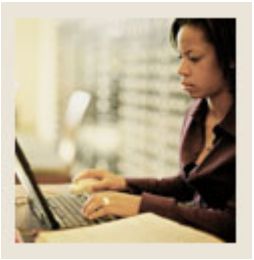
Objectives

At the end of this section, participants will be able to create and maintain:

- Views
- Sequences
- Synonyms

Security on objects will also be discussed.

Section Contents



Section K: Creating and Maintaining Other Database Objects

Lesson: Creating Views

Jump to TOC

Views

A view is a logical table that allows you to access data from other tables and views. A view contains no data itself. The tables upon which a view is based are called base tables.

Purpose

Views are used to:

- provide an additional level of table security, by restricting access to a predetermined set of rows and/or columns of a base table
- hide data complexity. A view may be used to act as one table when actually several tables are used to construct the results
- present data from another perspective. For example, views provide a means of renaming columns without actually changing the base table's definition

Syntax

```
CREATE [OR REPLACE] [FORCE | NOFORCE]
  VIEW [schema.]view [ (alias  [, alias] … ) ]
  AS subquery
 [WITH CHECK OPTION [CONSTRAINT constraint]]
```

- **OR REPLACE**
  Recreates the view if it already exists. Use this option to change the definition of an existing view without dropping, recreating, and regranting object privileges previously granted on it.
- **FORCE | NOFORCE**
  FORCE creates the view regardless of whether the view's base tables already exist within the owner's schema or the owner of the view has privileges to the base tables.
  NOFORCE creates this view only if the base tables exist within the owners schema or the owner of the view has privileges to the base tables.
- **WITH CHECK OPTION**
  Specifies that inserts and updates performed through the view must result in rows that the view query can select.

Constraint

The name assigned to the CHECK OPTION constraint. If the constraint is omitted, then Oracle automatically assigns the constraint a name of the form, SYS_cn, where *n* is an integer that makes the constraint name unique within the database.

SWVTELE

In our training exercises, there exists one view, **swvtele**.

```
SQL> SELECT * FROM swvtele;

PIDM           NAME                    PHONE
---------      --------------------    ---------
12340          Brown, Julie            (610) 562-4789
12341          Smith, Robert           (215) 795-4323
12342          Johnson, Peter          (610) 562-4789
12343          Jones-Erickson, Sandy   (610) 324-6734
12344          Erickson, Ralph         (850) 674-3213
```

Script

The view was created using the following script:

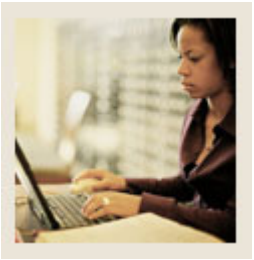
```
SQL> CREATE OR REPLACE VIEW swvtele
         (swvtele_pidm, swvtele_name, swvtele_phone)
       AS SELECT swriden_pidm, swriden_last_name||
                 ', '||swriden_first_name||' '|| swriden_mi,
                 '('||swraddr_phone_area||')'
                 || SUBSTR(swraddr_phone_number,1,3)
                 ||'-'|| SUBSTR(swraddr_phone_number,4,4)
            FROM swriden, swraddr
           WHERE swriden_pidm = swraddr_pidm
             AND swriden_change_ind IS NULL;
```

Check option

```
SQL> CREATE OR REPLACE VIEW SWVSUSP
               (SWVSUSP_PIDM, SWVSUSP_STDN_CODE,
                SWVSUSP_STDN_DATE, SWVSUSP_ACTIVITY_DATE)
     AS SELECT SWRSTDN_PIDM, SWRSTDN_STDN_CODE,
               SWRSTDN_STDN_DATE, SWRSTDN_ACTIVITY_DATE
        FROM SWRSTDN
         WHERE SWRSTDN_STDN_CODE = 'SS'
        WITH CHECK OPTION;
```

With the check option on this view, users will get the following error when trying to change a student's standing:

```
SQL> UPDATE    swvsusp
        SET    swvsusp_stdn_code = 'GS';
*
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```



Section K: Creating and Maintaining Other Database Objects

Lesson: Synonyms

Jump to TOC

Purpose

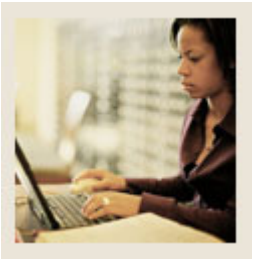
Synonyms are used for security and convenience. Creating a synonym for an object allows you to reference the object without specifying its owner (if the synonym is public) provide another name for the object.

Syntax

```
CREATE [PUBLIC] SYNONYM [schema.]synonym
               FOR [schema.]object
```

Example

```
SQL> CREATE PUBLIC SYNONYM standing
        FOR swrstdn;
```



Section K: Creating and Maintaining Other Database Objects

Lesson: Sequences

Jump to TOC

Purpose

A sequence is a database object from which multiple users may generate unique integers. You can use sequences to automatically generate primary keys.

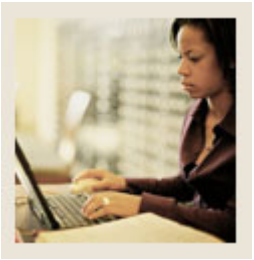
Syntax

```
CREATE SEQUENCE [schema.]sequence
    [INCREMENT BY integer]
    [START WITH integer]
    [MAXVALUE integer | NOMAXVALUE]
    [MINVALUE integer | NOMINVALUE]
    [CYCLE | NOCYCLE]
    [CACHE integer | NOCACHE]
    [ORDER | NOORDER]
```

- **INCREMENTED BY**
  Specifies the interval between sequence numbers. This value can be any positive or negative integer, but cannot be 0. If the increment is negative, the sequence descends. If the increment is positive, the sequence ascends. If omitted, the interval defaults to 1.
- **MAX & MIN VALUE**
  Specifies the sequence's minimum or maximum value. These integer values can have 28 or fewer characters. Ranges are 10^{27} to -10^{26}.
- **CYCLE**
  Specifies that a sequence will continue to generate values after reaching its MIN or MAX. The value generated is the MIN or MAX specification.
- **CACHE**
  Specifies how many values of the sequence Oracle preallocates and keeps in memory for faster access. The minimum value for this parameter is 2.
- **ORDER**
  Guarantees sequence numbers are generated in the order specified. (ONLY applies to Real Application Clusters or RAC)
- **START WITH**
  Specifies the beginning value generated by the sequence. The MIN and MAX values must be less than or equal to START WITH.

Example

```
SQL>  CREATE SEQUENCE pidm_seq
           START WITH 13000;
```



Accessing and incrementing

Once a sequence is created, you can access its value in SQL statements using the pseudo-columns CURRVAL and NEXTVAL. CURRVAL returns the current value of the sequence, NEXTVAL increments the sequence and returns the new value.

```
SQL> INSERT INTO swriden
       (swriden_pidm, swriden_id, swriden_last_name,
       swriden_first_name, swriden_mi, swriden_change_ind,
       swriden_activity_date)
     VALUES (pidm_seq.NEXTVAL, '254915791', 'McMahon', 'Stephen',
       'J', NULL, SYSDATE);
```

Section K: Creating and Maintaining Other Database Objects

Lesson: Security

Jump to TOC

Brief overview

System security is a complicated and lengthy topic and is beyond the scope of this course. However, there are a few minor aspects of security that should be discussed at this time in association to the creation of database objects.

GRANT privileges

Creating a table or view (or any database object), does not automatically grant other users the access to retrieve or manipulate data from those tables. The owner must explicitly GRANT privileges to other users within the database.

```
GRANT object_priv [ALL] [(column)]
   ON [schema.]object TO user [PUBLIC]
[WITH GRANT OPTION];
```

- **OBJECT_PRIV**
  An object privilege to be granted. You can substitute any of these values:

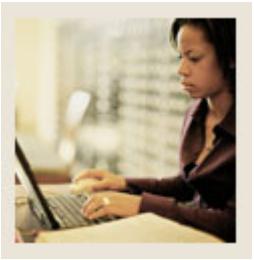
Object Privilege	Tables	Views	Sequences
ALTER	X		X
DELETE	X	X	
INDEX	X		
INSERT	X	X	
REFERENCES	X		
SELECT	X	X	X
UPDATE	X	X	

ALL
Grants all the privileges to the grantee
COLUMN
Specifies a table or view column on which privileges are granted. You can only specify columns when granting the INSERT, REFERENCES, or UPDATE privilege. If you do not list columns, the grantee has the specified privilege on all the columns in the table or view
ON
Identifies the object on which the privileges are granted. If you do not qualify object with schema, Oracle assumes the object is in your own schema
TO
Identifies users to which the object privilege is granted. PUBLIC grants object privileges to all users

WITH GRANT OPTION

Allows the grantee to grant the object privileges to other users.

```
SQL> GRANT SELECT
        ON swriden
        TO PUBLIC;

SQL> GRANT ALL
       ON swriden
       TO train01;

SQL> GRANT UPDATE (swriden_id, swriden_last_name, swriden_first_name,
                   swriden_mi, swriden_change_ind,
                   swriden_activity_date)
      ON  swriden
      TO  train02
    WITH GRANT OPTION;

SQL> GRANT SELECT, INSERT
       ON swriden
       TO train10;
```



Section K: Creating and Maintaining Other Database Objects

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

Exercise 1

Create a view called **SWVADDR_XX** ***(where XX is your*** user number) which contains a person's first name, last name (combine it into one column called name), city, state, and zip based on the **SWRIDEN** and **SWRADDR** tables.

Exercise 2

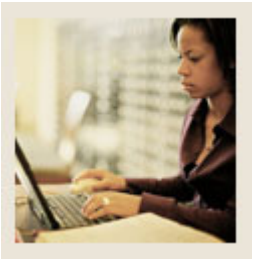
Retrieve all the rows from the new view.

Exercise 3

Grant the right to select from the view to a person sitting next to you. Make sure someone gives you the right to select from his/her new view.

Exercise 4

Try to select all columns from the view you were just granted access to in Exercise 3. What happened?



Exercise 5

Now, put the owner name in front of the table, in the syntax below. Did you get results?

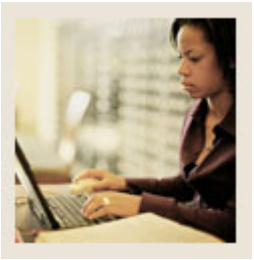
```
SELECT * FROM <owner.table_name>
```

Exercise 6

Because you have to specify the owner each time you are referring to the view, create a synonym to alleviate this.

Exercise 7

Select all columns from the view. You should not have to specify the owner in front of the view.



Section K: Creating and Maintaining Other Database Objects

Lesson: Self Check (Continued)

Jump to TOC

Exercise 8

Create a sequence which will be used to generate a new PIDM. Find out what the first value should be by finding the maximum existing PIDM +1. Insert a new row into the **SWRIDEN** using your sequence to generate the PIDM.

Section L: SQL*Loader & External Tables

Lesson: Overview

Jump to TOC

Introduction

SQL*Loader gives the capability to easily load data from a flat file to database tables. It is a great tool to use when converting existing legacy data into the Banner tables. Although we will cover only the basics of SQL*Loader in this section, the utility is quite powerful and has many options.

SQL*Loader can:

- load data from multiple datafiles of different file types
- handle fixed-format, delimited-format, and variable-length records
- manipulate data fields with SQL functions before inserting the data into database columns
- support a wide range of data types, including DATE, BINARY, PACKED DECIMAL, and ZONED DECIMAL, INTEGER
- load multiple tables during the same run, loading selected rows into each table packages
- combine multiple physical records into a single logical record
- treat a single physical record as multiple logical records
- generate unique, sequential key values in specified columns
- use the operating system's file or record management system to access datafiles
- load data from disk or tape
- provide thorough error reporting capabilities, so you can easily adjust and load all records
- use high-performance "direct" loads to load data directly into database files without Oracle processing

External Tables

External tables are data stored outside the Oracle database but referenced inside of Oracle as a standard Oracle table. External tables are discussed in this section as they use some of the syntax and functionality of the SQL*Loader product.

Intended audience

SQL is used for all types of database activities by many types of users. However, in order for attendees to receive the optimum benefit of this training, Sungard Higher Education recommends that prospective students come from one of the following groups:

- System administrators
- Database administrators
- Security administrators
- Application programmers
- Decision support system personnel

Section L: SQL*Loader & External Tables

Lesson: Overview (Continued)

Jump to TOC

Objectives

At the end of this section, participants will be able to

- define the basic file types which are required to load data
- invoke the SQL*Loader
- analyze output files for errors.
- describe external tables

Section contents

Section L: SQL*Loader & External Tables

Lesson: Required Input Files

Jump to TOC

Input files

In order to load data from a flat file into a table, there must be two files: a data file and a control file (or they can be combined into one file, as we will see later). The data file contains the data you want to load, and the control file specifies the format and the destination of the data.

The data file

There are two main data file types: fixed format files and variable format files.

Fixed format files contain records with fixed length, and the data fields in those records have fixed length, type, and position.

- Positioning of data is crucial because the position in the file is the only way to distinguish between the data items

Variable format files have records that are only as long as necessary to contain the data.

- The fields' positions and lengths are based on delimiters
  - Terminated fields - followed by a specified character (usually a comma)
  - Enclosed fields - both preceded and followed by specified characters (usually quotation marks). Used for strings that might contain spaces

The control file

The control file contains several kinds of information, which are necessary for loading the data (in bold in Control file format). The remaining clauses are all optional; they can be used to describe and manipulate the file data.

The control file uses the Data Definition Language (DDL), which describes:

- Data location
- Data format
- Column definition
- Data type mapping
- Field specifications

Considerations

Things to keep in mind:

- The name of the data field corresponds to the name of the table column into which the data is loaded
- The data type of the field tells SQL*Loader how to read the data in the datafile. It is not necessarily the same as the column data type
- Data is converted from the data type specified in the control file to the data type of the column in the database
- If any of your column names are an Oracle reserved word, then you must enclose them in quotation marks. The only SQL*Loader additional keyword is CONSTANT.

Command Line Listing

```
Usage: SQLLDR keyword=value [,keyword=value,...]

Valid Keywords:
      userid -- ORACLE username/password
      control -- Control file name
      log -- Log file name
      bad -- Bad file name
      data -- Data file name
      discard -- Discard file name
      discardmax -- Number of discards to allow (Default all)
      skip -- Number of logical records to skip (Default 0)
      load -- Number of logical records to load (Default all)
      errors -- Number of errors to allow (Default 50)
      rows -- Number of rows in conventional path bind array or between
      direct path data saves (Default: Conventional path 64, Direct path all)
      bindsize -- Size of conventional path bind array in bytes (Default
      256000)
      silent -- Suppress messages during run
      (header,feedback,errors,discards,partitions)
      direct -- use direct path (Default FALSE)
      parfile -- parameter file: name of file that contains parameter
      specifications
      parallel -- do parallel load (Default FALSE)
      file -- File to allocate extents from
      skip_unusable_indexes -- disallow/allow unusable indexes or index
      partitions (Default FALSE)
      skip_index_maintenance -- do not maintain indexes, mark affected
      indexes as unusable (Default FALSE)
      readsize -- Size of Read buffer (Default 1048576)
      external_table -- use external table for load; NOT_USED, GENERATE_ONLY,
      EXECUTE (Default NOT_USED)
      columnarrayrows -- Number of rows for direct path column array (Default
      5000)
      streamsize -- Size of direct path stream buffer in bytes (Default
      256000)
      multithreading -- use multithreading in direct path
      resumable -- enable or disable resumable for current session (Default
      FALSE)
      resumable_name -- text string to help identify resumable statement
      resumable_timeout -- wait time (in seconds) for RESUMABLE (Default
      7200)
      date_cache -- size (in entries) of date conversion cache (Default 1000)

NOTE: Command-line parameters may be specified either by position or by
keywords.
```

Functions

The following functions provide the means for SQL*Loader to generate the data stored in the database row rather than reading it from a data file. You can use these functions in your control file after you specify the database column that you want to populate.

- CONSTANT
- RECNUM
- SYSDATE
- SEQUENCE

CONSTANT

To specify a constant for a column, use the keyword CONSTANT followed by a value:

```
column_name CONSTANT value
```

RECNUM

Sets the column to the number of the logical record from which that row was loaded. Records are counted sequentially from the beginning of the first datafile starting with record one. It increments for records that are discarded, skipped, rejected, or loaded.

```
column_name RECNUM
```

SYSDATE

Sets the column to the current system date.

```
column_name SYSDATE
```

SEQUENCE

The SEQUENCE keyword ensures a unique value for a particular column (can be used for PIDMS). It does not increment for records that are discarded or skipped.

The combination of column name and the SEQUENCE function is a complete column specification.

```
column_name SEQUENCE (n | MAX | COUNT ,[increment])
```

where:

- *n*
  The sequence starts with the integer value n.
- *COUNT*
  The sequence starts with the number of rows already in the table, plus the increment.
- *MAX*
  The sequence starts with the current maximum value for the column, plus the increment.
- *increment*
  The sequence is incremented by this amount for each successive row. The default increment is 1.

Null values

If you want all inserted values for a given column to be null, omit the column's specifications entirely. To set a column's values conditionally to null based on a test of some condition in the logical record, use the NULLIF clause. To set a numeric column to zero instead of NULL, use the DEFAULTIF clause.

NULLIF

Totally blank fields for numeric or DATE fields cause the record to be rejected. To load one of these fields as null, use the NULLIF clause with the BLANKS keyword. If an all-blank CHAR field is surrounded by enclosure delimiters, then the blanks within the enclosures are loaded. Otherwise, the field is loaded as null.

DEFAULTIF

Using DEFAULTIF on numeric data sets the column to zero when the specified field condition is true. Using the DEFAULTIF on character data (CHAR, DATE, or numeric EXTERNAL) data sets the column to null.

Section L: SQL*Loader & External Tables

Lesson: SQL*Loader Examples

Jump to TOC

Basic Person Information

The following Basic Person Information is to be loaded into SWBPERS:

```
SQL> desc swbpers
 Name                      Null?      Type
 ------------------------  --------   -----------
 SWBPERS_PIDM               NOT NULL   NUMBER(8)
 SWBPERS_SSN                           VARCHAR2(9)
 SWBPERS_BIRTH_DATE                    DATE
 SWBPERS_MRTL_CODE                     VARCHAR2(1)
 SWBPERS_SEX                           VARCHAR2(1)
 SWBPERS_CONFID_IND                    VARCHAR2(1)
 SWBPERS_ACTIVITY_DATE     NOT NULL   DATE
 SWBPERS_USER_ID                       VARCHAR2(30)
 SWBPERS_DATA_ORIGIN                   VARCHAR2(30)
```

Fixed format

A fixed format data file will look something like this:

```
          1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345678
------------------------------------------------------------------

     64415628629112-OCT-1965SF
     65279653263 10-AUG-1972MM
     66169256894 05-JAN-1975SMY
     ...
(SWBPERS.DAT)
```

Control file

Your associated control file will look something like this:

```
(SWBPERS.CTL)
  LOAD DATA
  INFILE 'SWBPERS1.DAT'
  BADFILE 'SWBPERS1.BAD'
  DISCARDFILE 'SWBPERS1.DSC'
  APPEND
  INTO TABLE SWBPERS
  (swbpers_pidm             POSITION(01:08) INTEGER EXTERNAL,
   swbpers_ssn              POSITION(09:17) CHAR,
   swbpers_birth_date       POSITION(18:28) DATE "DD-MON-YYYY"
                           NULLIF birth_date = BLANKS,
   swbpers_mrtl_code        POSITION(29:29) CHAR,
   swbpers_sex              POSITION(30:30) CHAR,
   swbpers_confid_ind       POSITION(31:31) CHAR(1),
   swbpers_activity_date                    SYSDATE,
   swbpers_user_id                          CONSTANT "LOAD",
   swbpers_data_origin                      CONSTANT "SQLLOAD")
```

Analysis

- We have chosen the option of appending to the table. This is the safest option because the existing records within the table will not be affected by our actions.
- The length of the fields is explicitly specified and remains standard throughout the entire data file. Following the POSITION specification is the field's data type from the data file.
- The birth_date from the data file is in the format of a four-digit year rather than two-digit. Therefore, a mask is specified so that SQL*Loader can convert this.
- Because the birth_date is not required, using the NULLIF function will insert a null into the column. The DEFAULTIF function would have worked as well.
- Our character fields will automatically default to NULL if it is equal to spaces in the data file; so we have not used the NULLIF or DEFAULTIF function. We would only use the NULLIF or DEFAULTIF function for character fields when our data was enclosed with delimiters.

Text delimited

Your data file will look something like this:

```
64,156286291,12-OCT-1965,S,F,
65,279653263,10-AUG-1972,M,M,
66,169256894,05-JAN-1975,S,M,Y
...
(SWBPERS2.DAT)
```

Control file

Your associated control file will look something like this:

```
(SWBPERS2.CTL)

  LOAD DATA
  INFILE 'SWBPERS2.DAT'
  BADFILE 'SWBPERS2.BAD'
  DISCARDFILE 'SWBPERS2.DSC'
  INTO TABLE SWBPERS
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  (swbpers_pidm             integer external,
   swbpers_activity_date    SYSDATE,
   swbpers_user_id          CONSTANT  "LOAD",
   swbpers_data_origin      CONSTANT "SQLLOAD",
   swbpers_ssn              CHAR(9),
   swbpers_birth_date       DATE(11) "DD-MON-YYYY"
                       NULLIF birth_date = BLANKS,
   swbpers_mrtl_code        CHAR(1),
   swbpers_sex              CHAR(1),
   swbpers_confid_ind       CHAR(1))
```

Analysis

- Neither APPEND, INSERT, TRUNCATE, or REPLACE has been chosen as how the records should be inserted. Therefore, the default of INSERT will be used. This means that the table must be empty before SQL*Loader is run or it will abort.
- Because the data file is terminating the fields by a comma, each field's length from each record will be evaluated separately. Positions are relative and not fixed.

Combined files

You can combine the control file and data file into one. The asterisk indicates to SQL*Loader that the data is contained within the file, and the BEGINDATA keyword indicates that the data will begin on the following line.

```
LOAD DATA
INFILE *
BADFILE 'SWBPERS.BAD'
DISCARDFILE 'SWBPERS.DSC'
APPEND
INTO TABLE SWBPERS
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(swbpers_pidm             integer external,
 swbpers_activity_date    SYSDATE,
 swbpers_user_id          CONSTANT  "LOAD",
 swbpers_data_origin      CONSTANT "SQLLOAD",
 swbpers_ssn              CHAR(9),
 swbpers_birth_date       DATE(11) "DD-MON-YYYY"
              NULLIF swbpers_birth_date = BLANKS,
 swbpers_mrtl_code        CHAR(1),
 swbpers_sex              CHAR(1),
 swbpers_confid_ind       CHAR(1))

BEGINDATA
64,156286291,12-OCT-1965,S,F,
65,279653263,10-AUG-1972,M,M,
66,169256894,05-JAN-1975,S,M,Y
...

(SWBPERS.CTL)
```

Running SQL*Loader

Once you have your files ready, it is time to run. At the prompt (for UNIX), type the following:

```
sqlldr username control={control file}
```

You will be prompted for your password.
For our previous example, we would invoke SQL*Loader by typing the following:

```
$sqlldr username control=SWBPERS1.CTL
```

Output files

The following file types will be created if specified in either the control file or the command line. These files are created by SQL*Loader to indicate the success or failure of the load.

- Log Files (.LOG)
- Bad Files (.BAD)
- Discard Files (.DSC)

The log file

The log file will provide the statistics of the load. The first section shows the parameters and data layout that it pulled from the control file. The second section lists the rejected record numbers and rejection reasons. The end of the file will list the total number of records read, records that passed, and records that failed.

Log file example

Here is an example of a log file from the fixed format SWBPERS example, as shown previously:

```
SQL*Loader: Release 9.2.0.4.0 - Production on Wed Feb 26 15:04:45
2005

Copyright (c) Oracle Corporation 1982, 2004.  All rights reserved.

Control File:   ././SWBPERS1.CTL
Data File:      ././SWBPERS1.DAT
Bad File:     ././SWBPERS1.BAD
Discard File: SWBPERS1.DSC
(Allow all discards)
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 65536 bytes
Continuation:   none specified
Path used:      Conventional

Table SWBPERS, loaded from every logical record.
Insert option in effect for this table: APPEND
```

Log file example (cont.)

```
Column Name                    Position  Len  Term   Encl  Data type
------------------------------ --------  ---  ----   ----  ---------
SWBPERS_PIDM                       1:8    9     ,   O(")  CHARACTER
SWBPERS_SSN                       9:17    9     ,   O(")  CHARACTER
SWBPERS_BIRTH_DATE               18:28   11     ,   O(")  DATE  DD-MON-YYYY
SWBPERS_MRTL_CODE                29:29    1     ,   O(")  CHARACTER
SWBPERS_SEX                      30:30    1     ,   O(")  CHARACTER
SWBPERS_CONFID_IND               31:31    1     ,   O(")  CHARACTER
SWBPERS_ACTIVITY_DATE          SYSDATE
SWBPERS_USER_ID                                           CONSTANT
          Value is 'LOAD'
SWBPERS_DATA_ORIGIN                                       CONSTANT
          Value is 'SQLLOAD'

Column BIRTH_DATE is NULL if BIRTH_DATE = BLANKS
Record 4: Rejected - Error on table SWBPERS.
ORA-01400: mandatory (NOT NULL) column is missing or NULL during
insert
Record 5: Rejected - Error on table SWBPERS, column BIRTH_DATE.
ORA-01841: (full) year must be between -4713 and +9999
Table SWBPERS:
  3 Rows successfully loaded.
  2 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were
  failed.
  0 Rows not loaded because all fields were null.


Space allocated for bind array:  3328 bytes(64 rows)
Space allocated for memory
besides bind array:           58658 bytes

Total logical records skipped:          0
Total logical records read:             5
Total logical records rejected:         2
Total logical records discarded:        0

Run began on Wed Feb 26 15:04:45 2005
Run ended on Wed Feb 26 15:04:53 2005
Elapsed time was:     00:00:07.86
CPU time was:         00:00:00.30
```

Analyzing the log file

- Although we did not specify the errors allowed or the bind array in the control file, these values defaulted. You can always check the log to figure out the default values SQL*Loader is using
- Because the log file is from a fixed format data file, the positions are listed along with the length. For variable length format, position would have FIRST and then NEXT, and the Length would contain an asterisk (*) because the length will be determined on a record by record basis
- Although five logical records were read by SQL*Loader, only three were successfully inserted into the SWBPERS table. In order to fix these errors, edit the.BAD file and then run the control file against it
- Record 4 was rejected because a mandatory column was missing
- Record 5 was rejected because the date format was incorrect

The bad file

The bad file contains records rejected because of incorrect data. Data can be considered “bad” for a number of reasons. Data is rejected by SQL*Loader when the input format is invalid, such as when the second enclosure delimiter is missing or when a delimited field exceeds its maximum length.

Once SQL*Loader accepts a record for processing, the row is sent to Oracle for insertion. At this point, the row can also be considered “bad”. Examples might be because a key is not unique, because a required field is null, or because the field contains invalid data for the Oracle data type.

The discard file

The discard file contains records that were filtered out of the load because they did not match any of the record-selection criteria (WHEN clause) specified in the control file. This file is created only when it is needed, and only if you have specified that a discard file should be enabled.

The discard file is written in the same format as the data file. The discard data can be loaded with the existing control file after any necessary editing or correcting.

Section L: SQL*Loader & External Tables

Lesson: External Tables

Jump to TOC

Description

External tables are tables that look and act like Oracle tables, but the data is stored outside the database in a flat file.

External tables have some differences from standard Oracle tables:

- Read Only
- Cannot have Indexes or Constraints
- No DML allowed

External tables can be used for data that comes from a non-Oracle source and changes frequently. You can replace the external file with new data, as long as it has the same structure and file name, without making any changes to the database.

Because external tables cannot be indexed, they are not recommended for large amounts of data unless you plan on reading the entire file of data each time you reference that external table.

Sample Syntax:

```
CREATE TABLE emp_load
      (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
ORGANIZATION EXTERNAL
      (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
ACCESS PARAMETERS (RECORDS DELIMITED BY '|' FIELDS TERMINATED BY ','
       (first_name CHAR(7),
        last_name CHAR(8),
        year_of_birth CHAR(4)))
LOCATION ('EMPLOYEE.dat'));
```

Additional Information

Additional information on SQL*Loader functionality and External Tables can be found in the Oracle Utilities Reference Manual.

Section L: SQL*Loader & External Tables

Lesson: Self Check

Jump to TOC

Directions

Use the information you have learned in this workbook to complete this self check activity.

Exercise 1

Examine the SWRIDEN.DAT file that has been provided. What type of data file is it?

Exercise 2

Create a control file that will load the data into the table. For the PIDM, use the maximum PIDM number in the swriden table and increment by one. Use the current system date for the activity date.

Exercise 3

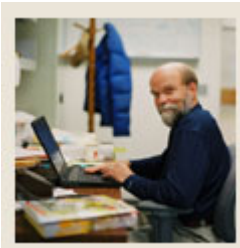
Run the SQL*Loader command line utility, either on your database server, or on your PC using the instruction in the supplied SQL-Loader exercise.doc.

Exercise 4

Examine your log file. What was the success rate? Which records, if any, did not load correctly?

Exercise 5

What steps would you take to fix the records that had errors and reload the data?



Section M: SQL*Plus Reporting

Lesson: Overview

Jump to TOC

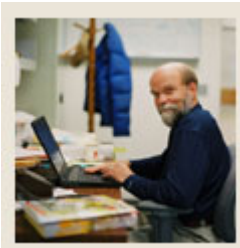
Introduction

So far, you have learned how to query the database to return information. However, the data has not been formatted for reporting purposes. SQL*Plus gives you the capability to format the appearance of the data returned from a SQL Statement.

Objectives

At the end of this section, participants will be able to

- change column headings
- format NUMBER, CHAR, VARCHAR2 (VARCHAR), LONG, DATE, and columns
- copy, list, and reset column display attributes
- suppress duplicate values and insert spaces for clarity
- calculate and print summary lines (totals, averages, minimums, maximums, and more)
- list and remove spacing and summary line definitions
- set page dimensions
- place titles at the top and bottom of each page
- display column values and the current date or page number in titles
- list and suppress page title definitions
- send query results to a file or printer.



Section M: SQL*Plus Reporting

Lesson: Overview (Continued)

Jump to TOC

Section contents

Section M: SQL*Plus Reporting

Lesson: Example Query

Jump to TOC

Example query

Our examples will be based upon the following query. Enter it into SQL*Plus, and execute it to see your results.

```
SQL> SELECT    twraccd_term_code, swriden_last_name||', '||
               swriden_first_name name, twraccd_bill_date,
               twraccd_detc_code, twraccd_amount
         FROM  swriden, twraccd
        WHERE  swriden_pidm = twraccd_pidm
          AND  swriden_change_ind is null
     ORDER BY  twraccd_term_code,swriden_last_name,
               Swriden_first_name,twraccd_bill_date,
               Twraccd_detc_code;
```

Section M: SQL*Plus Reporting

Lesson: Suppressing Duplicate Values in Break Columns

Jump to TOC

Suppress duplicate values

The BREAK command suppresses duplicate values by default in the column or expression you name. Thus, to suppress the duplicate values in a column specified in an ORDER BY clause, use the BREAK command in its simplest form:

```
BREAK ON column_name
```

Note: Whenever you specify a column or expression in a BREAK command, use an ORDER BY clause specifying the same column or expression. If you do not do this, the breaks may appear to occur randomly.

Example

To suppress the display of duplicate term codes in the previous query results, enter the following commands:

```
SQL> BREAK ON TWRACCD_TERM_CODE
```

SQL*Plus displays the following output:

```
                                 Billing
Term   Name                      Date      Category   Amount
------ ------------------------- --------- -------- ----------
200402 Erickson, Ralph           21-MAY-05 LABS        $120.00
       Erickson, Ralph           21-MAY-05 DORM      $1,000.00
200501 Brown, Julie              21-MAY-05 BOOK        $300.20
       Brown, Julie              21-MAY-05 TUIT      $1,500.50
       Brown, Julie              28-MAY-05 BKSC        $700.00
       Erickson, Ralph           21-MAY-05 MEAL        $900.00
       Erickson, Ralph           21-MAY-05 CRED        $400.00
       Erickson, Ralph           25-MAY-05 CASH        $800.00
       Erickson, Susan           21-MAY-05 TUIT        $300.00
       Jones-Erickson, Sandy     18-MAY-05 FAID      $1,100.00
       Jones-Erickson, Sandy     21-MAY-05 TUIT        $800.00
       Smith, Robert             21-MAY-05 DORM        $500.00
       Smith, Robert             21-MAY-05 TUIT      $1,100.00
       White, Nancy              21-MAY-05 CHEK        $950.00
       White, Nancy              21-MAY-05 LABS         $50.00
       White, Nancy              21-MAY-05 TUIT        $900.00
200502 Erickson, Ralph           21-MAY-05 BOOK        $400.00
       Erickson, Ralph           21-MAY-05 TUIT        $750.00
...
```

Insert blank lines

You can insert blank lines or begin a new page each time the value changes in the break column.

To insert n blank lines, use the BREAK command in the following form:

```
BREAK ON break_column SKIP n
```

Example

To place one blank line between term codes, enter the following command:

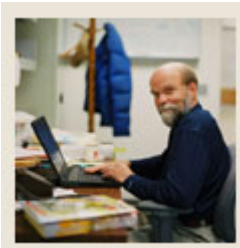
```
SQL> BREAK ON TWRACCD_TERM_CODE SKIP 1
```

Now rerun the query:

```
SQL> /
```

SQL*Plus displays the results:

```
                                 Billing
Term   Name                      Date      Category     Amount
------ ------------------------- --------- -------- ----------
200402 Erickson, Ralph           21-MAY-05 LABS        $120.00
       Erickson, Ralph           21-MAY-05 DORM      $1,000.00

200501 Brown, Julie              21-MAY-05 BOOK        $300.20
       Brown, Julie              21-MAY-05 TUIT      $1,500.50
       Brown, Julie              28-MAY-05 BKSC        $700.00
       Erickson, Ralph           21-MAY-05 MEAL        $900.00
       Erickson, Ralph           21-MAY-05 CRED        $400.00
       Erickson, Ralph           25-MAY-05 CASH        $800.00
       Erickson, Susan           21-MAY-05 TUIT        $300.00
       Jones-Erickson, Sandy     18-MAY-05 FAID      $1,100.00
       Jones-Erickson, Sandy     21-MAY-05 TUIT        $800.00
       Smith, Robert             21-MAY-05 DORM        $500.00
       Smith, Robert             21-MAY-05 TUIT      $1,100.00
       White, Nancy              21-MAY-05 CHEK        $950.00
       White, Nancy              21-MAY-05 LABS         $50.00
       White, Nancy              21-MAY-05 TUIT        $900.00

200502 Erickson, Ralph           21-MAY-05 BOOK        $400.00
...
```



Specify multiple columns

Suppose you have more than one column in your ORDER BY clause and wish to insert space when each column's value changes. Each BREAK command you enter replaces the previous one. Thus, if you want to use different spacing techniques in one report or insert space after the value changes in more than one ordered column, you must specify multiple columns and actions in a single BREAK command. A long BREAK statement may be continued on multiple lines by adding the continuation character "-" (dash) to the end of each line to be continued.

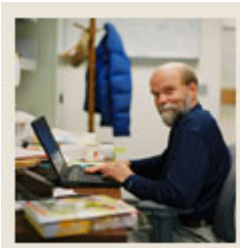
Example

To skip a page when the value of TERM_CODE changes, one line when the value of NAME changes, and two lines at the end of the report, enter the following command:

```
SQL>     BREAK ON TWRACCD_TERM_CODE SKIP PAGE ON NAME SKIP 1
ON REPORT SKIP 2
```

Run the query to see the results:

```
                                 Billing
Term   Name                      Date      Category    Amount
------ ------------------------- --------- -------- ---------
200402 Erickson, Ralph           21-MAY-05 LABS       $120.00
                                 21-MAY-05 DORM     $1,000.00

                                 Billing
Term   Name                      Date      Category    Amount
------ ------------------------- --------- -------- ---------

200501 Brown, Julie              21-MAY-05 BOOK       $300.20
                                 21-MAY-05 TUIT     $1,500.50
                                 28-MAY-05 BKSC       $700.00

       Erickson, Ralph           21-MAY-05 MEAL       $900.00
                                 21-MAY-05 CRED       $400.00
                                 25-MAY-05 CASH       $800.00

       Erickson, Susan           21-MAY-05 TUIT       $300.00

      Jones-Erickson, Sandy      18-MAY-05 FAID     $1,100.00
                                 21-MAY-05 TUIT       $800.00

      Smith, Robert              21-MAY-05 DORM       $500.00
                                 21-MAY-05 TUIT     $1,100.00

      White, Nancy               21-MAY-05 CHEK       $950.00
                                 21-MAY-05 LABS        $50.00
                                 21-MAY-05 TUIT       $900.00


                                 Billing
Term   Name                      Date      Category    Amount
------ ------------------------- --------- -------- ---------

200502 Erickson, Ralph           21-MAY-05 BOOK       $400.00
                                 21-MAY-05 TUIT       $750.00
...
```



List current break definition

You can list your current break definition by entering the BREAK command with no clauses:

```
BREAK
```

Remove current break definition

You can remove the current break definition by entering the CLEAR command with the BREAKS clause:

```
CLEAR  BREAKS
```

Note: You may wish to place the command CLEAR BREAKS at the beginning of every command file to ensure that previously entered BREAK commands will not affect queries you run in a given file.

Section M: SQL*Plus Reporting

Lesson: Computing Summary Lines When a Break Column's Value Changes

Jump to TOC

COMPUTE command

If you organize the rows of a report into subsets with the BREAK command, you can perform various computations on the rows in each subset. You do this with the functions of the SQL*Plus COMPUTE command. Like BREAK, COMPUTE replaces any previously issued COMPUTE statements. A long COMPUTE statement may be continued on multiple lines by adding the continuation character "-" (dash) to the end of each line to be continued.

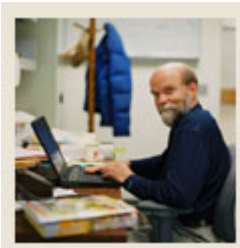
Use the BREAK and COMPUTE commands together in the following forms:

```
BREAK ON break_column
 COMPUTE function LABEL label OF column column column ...
        ON break_column
```

Functions and effects

The following table lists compute functions and their effects:

Function	Effect
SUM	Computes the sum of the values in the column
MIN	Computes the minimum value in the column
MAX	Computes the maximum value in the column
AVG	Computes the average of the values in the column
STD	Computes the standard deviation of the values in the column
VAR	Computes the variance of the values in the column
COUNT	Computes the number of non-null values in the column
NUM	Computes the number of rows in the column



Section M: SQL*Plus Reporting

Lesson: Computing Summary Lines When a Break Column's Value Changes (Continued)

Jump to TOC

Example

To compute the total of AMOUNT by name, first list the current BREAK definition:

```
SQL> BREAK

      break on report skip 2 nodup
            on TERM_CODE page nodup
            on NAME skip 1 nodup
```

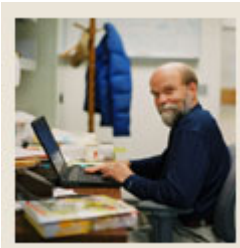
Note**:** When values in successive rows are duplicates, using `nodup` prevents the duplicates from being printed.

Now enter the following COMPUTE command and run the current query:

```
SQL> COMPUTE SUM OF AMOUNT ON NAME
SQL> /
```

SQL*Plus displays the following output:

```
                                 Billing
Term   Name                      Date      Category    Amount
------ ------------------------- --------- -------- ---------
200402 Erickson, Ralph           21-MAY-05 LABS       $120.00
                                 21-MAY-05 DORM     $1,000.00
       ************************                     -----------
       sum                                          $1,120.00


                                 Billing
Term   Name                      Date      Category    Amount
------ ------------------------- --------- -------- ---------
200501 Brown, Julie              21-MAY-05 BOOK       $300.20
                                 21-MAY-05 TUIT     $1,500.50
                                 28-MAY-05 BKSC       $700.00
       ************************                     -----------
       sum                                          $2,500.70
 ...
```



Section M: SQL*Plus Reporting

Lesson: Computing Summary Lines at the End of the Report

Jump to TOC

Compute summary lines

You will want to add a total billed amount for each term, and a grand total at the end of the report. First, find out what your existing COMPUTEs are by typing the following:

```
SQL> COMPUTE
```

Because we are summing for the same column, only at different breakpoints, we will revise the existing compute. Do this by typing the following:

```
SQL> COMPUTE SUM OF twraccd_amount ON name twraccd_term_code –
     REPORT
```

You should see the total appearing after each name and term and a grand total at the end of the report.

Clearing Computes

To remove all compute statements issue the command:

```
SQL> CLEAR COMPUTES
```

TTITLE and BTITLE

You can set a title to display at the top of each page of a report as well as a title to display at the bottom of each page. The TTITLE command defines the top title; the BTITLE command defines the bottom title.

```
TTITLE position_clause(s) char_value -
 position_clause(s) char_value ...
```

or

```
BTITLE position_clause(s) char_value -
 position_clause(s) char_value ...
```

A long TTITLE or BTITLE statement may be continued on multiple lines by adding the continuation character "-" (dash) to the end of each line to be continued, as in the above example.

Clauses

The most often used clauses of TTITLE and BTITLE are summarized in the following table:

Clause	Example	Description
COL n	COL 72	Makes the next CHAR value appear in the specified column of the line.
SKIP n	SKIP 2	Skips to a new line n times. If n is greater than 1, n-1 blank lines appear before the next CHAR value.
LEFT	LEFT	Left-aligns the following CHAR value.
CENTER	CENTER	Centers the following CHAR value.
RIGHT	RIGHT	Right-aligns the following CHAR value.

Turning Titles On/Off

There is no "clear TTITLE" command, so to remove the titles from showing up in your output, turn the TTITLE and BTITLE off. Titles can be turned on and off using:

TTITLE OFF / ON
BTITLE OFF / ON

Example

To put titles at the top and bottom of each page of a report, enter:

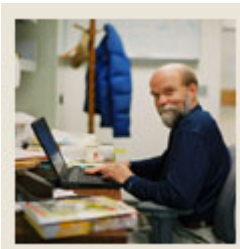
```
SQL> TTITLE CENTER 'ABC University' SKIP 1 CENTER -
 'Billing Report' SKIP 2
SQL> BTITLE CENTER 'CONFIDENTIAL'
```

Now run the current query:

```
SQL> /
```

SQL*Plus displays the following output:

```
                                        ABC University
                                        Billing Report

                                Billing
Term   Name                     Date      Category    Amount
------ ------------------------ --------- -------- ---------
200402 Erickson, Ralph          21-MAY-05 LABS       $120.00
                                 1-MAY-05 DORM     $1,000.00
       ************************                    -----------
       sum                                         $1,120.00

******                                             -----------
   sum                                             $1,120.00
   ...
                                          CONFIDENTIAL
   ...
```



Section M: SQL*Plus Reporting

Lesson: Displaying Column Values in Titles

Jump to TOC

Changing master column value

You may wish to create a master/detail report that displays a changing master column value at the top of each page with the detail query results for that value below. You can reference a column value in a top title by storing the desired value in a variable and referencing the variable in a TTITLE command. Use the following form of the COLUMN command to define the variable:

```
COLUMN column_name NEW_VALUE variable_name
```

Example

```
SQL> COLUMN TWRACCD_TERM_CODE NEW_VALUE TRMVAL NOPRINT
```

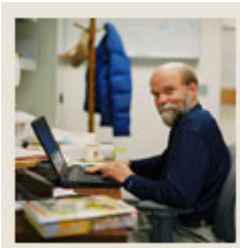
Because you will display the term code in the title, you do not want them to print as part of the detail. The NOPRINT clause you entered above tells SQL*Plus not to print the column TWRACCD_TERM_CODE.

Next, include a label and the value in your page title, enter the proper BREAK command, and suppress the bottom title from the last example:

```
SQL> TTITLE CENTER 'ABC University' -
                        RIGHT 'Page: ' -
          FORMAT 999 SQL.PNO -
          SKIP 1 CENTER 'Billing Report' -
          SKIP 2 - CENTER 'Term Code: ' TRMVAL -
          SKIP 3
```

SQL*Plus displays the following output:

```
                ABC University        Page:    1

                Billing Report

              Term Code: 200402


                           Billing
Name                       Date      Category   Amount
-------------------------- --------- -------- --------
Erickson, Ralph            21-MAY-05 LABS      $120.00
...
```



Section M: SQL*Plus Reporting

Lesson: Storing and Printing Query Results

Jump to TOC

Storing and printing

In most cases, you will want to store the results of the query into a file, or directly send the output to the printer.

Spool to file

To spool to a file:

```
SPOOL  <filename>
<report body>
....
SPOOL OFF
```

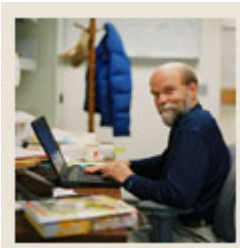
Spool to printer

To spool to the printer:

```
SPOOL OUT
<report body>
...
SPOOL OFF
```

Append to a Spool file (Oracle 10g ONLY)

```
SPOOL  <filename> APPEND
<report body>
....
SPOOL OFF
```



Section M: SQL*Plus Reporting

Lesson: Saving the Commands to a File

Jump to TOC

Storing commands

You should now have an almost finished report. However, it is unrealistic that a user would want to type the commands in, line by line, until the desired results are achieved. Instead, you can store all the commands into one file, and then run the file.

Example file

An example file follows that you would have for the report that you created.

```
SET FEEDBACK OFF
SET ECHO OFF
CLEAR COLUMNS
CLEAR COMPUTES
CLEAR BREAKS
COLUMN NAME HEADING 'Name'
COLUMN TWRACCD_DETC_CODE HEADING 'Category'
COLUMN TWRACCD_BILL_DATE HEADING 'Billing|Date'
COLUMN TWRACCD_AMOUNT HEADING 'Amount' FORMAT $99,999.00
SET LINESIZE 70
SET PAGESIZE 60
COLUMN TWRACCD_TERM_CODE NEW_VALUE TRMVAL NOPRINT
TTITLE CENTER 'ABC University' RIGHT 'Page: ' -
FORMAT 999 SQL.PNO SKIP 1 CENTER 'Billing Report' SKIP 2 -
CENTER 'Term Code: ' TRMVAL SKIP 3
BTITLE CENTER 'CONFIDENTIAL'
BREAK ON TWRACCD_TERM_CODE SKIP PAGE ON NAME SKIP 1
COMPUTE SUM LABEL Total OF TWRACCD_AMOUNT ON name twraccd_term_code -
REPORT
SPOOL BILLING.RPT
    SELECT     TWRACCD_TERM_CODE, SWRIDEN_LAST_NAME
               ||', '|| SWRIDEN_FIRST_NAME NAME,
               TWRACCD_BILL_DATE, TWRACCD_DETC_CODE,
               TWRACCD_AMOUNT
        FROM   SWRIDEN, TWRACCD
       WHERE   SWRIDEN_PIDM = TWRACCD_PIDM
         AND   SWRIDEN_CHANGE_IND IS NULL
    ORDER BY   TWRACCD_TERM_CODE, SWRIDEN_LAST_NAME,
               SWRIDEN_FIRST_NAME, TWRACCD_BILL_DATE,
               TWRACCD_DETC_CODE;
SPOOL OFF
SET FEEDBACK ON
SET ECHO ON
```

Section M: SQL*Plus Reporting

Lesson: HTML Reports

Jump to TOC

HTML Reports

Reports can be created that are either HTML pages themselves or produce output that can be incorporated into an existing HTML page.

To generate HTML output, you need to issue the "SET MARKUP HTML ON SPOOL ON" command. However, this immediately starts outputting HTML codes, so you may not want to turn this option on until you have your SQL defined and tested.

You may include HTML tags as part of your SQL for additional formatting.

To create an HTML report, use the following steps:

- Create and test SQL
- SET MARKUP HTML ON SPOOL ON
- SPOOL <report_name.htm>
- Execute query
- SPOOL off
- Open <report_name.htm> in a browser

The output in SQL*Plus will look like this when you execute the query:

```
14:33:04 c700 &gt; spool myrept.htm
<br>
14:33:11 c700 &gt; /
<br>
<p>
<table border='1' width='90%' align='center' summary='Script output'>
<tr>
<th scope="col">
Term
</th>
<th scope="col">
NAME
</th>
<th scope="col">
Billing
<br>
Date
</th>
<th scope="col">
Category
</th>
<th scope="col">
Amount
</th>
</tr>
<tr>
<td>
200402
</td>
<td>
Erickson, Ralph
</td>
<td>
10-SEP-04
</td>
<td>
LABS
</td>
<td align="right">
    $120.00
</td>
</tr>
```

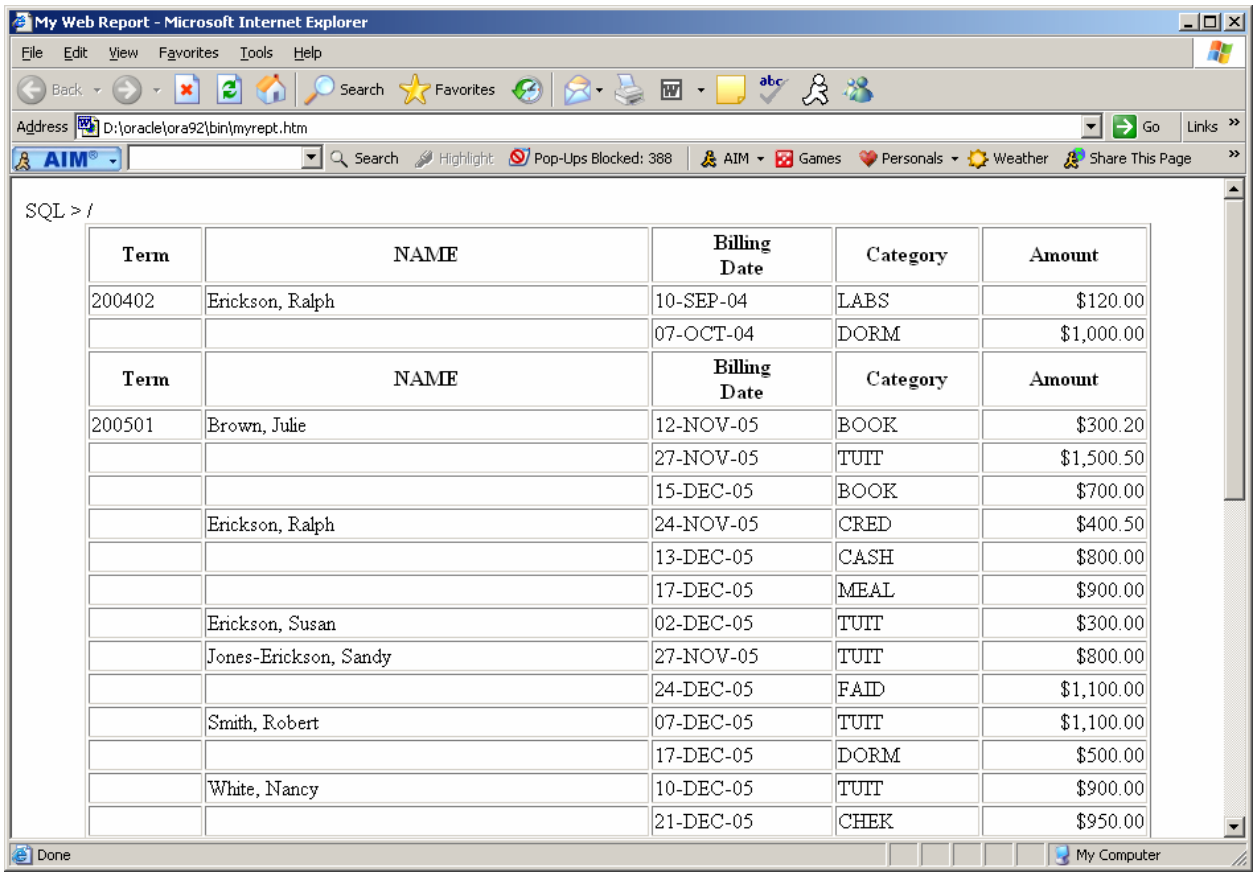
Section M: SQL*Plus Reporting

Lesson: HTML Reports (Continued)

Jump to TOC

HTML Output

Opening the output in a browser results in a report that looks like:



Term	NAME	Billing Date	Category	Amount
200402	Erickson, Ralph	10-SEP-04	LABS	$120.00
		07-OCT-04	DORM	$1,000.00
Term	**NAME**	**Billing Date**	**Category**	**Amount**
200501	Brown, Julie	12-NOV-05	BOOK	$300.20
		27-NOV-05	TUIT	$1,500.50
		15-DEC-05	BOOK	$700.00
	Erickson, Ralph	24-NOV-05	CRED	$400.50
		13-DEC-05	CASH	$800.00
		17-DEC-05	MEAL	$900.00
	Erickson, Susan	02-DEC-05	TUIT	$300.00
	Jones-Erickson, Sandy	27-NOV-05	TUIT	$800.00
		24-DEC-05	FAID	$1,100.00
	Smith, Robert	07-DEC-05	TUIT	$1,100.00
		17-DEC-05	DORM	$500.00
	White, Nancy	10-DEC-05	TUIT	$900.00
		21-DEC-05	CHEK	$950.00

Section M: SQL*Plus Reporting

Lesson: Self Check

Jump to TOC

Description

Use the concepts presented in this section to complete the following exercises.

Exercise 1

Create a SQL*Plus report that shows the Student ID, Name, and the courses for which they are registered (use TERM 200502). Make a separate page for each Student. Create a title with your institution name and a subtitle of 'Registered Courses'. Include the term in the title.

Exercise 2

Using the same report as Exercise 1, create it as an HTML report.

Section N: Answer Key for Self Check Exercises

Lesson: Overview

Jump to TOC

Overview
This section provides the answers to the self check exercises in this course.

This section should be used as an aid when performing the exercises in this course and also as a reference for learning SQL concepts.

Section Contents

Section N: Answer Key for Self Check Exercises

Lesson: Section B – Answer Key

Jump to TOC

Exercise 1
Using the login and database information provided by the instructor, make sure you can log into SQL*Plus on your computer.

Locate the Icon on your desktop or under START→Programs → {Oracle Home or something similar} → Application Development → SQL*Plus

Exercise 2
Issue the following statement to view the tables that you own:
SELECT table_name FROM USER_TABLES;
If no records are retrieved, contact the instructor.

```
SQL> select table_name from user_tables;

    TABLE_NAME
    ------------------------------
    HIGH_MATH
    HIGH_VERBAL
    SWBPERS
    SWRADDR
    SWRIDEN
    SWRREGS
    SWRSTDN
    SWRTEST
    SWVCRSE
    SWVSTDN
    SWVTERM
    TEMP
    TWRACCD
    TWVDETC
```

Exercise 3
Issue the following statement to view the tables that you have permission to view (if the list is long and scrolls off the screen, consider setting PAUSE on with an appropriate pause message):
SELECT table_name FROM ALL_TABLES;

(Your results may vary based on the privileges granted to your account)

```
SQL> select table_name from all_tables;

TABLE_NAME
------------------------------
DUAL
SYSTEM_PRIVILEGE_MAP
TABLE_PRIVILEGE_MAP
STMT_AUDIT_OPTION_MAP
DEF$_TEMP$LOB
WM$WORKSPACES_TABLE
HELP
OGIS_SPATIAL_REFERENCE_SYSTEMS
USER_CS_SRS
USER_TRANSFORM_MAP
SDO_DIST_UNITS
SDO_AREA_UNITS
SDO_ANGLE_UNITS
AUDIT_ACTIONS
SWRADDR
SWBPERS
TWRACCD
TWVDETC
SWVTERM
SWRREGS
SWVCRSE
SWRSTDN
SWVSTDN
SWRTEST
TEMP
HIGH_VERBAL
HIGH_MATH
SWRIDEN
CS_SRS
SDO_ELLIPSOIDS
SDO_PROJECTIONS
SDO_DATUMS
...
```

Section N: Answer Key for Self Check Exercises

Lesson: Section C – Answer Key

Jump to TOC

Exercises use the SWRREGS table.

Exercise 1

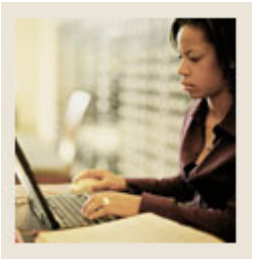
Write a query to return all the columns.

```
SQL>select * from swrregs;

SWRREGS_PIDM SWRREG SWRREGS_CRN SWRREGS_GPA SWRREGS_A
------------ ------ ----------- ----------- ---------
       12340 200501       10001         3.2 20-DEC-05
       12349 200501       10001         3.4 20-DEC-05
       12349 200302       10007         3.1 20-DEC-05
       12346 200402       10001         2.6 20-DEC-05
       12348 200402       10001         2.9 20-DEC-05
       12343 200402       10001           3 20-DEC-05
       12340 200501       10007         2.1 20-DEC-05
       12340 200402       10015         2.8 20-DEC-05
       12340 200502       10017             20-DEC-05
       12340 200502       10004             20-DEC-05
       12340 200402       10008           2 20-DEC-05
       12340 200402       10005           3 20-DEC-05
      ... 44 rows selected
```

Exercise 2

Write a query to return the PIDM (Personal Identification Master), CRN (course number), and GPA (grade point average) for each record.

```
SQL> SELECT swrregs_pidm, swrregs_crn, swrregs_gpa
       FROM swrregs;

SWRREGS_PIDM SWRREGS_CRN SWRREGS_GPA
------------ ----------- -----------
       12340       10001         3.2
       12349       10001         3.4
       12349       10007         3.1
       12346       10001         2.6
       12348       10001         2.9
       12343       10001           3
       12340       10007         2.1
       12340       10015         2.8
... 44 rows selected
```



Exercise 3

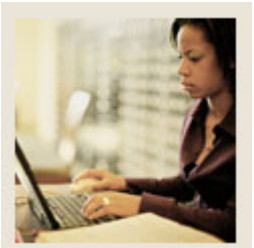
Write a query to return the unique course numbers.

```
SQL> SELECT DISTINCT swrregs_crn
       FROM swrregs;

SWRREGS_CRN
-----------
      10001
      10002
      10003
      10004
      10005
      10006
      10007
      10008
      10009
    ... 22 rows selected
```

Exercise 4

Write a query to return the row number, row identification, and PIDM for each record.

```
SQL> SELECT ROWNUM, ROWID, swrregs_pidm
       FROM SWRREGS;

    ROWNUM ROWID              SWRREGS_PIDM
---------- ------------------ ------------
         1 AAAN0bAAJAAAAA9AAA        12340
         2 AAAN0bAAJAAAAA9AAB        12349
         3 AAAN0bAAJAAAAA9AAC        12349
         4 AAAN0bAAJAAAAA9AAD        12346
         5 AAAN0bAAJAAAAA9AAE        12348
         6 AAAN0bAAJAAAAA9AAF        12343
         7 AAAN0bAAJAAAAA9AAG        12340
         8 AAAN0bAAJAAAAA9AAH        12340
         9 AAAN0bAAJAAAAA9AAI        12340
        10 AAAN0bAAJAAAAA9AAJ        12340
        11 AAAN0bAAJAAAAA9AAK        12340
       ... 44 rows selected
```



Exercise 5

Select the system date from the dummy table DUAL.

```
SQL> SELECT SYSDATE
       FROM DUAL;

SYSDATE
---------
<today's date>
```

Section N: Answer Key for Self Check Exercises

Lesson: Section D – Answer Key

Jump to TOC

Exercise 1

Query the first five rows from the **SWRREGS** table.

```
SQL> SELECT * FROM swrregs
   2  WHERE ROWNUM <6;

SWRREGS_PIDM SWRREG SWRREGS_CRN SWRREGS_GPA SWRREGS_A
------------ ------ ----------- ----------- ---------
       12340 200501       10001         3.2 20-DEC-05
       12349 200501       10001         3.4 20-DEC-05
       12349 200302       10007         3.1 20-DEC-05
       12346 200402       10001         2.6 20-DEC-05
       12348 200402       10001         2.9 20-DEC-05
5 rows selected.
```

Try to query rows 3 and higher from **SWRREGS**. What occurred?

```
SQL>  SELECT *
           FROM swrregs
          WHERE ROWNUM > 3;

   no rows selected
```

This will never return any rows because ROWNUM is assigned on order of display. For example, the first row to be displayed is assigned ROWNUM of 1, but then fails the condition of ROWNUM > 3 and is consequently discarded. The second row fetched will therefore be assigned ROWNUM of 1, and will also fail the condition. Subsequently, all rows will fail to meet the condition.

Exercise 2

Using **SWRADDR**, find the city, state, and zip code for PIDM (internal identification master) 12340. Remember to describe the table first to see the names of the fields.

```
SQL> SELECT    swraddr_city, swraddr_stat_code, swraddr_zip
       FROM    swraddr
      WHERE    swraddr_pidm = 12340;

SWRADDR_CITY         SWR SWRADDR_ZI
-------------------- --- ----------
WEST CHESTER         PA  19380
```

Exercise 3

From **SWRIDEN**, query the students who do not have the last name of 'Erickson'. Remember to only include the most current record for each student. Return all columns.

```
SQL> SELECT *
       FROM swriden
       WHERE swriden_change_ind IS NULL
        AND swriden_last_name <> 'Erickson';

  PIDM SWRIDEN_I SWRIDEN_LAST_NA SWRIDEN_FIRST_N MI      S SWRIDEN_A SWRIDEN_USER SWRIDEN_DA
------ --------- -------------- -------------- ------- - --------- ------------ ----------
 12340 857834585 Brown          Julie          K         20-DEC-05 TRAIN_ORA101 TRAINING
 12341 882993466 Smith          Robert         E         20-DEC-05 TRAIN_ORA101 TRAINING
 12342 853954312 Johnson        Peter          S         20-DEC-05 TRAIN_ORA101 TRAINING
 12343 845672112 Jones-Erickson Sandy          J         20-DEC-05 TRAIN_ORA101 TRAINING
... 18 rows selected
```

Exercise 4

Use the single ampersand in a SQL statement to prompt for a table name, displaying all columns within a table. Run this query for **SWRREGS** and **SWRIDEN.**

```
SQL> SELECT * FROM &SELECT_TABLE;
Enter value for table: swrregs
old   1: SELECT * FROM &TABLE
new   1: SELECT * FROM swrregs

SQL> /
Enter value for table: swriden
old   1: SELECT * FROM &TABLE
new   1: SELECT * FROM swriden
```

Exercise 5

Retrieve the first name, last name, and ID from **SWRIDEN** where the ID begins with a 0. (Hint: use the keyword LIKE)

```
SQL> SELECT swriden_first_name, swriden_last_name, swriden_id
       FROM swriden
      WHERE swriden_id like '0%';

 SWRIDEN_FIRST_N SWRIDEN_LAST_NA SWRIDEN_I
--------------- --------------- ---------
Frederick       Dukes           029348721
Marcus          Jordan          029348721
```

Retrieve the first name, last name, and ID from **SWRIDEN** where the fourth character in the ID field is a 6 and the total length of the column is 7.

```
SQL>  SELECT swriden_first_name, swriden_last_name, swriden_id
        FROM swriden
       WHERE swriden_id LIKE '___6___';

SWRIDEN_FIRST_N SWRIDEN_LAST_NA SWRIDEN_I
--------------- --------------- ---------
Jennifer        Jameson         8486565
```

Section N: Answer Key for Self Check Exercises

Lesson: Section D – Answer Key (Continued)

Jump to TOC

Exercise 6

Using the **SWRREGS** table, list the PIDM, CRN, and GPA for students who have taken golf, tennis, or swimming. (Hint: look at the validation table SWVCRSE first to find the code values for the courses.)

```
SQL> SELECT * FROM swvcrse;

 SQL> SELECT   swrregs_pidm, swrregs_crn, swrregs_gpa
        FROM   swrregs
       WHERE   swrregs_crn IN (10018, 10019, 10020);

 SWRREGS_PIDM SWRREGS_CRN SWRREGS_GPA
 ------------ ----------- -----------
        12341       10018         2.4
        12342       10020
        12344       10019
```

Section N: Answer Key for Self Check Exercises

Lesson: Section D – Answer Key (Continued)

Jump to TOC

Note

For exercises 7 - 9, refer to the SAT test table, **SWRTEST.**

Exercise 7

Retrieve the student records where the student achieved above 550 in both math and verbal.

```
SQL> SELECT *
       FROM swrtest
      WHERE swrtest_sat_verbal > 550
        AND swrtest_sat_math > 550;

SWRTEST_PIDM SWRTEST_T SWRTEST_SAT_VERBAL SWRTEST_SAT_MATH SWRTEST_A
------------ --------- ------------------ ---------------- ---------
       12341 08-JUN-05                590              610 20-NOV-05
       12345 10-DEC-05                590              620 31-OCT-05
       12346 20-NOV-05                630              590 31-OCT-05
```

Exercise 8

Retrieve the student records where the student achieved above 550 in either math or verbal.

```
SQL> SELECT *
       FROM swrtest
      WHERE swrtest_sat_verbal > 550
         OR swrtest_sat_math > 550;

SWRTEST_PIDM SWRTEST_T SWRTEST_SAT_VERBAL SWRTEST_SAT_MATH SWRTEST_A
------------ --------- ------------------ ---------------- ---------
       12341 03-MAR-05                530              580 31-OCT-05
       12342 13-FEB-05                660              520 31-OCT-05
       12341 08-JUN-05                590              610 20-NOV-05
       12345 10-DEC-05                590              620 31-OCT-05
       12346 20-NOV-05                630              590 31-OCT-05
```

Exercise 9

Retrieve the student records where the student took the SAT between '01-JAN-05' and '31-MAY-05' (if the two digit year does not work try entering the four digit year).

```
  SQL> SELECT * FROM swrtest
        WHERE swrtest_test_date BETWEEN '01-JAN-05' AND '31-MAY-05';

SWRTEST_PIDM SWRTEST_T SWRTEST_SAT_VERBAL SWRTEST_SAT_MATH SWRTEST_A
------------ --------- ------------------ ---------------- ---------
       12340 01-MAR-05                550              480 31-OCT-05
       12341 03-MAR-05                530              580 31-OCT-05
       12342 13-FEB-05                660              520 31-OCT-05
       12343 03-FEB-05                530              420 11-OCT-05
```

Section N: Answer Key for Self Check Exercises

Lesson: Section E– Answer Key

Jump to TOC

Exercise 1

In the **SWRREGS** table, what is the average GPA of all the classes that PIDM 12342 took?

```
SQL>  SELECT AVG(NVL(swrregs_gpa,0))
        FROM swrregs
       WHERE swrregs_pidm = 12342;

 AVG(NVL(SWRREGS_GPA,0))
-----------------------
                   2.29
```

Exercise 2

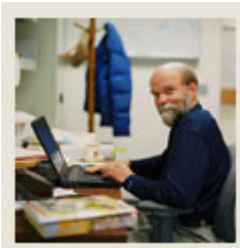
How many records does PIDM 12343 have in the **SWRIDEN** table?

```
SQL>  SELECT COUNT(*)
        FROM swriden
       WHERE swriden_pidm = 12343;

  COUNT(*)
 ---------
         2
```

Exercise 3

Select the PIDM and the combined score of the SAT verbal and math for each record from the **SWRTEST** table.

```
SQL> SELECT swrtest_pidm,
     NVL(swrtest_sat_verbal,0) + NVL(swrtest_sat_math,0)
     FROM swrtest;

SWRTEST_PIDM NVL(SWRTEST_SAT_VERBAL,0)+NVL(SWRTEST_SAT_MATH,0)
------------ ---------------------------------------------
       12340                                           1030
       12341                                           1110
       12342                                           1180
       12341                                           1200
       12343                                            950
       12345                                           1210
       12346                                           1220
       12346                                            980
       12347                                            550
       12345                                            500
       12344                                              0
```



Exercise 4

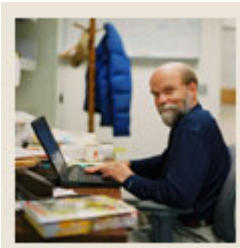
Return the first name concatenated with the last name from the **SWRIDEN** table. Return only rows where the uppercase value of the first name is 'PETER'.

```
SQL> SELECT swriden_first_name||' '|| swriden_last_name NAME
       FROM swriden
      WHERE UPPER (swriden_first_name) = 'PETER';

NAME
----------------------------------------
Peter Johnson
```

Exercise 5

What is the lowest and highest SAT verbal scores for students who took the test in March or April 2005 using **SWRTEST**?

```
SQL>  SELECT MAX(swrtest_sat_verbal), MIN(swrtest_sat_verbal)
        FROM swrtest
       WHERE swrtest_test_date BETWEEN '01-MAR-05' AND
              '30-APR-05';

MAX(SWRTEST_SAT_VERBAL) MIN(SWRTEST_SAT_VERBAL)
----------------------- -----------------------
                    550                     530
```



Exercise 6

Retrieve the PIDM and age (whole number) from **SWBPERS**. Use the birth date and current system date to obtain the age.

```
SQL> SELECT swbpers_pidm,
      TRUNC(MONTHS_BETWEEN(SYSDATE, swbpers_birth_date)/12) "AGE"
     FROM swbpers;

SWBPERS_PIDM        AGE
------------ ----------
       12340         33
       12341         36
       12343         33
       12344         33
       12345         23
       12346         46
       12348         65
       12350         30
       12352         28
       12353         41
       12355         46
       12357         54
       12359         52
```

Section N: Answer Key for Self Check Exercises

Lesson: Section F – Answer Key

Jump to TOC

Exercise 1

Using a combination of functions, parse the street address field into House Number, Street Name, and Street direction.

```
SELECT substr(swraddr_street_line1,1,
              instr(swraddr_street_line1,' ') -1) "House Nbr",
       substr(swraddr_street_line1,instr(swraddr_street_line1,' ') +1,
              instr(swraddr_street_line1,' ',1,2) –
              instr(swraddr_street_line1,' ') -1) "Street",
       substr(swraddr_street_line1,
              instr(swraddr_street_line1,' ',1,2) +1) "Type"
  FROM swraddr;
```

```
House Nbr       Street                         Type
--------------- ------------------------------ --------------------
506             Brown                          St.
210             Pine                           St.
PO              BOX                            1035
23              Market                         St.
18              Chestnut                       Rd.
1821            Canal                          St.
21087           Streetville                    Rd.
2300            Leaffall                       Rd.
3               Ceasar                         Palace Ln.
27              Macchiato                      Blvd.
1               Titans                         Way
43              W.                             Murray St.
21087           Streetville                    Rd.
359             Mustang                        Ave.
```

Exercise 2

Sum the amount column in the TWRACCD table assigning positive values to Trans Type C and negative values to Trans Type P.

```
SELECT sum(decode(twraccd_trans_type,'P',
           twraccd_amount * -1, twraccd_amount)) "Total"
FROM twraccd;
```

```
     Total
----------
    6221.2
```

Section N: Answer Key for Self Check Exercises

Lesson: Section F – Answer Key (continued)

Jump to TOC

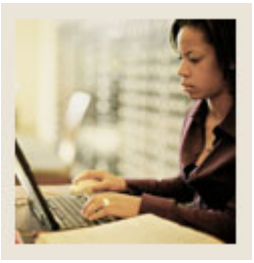
Exercise 3

Adding to the SQL in Exercise 1, check for PO Box addresses and lump 'PO Box' into the Street Name. (Hint: Use the DECODE function).

```
SELECT decode(substr(swraddr_street_line1,1,2),'PO',null,
              substr(swraddr_street_line1,1,
              instr(swraddr_street_line1,' ') -1)) "House Nbr",
       decode(substr(swraddr_street_line1,1,2),'PO',
              substr(swraddr_street_line1,1,
              instr(swraddr_street_line1,' ',1,2) -1),
              substr(swraddr_street_line1,
              instr(swraddr_street_line1,' ') +1,
              instr(swraddr_street_line1,' ',1,2) –
              instr(swraddr_street_line1,' ') -1)) "Street",
       substr(swraddr_street_line1,
              instr(swraddr_street_line1,' ',1,2) +1) "Type"
  FROM swraddr;
```

```
House Nbr       Street                         Type
--------------- ------------------------------ --------------------
506             Brown                          St.
210             Pine                           St.
                PO BOX                         1035
23              Market                         St.
18              Chestnut                       Rd.
1821            Canal                          St.
21087           Streetville                    Rd.
2300            Leaffall                       Rd.
3               Ceasar                         Palace Ln.
27              Macchiato                      Blvd.
1               Titans                         Way
43              W.                             Murray St.
21087           Streetville                    Rd.
359             Mustang                        Ave.
```

Note: As you can see, this is still not a "perfect" parse of addresses as two-word street names get broken apart, and no provision has been made for street direction.



Section N: Answer Key for Self Check Exercises

Lesson: Section G – Answer Key

Jump to TOC

Exercise 1

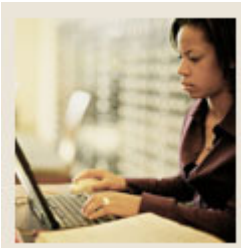
Examine the Student Grades table (**SWRREGS**) using the DESC command. You will be referring to this table for the rest of the section.

```
SQL> DESC swrregs

 Name                           Null?    Type
 ------------------------------ -------- ----
 SWRREGS_PIDM                   NOT NULL NUMBER(8)
 SWRREGS_TERM_CODE              NOT NULL VARCHAR2(6)
 SWRREGS_CRN                    NOT NULL NUMBER(5)
 SWRREGS_GPA                             NUMBER(4,2)
 SWRREGS_ACTIVITY_DATE          NOT NULL DATE
```

Exercise 2

Find the average GPA for each course number. Group by the course number.

```
SQL> SELECT    swrregs_crn, AVG(swrregs_gpa)
       FROM    swrregs
   GROUP BY    swrregs_crn;

SWRREGS_CRN AVG(SWRREGS_GPA)
----------- ----------------
      10001             3.02
      10002              3.1
      10003              2.8
      10004              3.2
      10005             3.05
      10006             2.55
      10007              2.6
      10008       3.03333333
      10009            3.275
      10011       2.56666667
      10012
      10013              2.6
      10014              3.2
      10015             3.15
      10016              1.1
      10017
    ... 22 rows selected.
```



Exercise 3

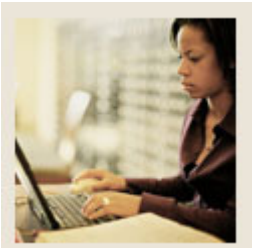
To easily locate the courses with particularly low averages, order your data by the average (lowest first).

```
SQL>   SELECT  swrregs_crn, AVG(swrregs_gpa)
         FROM  swrregs
      GROUP BY  swrregs_crn
      ORDER BY  AVG(swrregs_gpa);

SWRREGS_CRN AVG(SWRREGS_GPA)
----------- ---------------
      10016             1.1
      10023             1.2
      10018             2.4
      10006            2.55
      10011      2.56666667
      10007             2.6
      10013             2.6
      10003             2.8
      10001            3.02
      10008      3.03333333
      10005            3.05
      10002             3.1
    ... 22 rows selected.
```

Exercise 4

Reduce the list so that only courses with averages below 2.0 are returned using a **WHERE** clause. Did you receive an error? Why?

```
SQL> SELECT    swrregs_crn, AVG(swrregs_gpa)
       FROM    swrregs
      WHERE    AVG(swrregs_gpa) < 2.0
   GROUP BY    swrregs_crn order by AVG(swrregs_gpa);


WHERE AVG(swrregs_gpa) < 2.0
      *
ERROR at line 2:
ORA-00934: group function is not allowed here
```



Exercise 5

Try Exercise 4 again, but put the condition in a **HAVING** clause.

```
  SQL> SELECT swrregs_crn, AVG(swrregs_gpa)
         FROM swrregs
        GROUP BY swrregs_crn
       HAVING AVG(swrregs_gpa) < 2.0
        ORDER BY AVG(swrregs_gpa);

SWRREGS_CRN AVG(SWRREGS_GPA)
----------- ----------------
      10016              1.1
      10023              1.2
```

Section N: Answer Key for Self Check Exercises

Lesson: Section G – Answer Key (Continued)

Jump to TOC

Exercise 6

Your institution has changed the testing format for courses 10001 through 10006 from consecutive terms of 200402 and 200501. Examine the effects of the format change. In order to do this, select the course number, term code, and average GPA for the above courses and terms using **SWRREGS**. Group and order by course number and term code.

```
SQL>  SELECT  swrregs_crn, swrregs_term_code, AVG(swrregs_gpa)
        FROM  swrregs
       WHERE  swrregs_term_code in ('200602', '200702')
         AND  swrregs_crn BETWEEN 10001 AND 10006
    GROUP BY  swrregs_crn, swrregs_term_code
    ORDER BY  swrregs_crn, swrregs_term_code;
```

```
SWRREGS_CRN    SWRREGS_TERM_CODE AVG(SWRREGS_GPA)
-------------- ----------------- ----------------------
10001          200702            2.833333333333333333333333333333
10003          200602            3.9
10004          200602            3.6
10004          200702            2.8
10005          200602            3.1
10005          200702            3
10006          200602            2.3
```

According to the data, has the new test format had a positive or negative effect on the GPAs?

The overall average GPAs have risen for each course. Therefore, we are going to assume that the test format has had a positive effect on the student GPAs.

Section N: Answer Key for Self Check Exercises

Lesson: Section G – Answer Key (Continued)

Jump to TOC

Exercise 7

To ensure that there is enough data to make a valid conclusion, make sure at least 3 students have taken the course in a term for the row to be returned. Use a **HAVING** clause to restrict the data being returned.

```
SQL>  SELECT swrregs_crn, swrregs_term_code, AVG(swrregs_gpa)
        FROM swrregs
       WHERE swrregs_term_code between '200501' and '200702'
         AND swrregs_crn BETWEEN 10001 AND 10006
    GROUP BY swrregs_crn, swrregs_term_code
      HAVING COUNT(*) > 2
    ORDER BY swrregs_crn, swrregs_term_code;


SWRREGS_CRN   SWRREGS_TERM_CODE AVG(SWRREGS_GPA)
------------- ----------------- ----------------
10001         200702            2.833333333333333333333333333333333333
```

Section N: Answer Key for Self Check Exercises

Lesson: Section H – Answer Key

Jump to TOC

Exercise 1

Select the ID and combined SAT scores from the **SWRIDEN** and **SWRTEST** tables, using an equi-join. Join by PIDM.

```
  SQL> SELECT swriden_id, swrtest_sat_verbal + swrtest_sat_math
         FROM swriden, swrtest
        WHERE swriden_pidm = swrtest_pidm
          AND swriden_change_ind IS NULL
     ORDER BY swriden_id, swrtest_sat_verbal + swrtest_sat_math;

SWRIDEN_I SWRTEST_SAT_VERBAL+SWRTEST_SAT_MATH
--------- ------------------------------------
822874301
843853339                                   980
843853339                                  1220
845672112                                   950
853954312                                  1180
857834585                                  1030
878549991                                  1210
878549991
882993466                                  1110
882993466                                  1200
892568211
```

Exercise 2

Create a report that contains the same information as above (using the same tables), but also include students who have not taken the SAT test. Use an outer join.

```
SQL> SELECT swriden_id, swrtest_sat_verbal + swrtest_sat_math
       FROM swriden, swrtest
      WHERE swriden_pidm = swrtest_pidm (+)
        AND swriden_change_ind IS NULL
   ORDER BY swriden_id, swrtest_sat_verbal + swrtest_sat_math;

SWRIDEN_I SWRTEST_SAT_VERBAL+SWRTEST_SAT_MATH
--------- -----------------------------------
817253082
822874301
825110988
831603288
832092865
832763321
843853339                                   980
843853339                                  1220
845672112                                   950
8486565
853084511
853954312                                  1180
855231118
857834585                                  1030
861232200
862100933
872109834
878549991                                  1210
878549991
881337923
882993466                                  1110
882993466                                  1200
892568211
... 23 rows selected.
```

Exercise 3

Return the PIDM(s) of the students who are in the **SWRIDEN** table but not in the **SWRREGS** table, using the keyword **MINUS**.

```
  SQL> SELECT swriden_pidm
         FROM swriden
        MINUS
       SELECT swrregs_pidm
         FROM swrregs;

  PIDM
------
 12347
 12350
 12351
 12352
 12353
 12354
 12355
 12356
 12357
 12358
 12359
```

Exercise 4

Find the person who has the highest SAT VERBAL score in the SWRTEST table. Show the PIDM, Name, and test score for that person. (Hint: Use a subquery to determine the highest score.)

```
 SELECT swriden_pidm,
        swriden_last_name || ', ' || swriden_first_name
        || ' ' || swriden_mi "Name",
        swrtest_sat_verbal
   FROM swriden, swrtest
  WHERE swriden_pidm = swrtest_pidm
    AND swrtest_sat_verbal = (SELECT max(swrtest_sat_verbal)
                              FROM swrtest);


SWRIDEN_PIDM Name                      SWRTEST_SAT_VERBAL
------------ ------------------------ ------------------
       12342 Johnson, Peter S                        660
```

Section N: Answer Key for Self Check Exercises

Lesson: Section H – Answer Key (Continued)

Jump to TOC

Exercise 5

Create a query that will select the ID, Full Name, most recent TERM, and classes registered for that term from the **SWRIDEN, SWVCRSE** and **SWRREGS** tables. (Hint: Use a correlated subquery to obtain the most recent TERM_CODE for each person.)

```
SQL> SELECT swriden_id,
            swriden_last_name || ', ' || swriden_first_name
            || ' ' || swriden_mi Full_Name,
            swrregs_term_code, swvcrse_crn, swvcrse_desc
       FROM swriden, swvcrse, swrregs sr1
      WHERE swriden_pidm = sr1.swrregs_pidm
        AND sr1.swrregs_crn = swvcrse_crn
        AND swriden_change_ind is null
        AND sr1.swrregs_term_code =
                 (SELECT max(sr2.swrregs_term_code)
                    FROM swrregs sr2
                   WHERE sr1.swrregs_pidm = sr2.swrregs_pidm)
      ORDER BY 3,1,4
```

```
SWRIDEN_ID FULL_NAME                 SWRREGS_TERM_CODE SWVCRSE_CRN  SWVCRSE_DESC
---------- ------------------------- ----------------- ------------ -----------------
892568211  Erickson, Ralph L         200608            10019        Golf
853954312  Johnson, Peter S          200701            10006        Zoology
853954312  Johnson, Peter S          200701            10009        Calculus
881337923  Serum, Tracy Paige        200701            10007        Philosophy
882993466  Smith, Robert E           200701            10018        Tennis
843853339  White, Nancy Carol        200702            10001        Writing
843853339  White, Nancy Carol        200702            10016        C Programming
845672112  Jones-Erickson, Sandy J   200702            10001        Writing
857834585  Brown, Julie K            200702            10005        Biology
857834585  Brown, Julie K            200702            10008        Psychology
857834585  Brown, Julie K            200702            10015        Speech
862100933  Clifford, Stephanie Geena 200702            10001        Writing
878549991  Erickson, Susan T         200702            10004        Physics
878549991  Erickson, Susan T         200702            10011        Anthropology

14 rows selected
```

Exercise 6

Create a SQL statement to create dynamic SQL that will issue a query against all of your tables, returning the number of records per table that belong to Julie Brown (Hint: Get Julie's PIDM first). Execute the SQL that you create to make sure it works correctly.

```
SQL> SELECT 'SELECT COUNT(*) FROM ' || table_name || ' WHERE ' ||
  2  table_name || '_PIDM = 12340;'
  3  FROM user_tables;

'SELECTCOUNT(*)FROM'||TABLE_NAME||'WHERE'||TABLE_NAME||'_PIDM=12340;'
--------------------------------------------------------------------------------
SELECT COUNT(*) FROM HIGH_MATH WHERE HIGH_MATH_PIDM = 12340;
SELECT COUNT(*) FROM HIGH_VERBAL WHERE HIGH_VERBAL_PIDM = 12340;
SELECT COUNT(*) FROM SWBPERS WHERE SWBPERS_PIDM = 12340;
SELECT COUNT(*) FROM SWRADDR WHERE SWRADDR_PIDM = 12340;
SELECT COUNT(*) FROM SWRIDEN WHERE SWRIDEN_PIDM = 12340;
SELECT COUNT(*) FROM SWRREGS WHERE SWRREGS_PIDM = 12340;
SELECT COUNT(*) FROM SWRSTDN WHERE SWRSTDN_PIDM = 12340;
SELECT COUNT(*) FROM SWRTEST WHERE SWRTEST_PIDM = 12340;
SELECT COUNT(*) FROM SWVCRSE WHERE SWVCRSE_PIDM = 12340;
SELECT COUNT(*) FROM SWVSTDN WHERE SWVSTDN_PIDM = 12340;
SELECT COUNT(*) FROM SWVTERM WHERE SWVTERM_PIDM = 12340;
SELECT COUNT(*) FROM TEMP WHERE TEMP_PIDM = 12340;
SELECT COUNT(*) FROM TERM_GPA WHERE TERM_GPA_PIDM = 12340;
SELECT COUNT(*) FROM TWRACCD WHERE TWRACCD_PIDM = 12340;
SELECT COUNT(*) FROM TWVDETC WHERE TWVDETC_PIDM = 12340;

15 rows selected.

SQL> SELECT COUNT(*) FROM HIGH_MATH WHERE HIGH_MATH_PIDM = 12340;
  COUNT(*)
----------
         0
SQL> SELECT COUNT(*) FROM HIGH_VERBAL WHERE HIGH_VERBAL_PIDM = 12340;
  COUNT(*)
----------
         0
SQL> SELECT COUNT(*) FROM SWBPERS WHERE SWBPERS_PIDM = 12340;
  COUNT(*)
----------
         1

(continued…)
```

Section N: Answer Key for Self Check Exercises

Lesson: Section H – Answer Key (Continued)

Jump to TOC

```
SQL> SELECT COUNT(*) FROM SWRADDR WHERE SWRADDR_PIDM = 12340;
  COUNT(*)
----------
         1
SQL> SELECT COUNT(*) FROM SWRIDEN WHERE SWRIDEN_PIDM = 12340;
  COUNT(*)
----------
         2
SQL> SELECT COUNT(*) FROM SWRREGS WHERE SWRREGS_PIDM = 12340;
  COUNT(*)
----------
         9
SQL> SELECT COUNT(*) FROM SWRSTDN WHERE SWRSTDN_PIDM = 12340;
  COUNT(*)
----------
         1
SQL> SELECT COUNT(*) FROM SWRTEST WHERE SWRTEST_PIDM = 12340;
  COUNT(*)
----------
         1
SQL> SELECT COUNT(*) FROM SWVCRSE WHERE SWVCRSE_PIDM = 12340;
SELECT COUNT(*) FROM SWVCRSE WHERE SWVCRSE_PIDM = 12340
                                  *
ERROR at line 1:
ORA-00904: "SWVCRSE_PIDM": invalid identifier

SQL> SELECT COUNT(*) FROM SWVSTDN WHERE SWVSTDN_PIDM = 12340;
SELECT COUNT(*) FROM SWVSTDN WHERE SWVSTDN_PIDM = 12340
                                  *
ERROR at line 1:
ORA-00904: "SWVSTDN_PIDM": invalid identifier

SQL> SELECT COUNT(*) FROM SWVTERM WHERE SWVTERM_PIDM = 12340;
SELECT COUNT(*) FROM SWVTERM WHERE SWVTERM_PIDM = 12340
                                  *
ERROR at line 1:
ORA-00904: "SWVTERM_PIDM": invalid identifier

SQL> SELECT COUNT(*) FROM TEMP WHERE TEMP_PIDM = 12340;
SELECT COUNT(*) FROM TEMP WHERE TEMP_PIDM = 12340
                                *
ERROR at line 1:
ORA-00904: "TEMP_PIDM": invalid identifier
```

(continued…)

Section N: Answer Key for Self Check Exercises

Lesson: Section H – Answer Key (Continued)

Jump to TOC

```
SQL> SELECT COUNT(*) FROM TERM_GPA WHERE TERM_GPA_PIDM = 12340;
  COUNT(*)
----------
         2
SQL> SELECT COUNT(*) FROM TWRACCD WHERE TWRACCD_PIDM = 12340;
  COUNT(*)
----------
         3
SQL> SELECT COUNT(*) FROM TWVDETC WHERE TWVDETC_PIDM = 12340;
SELECT COUNT(*) FROM TWVDETC WHERE TWVDETC_PIDM = 12340
                                  *
ERROR at line 1:
ORA-00904: "TWVDETC_PIDM": invalid identifier
```

NOTE: Notice that using this method, not all the tables have PIDM columns. Using the USER_TAB_COLUMNS view instead produces more accurate results.

```
SELECT 'SELECT COUNT(*) FROM ' || table_name || ' WHERE ' ||
       column_name || ' = 12340;'
  FROM user_tab_columns
 WHERE column_name like '%PIDM';

'SELECTCOUNT(*)FROM'||TABLE_NAME||'WHERE'||COLUMN_NAME||'=12340;'
--------------------------------------------------------------------------------
SELECT COUNT(*) FROM HIGH_MATH WHERE HIGH_MATH_PIDM = 12340;
SELECT COUNT(*) FROM HIGH_VERBAL WHERE HIGH_VERBAL_PIDM = 12340;
SELECT COUNT(*) FROM SWBPERS WHERE SWBPERS_PIDM = 12340;
SELECT COUNT(*) FROM SWRADDR WHERE SWRADDR_PIDM = 12340;
SELECT COUNT(*) FROM SWRIDEN WHERE SWRIDEN_PIDM = 12340;
SELECT COUNT(*) FROM SWRREGS WHERE SWRREGS_PIDM = 12340;
SELECT COUNT(*) FROM SWRSTDN WHERE SWRSTDN_PIDM = 12340;
SELECT COUNT(*) FROM SWRTEST WHERE SWRTEST_PIDM = 12340;
SELECT COUNT(*) FROM SWVTELE WHERE SWVTELE_PIDM = 12340;
SELECT COUNT(*) FROM TERM_GPA WHERE TERM_GPA_PIDM = 12340;
SELECT COUNT(*) FROM TWRACCD WHERE TWRACCD_PIDM = 12340;

11 rows selected.
```

Note: The query using USER_TAB_COLUMNS included SWVTELE which is a VIEW, not a table. Some of the *_TAB_* views contain information on objects other than tables.

Section N: Answer Key for Self Check Exercises

Lesson: Section I – Answer Key

Jump to TOC

Exercise 1

Insert a new student in the **SWRIDEN** table using your own name, PIDM 2045 and ID 432G. Do not use a middle name.

```
SQL>  INSERT INTO swriden (swriden_pidm, swriden_activity_date,
                           swriden_id, swriden_last_name,
                           swriden_first_name)
         VALUES  (2045, sysdate, '432G',
                 'My_Last_Name', 'My_First_Name');
```

Exercise 2

Add a new student profile record in **SWBPERS** for the new student added in step 1, using the following information:

Activity Date	Current system date
Social Security Number	124-62-8747
Birth Date	Unknown (leave null)
Marital Code	Unknown (leave null)
Sex	Female
Confidential Indicator	Y

Note: Check the description of the table for column size constraints.

```
SQL> INSERT INTO swbpers (swbpers_pidm, swbpers_activity_date,
                  swbpers_ssn, swbpers_sex, swbpers_confid_ind)
         VALUES  (2045,sysdate,124628747,'F','Y');
```

or

```
SQL> INSERT INTO swbpers
        VALUES  (2045,'124628747', NULL, NULL,
                 'F','Y', sysdate, NULL, NULL);
```

Exercise 3

Create a savepoint named SP1.

```
SQL> SAVEPOINT SP1;
Savepoint created.
```

Exercise 4

Insert another row into the **SWRIDEN** table, but prompt the operator for each variable except for the activity date.

```
SQL>  INSERT INTO swriden (swriden_pidm, swriden_activity_date,
                           swriden_id, swriden_last_name,
                           swriden_first_name, swriden_mi,
                           swriden_change_ind)
         VALUES (&Internal_ID, SYSDATE, '&ID',
              '&last_name', '&first_name',
              '&middle_name', '&change_ind');
```

Exercise 5

Update the new student profile record created in Exercise 2 so that the Social Security number is 635-56-1525 and the marital code is 'S' (**SWBPERS)**.

```
SQL> UPDATE SWBPERS
        SET swbpers_ssn = '635561525', swbpers_mrtl_code = 'S'
      WHERE swbpers_pidm = 2045;
```

or

```
SQL> UPDATE SWBPERS
        SET swbpers_ssn = 635561525
      WHERE swbpers_pidm = 2045;

SQL> UPDATE SWBPERS
        SET swbpers_mrtl_code = 'S'
      WHERE swbpers_pidm = 2045;
```

Exercise 6

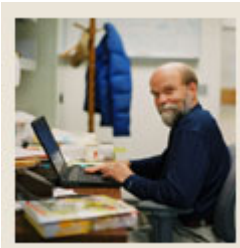
Roll back to savepoint SP1.

```
SQL> ROLLBACK TO SP1;
```

Exercise 7

Commit your changes.

```
COMMIT;
```



Exercise 8

Delete the student profile record created in Exercise 2 (**SWBPERS)**.

```
SQL> DELETE FROM swbpers
           WHERE swbpers_pidm = 2045;

 1 row deleted.
```

Exercise 9

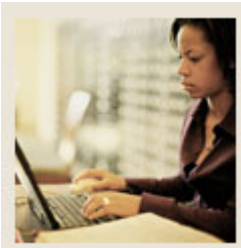
Delete the **SWRIDEN** record you created in Exercise 1.

```
SQL> DELETE FROM swriden
           WHERE swriden_pidm = 2045;
1 row deleted.
```

Exercise 10

Commit your changes.

```
COMMIT;
```



Section N: Answer Key for Self Check Exercises

Lesson: Section J – Answer Key

Jump to TOC

Exercise 1

To make data retrieval faster, create an index for PIDM on **SWRIDEN**.

```
SQL> CREATE INDEX swriden_key_index
          ON swriden (swriden_pidm);

Index created.
```

Exercise 2

Create a relationship between the validation table T**WVDETC** and the repeating table **TWRACCD**. TWVDETC should have the primary key of TWVDETC_DETC_CODE and TWRACCD should have the foreign key of TWRACCD_DETC_CODE.

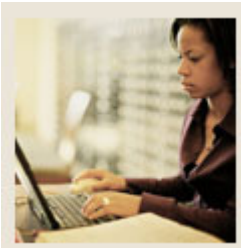
```
SQL> ALTER TABLE twvdetc
       ADD CONSTRAINT pk_twvdetc
         PRIMARY KEY (twvdetc_code);

Table altered.

SQL> ALTER TABLE twraccd
     ADD CONSTRAINT fk1_twraccd_inv_twvdetc_key
         FOREIGN KEY (twraccd_detc_code)
         REFERENCES twvdetc;
Table altered.
```

Exercise 3

Create a table called **TEMP_XX** (where XX is your user number) with the following structure:

```
MYNUMBER NUMBER(8)
TEXT     VARCHAR2(30)
MYDATE   DATE
MESSAGE  VARCHAR2(50)
```

```
SQL> CREATE TABLE temp_xx
    (mynumber  NUMBER(8),
     text      VARCHAR2(30),
     mydate    DATE,
     message   VARCHAR2(50));
```



Section N: Answer Key for Self Check Exercises

Lesson: Section J – Answer Key (Continued)

Jump to TOC

Exercise 4

Add a column called SWRADDR_COUNTRY_CODE, type VARCHAR2(10) NOT NULL to the swraddr table.

What happens when you try to add a NOT NULL column to an existing table?

```
SQL> ALTER TABLE swraddr ADD(swraddr_country_code varchar2(10)
NOT NULL);

ALTER TABLE swraddr ADD(swraddr_country_code varchar2(10) NOT
NULL)
            *
ERROR at line 1:
ORA-01758: table must be empty to add mandatory (NOT NULL) column
```

You cannot add a NOT NULL column to a table with data in it unless you supply a default value, which will populate all existing rows with that default value.

```
SQL> ALTER TABLE swraddr ADD(swraddr_country_code varchar2(10)
DEFAULT 'USA' NOT NULL);

Table altered.
```

Exercise 5

Add a constraint on the SWRREGS table to the SWVCRSE table on CRN. What happens? Correct the problem and try again.

```
SQL> ALTER TABLE swrregs ADD CONSTRAINT fk_swrregs_INV_swvcrse
  2> FOREIGN KEY (swrregs_crn) REFERENCES swvcrse(swvcrse_crn);
(swrregs_crn) REFERENCES swvcrse(swvcrse_crn)
                                 *
ERROR at line 2:
ORA-02270: no matching unique or primary key for this column-list
```

The SWVCRSE table does not have a primary key. Add a primary key on CRN.

```
SQL> ALTER TABLE swvcrse ADD CONSTRAINT pk_swvcrse PRIMARY KEY
(swvcrse_crn);

Table altered.
```

Exercise 5 (continued)

```
SQL> ALTER TABLE swrregs ADD CONSTRAINT fk_swrregs_INV_swvcrse
  2  FOREIGN KEY (swrregs_crn) REFERENCES swvcrse(swvcrse_crn);
alter table swrregs add constraint fk_swrregs_INV_swvcrse FOREIGN
KEY
                                    *
ERROR at line 1:
ORA-02298: cannot validate (TRAIN_ORA101.FK_SWRREGS_INV_SWVCRSE)
- parent keys not found
```

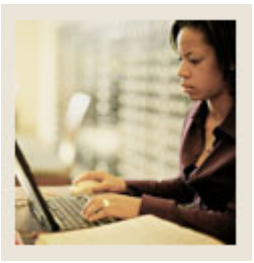
Why will the constraint still not create? How would you fix this problem (Hint: Use a minus query to identify CRN records that do not match)?

The constraint will not enable because there are child CRN values in SWRREGS that do not exist in the parent SWVCRSE table.

```
SQL> SELECT swrregs_crn FROM swrregs
  2  MINUS
  3  SELECT swvcrse_crn FROM swvcrse;

SWRREGS_CRN
-----------
      10033
```

At this point, you have two choices. If 10033 is a valid CRN, add that record to the SWVCRSE table and then create the constraint. If 10033 is not a valid CRN, find out what the correct CRN should be and change the SWRREGS record and then add the constraint. We will assume it was supposed to be CRN 10003.

```
SQL> UPDATE swrregs SET swrregs_crn = 10003 WHERE swrregs_crn =
10033;

1 row updated.


SQL> ALTER TABLE swrregs ADD CONSTRAINT fk_swrregs_INV_swvcrse
   2 FOREIGN KEY (swrregs_crn) REFERENCES swvcrse(swvcrse_crn);

Table altered.
```



Section N: Answer Key for Self Check Exercises

Lesson: Section J – Answer Key (Continued)

Jump to TOC

Exercise 6

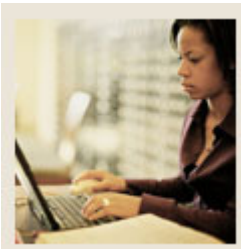
Add a table comment for the SWRIDEN, SWRADDR, and SWBPERS tables.

```
SQL> comment on table SWRIDEN is 'Identification table used in
Intro to Oracle class';
SQL> comment on table SWRADDR is 'Address Repeating Table';
SQL> comment on table SWBPERS is 'Biographical and Demographic
information on persons';
```

Exercise 7

Locate your new indexes and constraints in the user_indexes, user_ind_columns, user_constraints and user_cons_columns data dictionary views.

```
SQL>SELECT index_name, index_type, table_name, uniqueness, status
      FROM user_indexes;
INDEX_NAME           INDEX_TYPE TABLE_NAME      UNIQUENES STATUS
-------------------- ---------- --------------- --------- -------
PK_SWVCRSE           NORMAL     SWVCRSE         UNIQUE    VALID
PK_TWVDETC           NORMAL     TWVDETC         UNIQUE    VALID
SWRIDEN_KEY_INDEX    NORMAL     SWRIDEN         NONUNIQUE VALID

SQL> SELECT index_name, table_name, column_name, column_position,
           descend
      FROM user_ind_columns;

INDEX_NAME           TABLE_NAME    COLUMN_NAME          COL_POSITION DESC
-------------------- ------------ -------------------- ------------ ----
SWRIDEN_KEY_INDEX    SWRIDEN      SWRIDEN_PIDM                    1 ASC
PK_SWVCRSE           SWVCRSE      SWVCRSE_CRN                     1 ASC
PK_TWVDETC           TWVDETC      TWVDETC_CODE                  1 ASC

SQL> SELECT constraint_name, constraint_type, table_name,
           search_condition, status
       FROM user_constraints
    ORDER BY 2;

CONSTRAINT_NAME            C TABLE_NAME  SEARCH_CONDITION                    STATUS
-------------------------- - ----------- ---------------------------------- --------
SYS_C0029178               C HIGH_MATH   "HIGH_MATH_PIDM" IS NOT NULL        ENABLED
SYS_C0029179               C HIGH_MATH   "HIGH_MATH_TEST_DATE" IS NOT NULL   ENABLED
```



Section N: Answer Key for Self Check Exercises

Lesson: Section J – Answer Key (Continued)

Jump to TOC

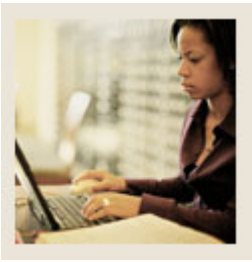
Exercise 7 (Continued)

```
SYS_C0029176                C HIGH_VERBAL "HIGH_VERBAL_PIDM" IS NOT NULL      ENABLED
SYS_C0029177                C HIGH_VERBAL "HIGH_VERBAL_TEST_DATE" IS NOT NULL ENABLED
SYS_C0029143                C SWBPERS     "SWBPERS_PIDM" IS NOT NULL          ENABLED
SYS_C0029144                C SWBPERS     "SWBPERS_ACTIVITY_DATE" IS NOT NULL ENABLED
...
PK_SWVCRSE                  P SWVCRSE                                         ENABLED
PK_TWVDETC                  P TWVDETC                                         ENABLED
FK_SWRREGS_INV_SWVCRSE      R SWRREGS                                         ENABLED
FK1_TWRACCD_INV_TWVDETC_KEY R TWRACCD                                         ENABLED
```

```
SQL> SELECT constraint_name, table_name, column_name, position
       FROM user_cons_columns;

CONSTRAINT_NAME             TABLE_NAME  COLUMN_NAME                    POSITION
--------------------------- ----------- -------------------------- ----------
FK1_TWRACCD_INV_TWVDETC_KEY TWRACCD     TWRACCD_DETC_CODE                   1
FK_SWRREGS_INV_SWVCRSE      SWRREGS     SWRREGS_CRN                         1
PK_SWVCRSE                  SWVCRSE     SWVCRSE_CRN                         1
PK_TWVDETC                  TWVDETC     TWVDETC_CODE                      1
SYS_C0029139                SWRIDEN     SWRIDEN_PIDM
SYS_C0029140                SWRIDEN     SWRIDEN_ID
SYS_C0029141                SWRIDEN     SWRIDEN_LAST_NAME
SYS_C0029142                SWRIDEN     SWRIDEN_ACTIVITY_DATE
SYS_C0029143                SWBPERS     SWBPERS_PIDM
SYS_C0029144                SWBPERS     SWBPERS_ACTIVITY_DATE
SYS_C0029145                SWRADDR     SWRADDR_PIDM
SYS_C0029146                SWRADDR     SWRADDR_ATYP_CODE
SYS_C0029147                SWRADDR     SWRADDR_STREET_LINE1
SYS_C0029148                SWRADDR     SWRADDR_ACTIVITY_DATE
SYS_C0029149                TWRACCD     TWRACCD_PIDM
SYS_C0029150                TWRACCD     TWRACCD_TERM_CODE
SYS_C0029151                TWRACCD     TWRACCD_DETC_CODE
SYS_C0029152                TWRACCD     TWRACCD_TRANS_TYPE
SYS_C0029153                TWRACCD     TWRACCD_BILL_DATE
SYS_C0029154                TWRACCD     TWRACCD_AMOUNT
...
```

The SYS_Cxxxxxxx constraints are system generated names that Oracle assigns when you do not create a name for your constraints. Your numbers may vary as they are sequentially assigned by the database.

As you may have noticed, the not null constraints are created when you define the table and column specifications. You can assign user-defined names to these constraints when you create the table.



Section N: Answer Key for Self Check Exercises

Lesson: Section K – Answer Key

Jump to TOC

Exercise 1

Create a view called **SWVADDR_XX** ***(where XX is your*** user number) which contains a person's first name, last name (combine it into one column called name), city, state, and zip based on the **SWRIDEN** and **SWRADDR** tables.

```
SQL> CREATE VIEW swvaddr_XX  (name, city, state, zip)
        AS SELECT swriden_first_name||' '|| swriden_last_name ,
                  swraddr_city, swraddr_stat_code, swraddr_zip
             FROM swriden, swraddr
            WHERE swriden_pidm = swraddr_pidm
              AND swriden_change_ind IS NULL;
```

Exercise 2

Retrieve all the rows from the new view.

```
SQL> SELECT * FROM swvaddr_XX
```

Exercise 3

Grant the right to select from the view to a person sitting next to you. Make sure someone gives you the right to select from his/her new view.

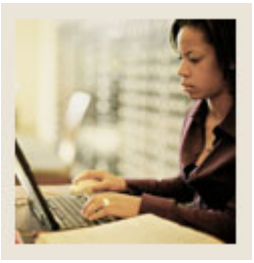
```
SQL> GRANT SELECT ON swvaddr_XX TO trainXX;
```

Exercise 4

Try to select all columns from the view you were just granted access to in Exercise 3. What happened?

```
SELECT * FROM swvaddr_XX;

ERROR at line 1:
ORA-00942: table or view does not exist
```

This is because you did not specify the owner name in front of the view.



Section N: Answer Key for Self Check Exercises

Lesson: Section K – Answer Key (Continued)

Jump to TOC

Exercise 5

Now, put the owner name in front of the table, in the syntax below. Did you get results?

```
SELECT * FROM <owner.table_name>
```

```
SELECT * FROM trainXX.swvaddr_XX;
```

Exercise 6

Because you have to specify the owner each time you are referring to the view, create a synonym to alleviate this.

```
SQL> CREATE SYNONYM swvaddr_XX
        FOR trainXX.swvaddr_XX;
```

Exercise 7

Select all columns from the view. You should not have to specify the owner in front of the view.

SQL> SELECT * FROM swvaddr_XX;

Exercise 8

Create a sequence which will be used to generate a new PIDM. Find out what the first value should be by finding the maximum existing PIDM +1. Insert a new row into the **SWRIDEN** using your sequence to generate the PIDM.

```
SQL> SELECT MAX(swriden_pidm) + 1 FROM swriden;

MAX(SWRIDEN_PIDM)+1
-------------------
              12360

SQL> CREATE SEQUENCE PIDM_SEQUENCE
      START WITH 12360;

Sequence created.

SQL>  INSERT INTO swriden (swriden_pidm, swriden_id,
                           swriden_last_name, swriden_activity_date)
      VALUES   (pidm_sequence.NEXTVAL, '123ME',
                    'Smith', sysdate);
```

Section N: Answer Key for Self Check Exercises

Lesson: Section L – Answer Key

Jump to TOC

Exercise 1

Examine the SWRIDEN.DAT file that has been provided. What type of data file is it?

A comma delimited file.

Exercise 2

Create a control file that will load the data into the table. For the PIDM, use the maximum PIDM number in the swriden table and increment by one. Use the current system date for the activity date.

```
LOAD DATA
INFILE 'swriden.dat'
BADFILE 'swriden.bad'
DISCARDFILE 'swriden.dsc'
APPEND
INTO TABLE swriden
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
        TRAILING NULLCOLS
(swriden_pidm             SEQUENCE(MAX, 1),
 swriden_activity_date    SYSDATE,
 swriden_id               CHAR,
 swriden_last_name        CHAR,
 swriden_first_name       CHAR,
 swriden_mi               CHAR,
 swriden_change_ind       CHAR)
```

Exercise 3

Run the SQL*Loader command line utility, either on your database server, or on your PC using the instruction in the supplied SQL-Loader exercise.doc.

```
$sqlldr <username>/<password> CONTROL = swriden.ctl
```

Section N: Answer Key for Self Check Exercises

Lesson: Section L – Answer Key (Continued)

Jump to TOC

Exercise 4

Examine your log file. What was the success rate? Which records, if any, did not load correctly?

Record 4 should have errored because the ID was too long. All others should have been successfully inserted.

Exercise 5

What steps would you take to fix the records that had errors and reload the data?

You can go into your SWRIDEN.BAD file and fix the errors in the records that contained errors. Then, you could run your control file against the SWRIDEN.BAD file instead of the SWRIDEN.DAT file.

Section N: Answer Key for Self Check Exercises

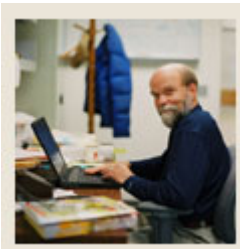
Lesson: Section L – Answer Key

Jump to TOC

Exercise 1

Create a SQL*Plus report that shows the Student ID, Name, and the courses for which they are registered (use TERM 199702). Make a separate page for each Student. Create a title with your institution name and a subtitle of 'Registered Courses'. Include the term in the title.

```
SET FEEDBACK OFF
SET ECHO OFF
CLEAR COLUMNS
CLEAR COMPUTES
CLEAR BREAKS
COLUMN NAME FORMAT a25 HEADING 'Name'
COLUMN SWVCRSE_DESC FORMAT a35 HEADING 'Course Name'
COLUMN SWRREGS_CRN FORMAT 99999 HEADING 'Course|Number'
SET LINESIZE 80
SET PAGESIZE 60
COLUMN SWRREGS_TERM_CODE NEW_VALUE TRMVAL NOPRINT
TTITLE CENTER 'BANNER University' RIGHT 'Page: ' -
FORMAT 999 SQL.PNO SKIP 1 CENTER 'Registered Courses' SKIP 2 -
CENTER 'Term Code: ' TRMVAL SKIP 3
BREAK ON SWRIDEN_ID SKIP PAGE ON NAME
SPOOL REGISTR.RPT
SELECT SWRREGS_TERM_CODE, SWRIDEN_ID,
       SWRIDEN_LAST_NAME||', '|| SWRIDEN_FIRST_NAME NAME,
       SWRREGS_CRN, SWVCRSE_DESC
  FROM SWRIDEN, SWVCRSE, SWRREGS
 WHERE SWRIDEN_PIDM = SWRREGS_PIDM
   AND SWRIDEN_CHANGE_IND IS NULL
   AND SWRREGS_CRN = SWVCRSE_CRN
   AND SWRREGS_TERM_CODE = '200502'
 ORDER BY SWRREGS_TERM_CODE, SWRIDEN_LAST_NAME,
          SWRIDEN_FIRST_NAME, SWRREGS_CRN;
SPOOL OFF
SET FEEDBACK ON
SET ECHO ON
```



Section N: Answer Key for Self Check Exercises

Lesson: Section L – Answer Key (Continued)

Jump to TOC

Exercise 1 (continued)

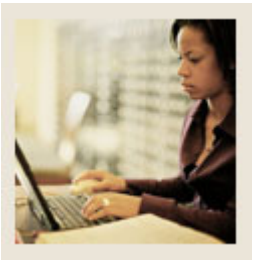
```
                              BANNER University                  Page:    1
                             Registered Courses

                              Term Code: 200502


                                  Course
SWRIDEN_I Name                    Number Course Name
--------- ------------------------ ------ ----------------------------------
857834585 Brown, Julie              10004 Physics
                                    10017 Management Information Systems

                              BANNER University                  Page:    2
                             Registered Courses

                              Term Code: 200502


                                  Course
SWRIDEN_I Name                    Number Course Name
--------- ------------------------ ------ ----------------------------------
892568211 Erickson, Ralph           10012 Statistics
                                    10019 Golf

                              BANNER University                  Page:    3
                             Registered Courses

                              Term Code: 200502


                                  Course
SWRIDEN_I Name                    Number Course Name
--------- ------------------------ ------ ----------------------------------
853954312 Johnson, Peter            10020 Swimming
                                    10021 Economics
...
```

Exercise 2

Using the same report as Exercise 1, create it as an HTML report.

```
SQL> SET MARKUP HTML ON SPOOL ON
```

Re-execute the same query.



Section O: Table Descriptions and Contents

Lesson: Overview

Jump to TOC

Introduction

This section provides listings for important tables and their contents.

Section contents

Section O: Table Descriptions and Contents

Lesson: SWRADDR

Jump to TOC

Description

```
SQL> DESC SWRADDR  - Student address & phone number table.
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
SWRADDR_PIDM                               NOT NULL NUMBER(8)
SWRADDR_ATYP_CODE                          NOT NULL VARCHAR2(2)
SWRADDR_STREET_LINE1                       NOT NULL VARCHAR2(30)
SWRADDR_STREET_LINE2                                VARCHAR2(30)
SWRADDR_CITY                                        VARCHAR2(20)
SWRADDR_STAT_CODE                                   VARCHAR2(3)
SWRADDR_ZIP                                         VARCHAR2(10)
SWRADDR_PHONE_AREA                                  VARCHAR2(3)
SWRADDR_PHONE_NUMBER                                VARCHAR2(7)
SWRADDR_ACTIVITY_DATE                      NOT NULL DATE
SWRADDR_USER_ID                                     VARCHAR2(30)
SWRADDR_DATA_ORIGIN                                 VARCHAR2(30)
```

Contents

```
SQL> SELECT * FROM SWRADDR;
SWRRAD SW SWRADDR_STREET_LINE1      SWRADDR_STREET_ SWRADDR_CITY    SWR SWRADDR_ZI SWR SWRADDR
------ -- ------------------------ --------------- --------------- --- ---------- --- -------
 12340 MA 506 Brown St.                             West Chester    PA  19380      610 5624789
 12341 PR 210 Pine St.                              San Francisco   CA  94104-2702 415 7954323
 12342 PR PO BOX 1035               1200 Elm Ln.    Lexington       KY  40223      859 5624789
 12343 P1 23 Market St.                             West Chester    PA  19382      610 3246734
 12344 MA 18 Chestnut Rd.                           NewOrleans      LA  70112      504 6743213
 12345 PR 1821 Canal St.                            New York        NY  10012      212 6452399
 12347 P1 21087 Streetville Rd.     Apt.5B          Fayetteville    AR  10012      870 6743213
 12350 MA 2300 Leaffall Rd.                         Dallas          TX  75202      469 6743213
 12352 PR 3 Ceasar Palace Ln.       2nd Fl.         Reno            NV  89502      775 6743213
 12358 MA 27 Macchiato Blvd.        15th Fl.        Seattle         WA  98108      206 6743213
 12356 P1 1 Titans Way                              Nashville       TN  37213      615 5654300
 12355 P1 43 W. Murray St.                          Bartlett        NH  03812      603 6743213
 12348 P1 21087 Streetville Rd.                     Chicago         IL  60605      312 6743213
 12349 P1 359 Mustang Ave.                          Dearborn        MI  48124      679 6743213
 12999 P1 7427 Cornell Pl.                          Wesmont         VA  23456      211 1234567

SWRADDR_A USERID         ORIGIN     SWRA
--------- -------------- ---------- ----
15-NOV-05 TRAIN_ORA101   TRAINING   USA
14-NOV-05 TRAIN_ORA101   TRAINING   USA
12-NOV-05 TRAIN_ORA101   TRAINING   USA
02-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
06-DEC-05 TRAIN_ORA101   TRAINING   USA
08-DEC-05 TRAIN_ORA101   TRAINING   USA
```

Section O: Table Descriptions and Contents

Lesson: SWBPERS

Jump to TOC

Description

```
SQL> DESC SWBPERS  -Student profile table.

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 SWBPERS_PIDM                              NOT NULL NUMBER(8)
 SWBPERS_SSN                                        VARCHAR2(9)
 SWBPERS_BIRTH_DATE                                 DATE
 SWBPERS_MRTL_CODE                                  VARCHAR2(1)
 SWBPERS_SEX                                        VARCHAR2(1)
 SWBPERS_CONFID_IND                                 VARCHAR2(1)
 SWBPERS_ACTIVITY_DATE                     NOT NULL DATE
 SWBPERS_USER_ID                                    VARCHAR2(30)
 SWBPERS_DATA_ORIGIN                                VARCHAR2(30)
```

Contents

```
SQL> SELECT * FROM SWBPERS;

SWBPERS_PIDM SWBPERS_S SWBPERS_B S S S SWBPERS_A SWBPERS_USER SWBPERS_
------------ --------- --------- - - - --------- ------------ --------
       12340 585442212 02-AUG-73 S F Y 31-OCT-05 TRAIN_ORA101 TRAINING
       12341 682082678 12-NOV-70 M M N 10-DEC-05 TRAIN_ORA101 TRAINING
       12343 555444412 22-SEP-73 S F Y 05-DEC-05 TRAIN_ORA101 TRAINING
       12344 198767345 30-OCT-73 S M N 08-DEC-05 TRAIN_ORA101 TRAINING
       12345 955433412 05-JAN-84 W F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12346 643091257 15-FEB-61 D F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12348 231560987 25-MAR-41 M F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12350 340541234 01-APR-76 W F N 07-DEC-05 TRAIN_ORA101 TRAINING
       12352 189054387 19-MAY-78 D M N 07-DEC-05 TRAIN_ORA101 TRAINING
       12353 035341098 29-JUN-65 M M N 07-DEC-05 TRAIN_ORA101 TRAINING
       12355 608321875 04-JUL-60 W F Y 07-DEC-05 TRAIN_ORA101 TRAINING
       12357 430896512 27-NOV-52 M M Y 07-DEC-05 TRAIN_ORA101 TRAINING
       12359 318760932 31-DEC-54 D F Y 07-DEC-05 TRAIN_ORA101 TRAINING
```

Section O: Table Descriptions and Contents

Lesson: SWRIDEN

Jump to TOC

Description

```
SQL> DESC SWRIDEN  – Student master identification table.

Name                                      Null?    Type
----------------------------------------- -------- ----------------
 SWRIDEN_PIDM                             NOT NULL NUMBER(8)
 SWRIDEN_ID                               NOT NULL VARCHAR2(9)
 SWRIDEN_LAST_NAME                        NOT NULL VARCHAR2(25)
 SWRIDEN_FIRST_NAME                                VARCHAR2(15)
 SWRIDEN_MI                                        VARCHAR2(15)
 SWRIDEN_CHANGE_IND                                VARCHAR2(1)
 SWRIDEN_ACTIVITY_DATE                    NOT NULL DATE
 SWRIDEN_USER_ID                                   VARCHAR2(30)
 SWRIDEN_DATA_ORIGIN                               VARCHAR2(30)
```

Contents

```
SQL> SELECT * FROM SWRIDEN;
```

SWRIDE	SWRIDEN_I	SWRIDEN_LAST_NA	SWRIDEN_FIRST_N	SWRIDEN_MI	S	SWRIDEN_A	SWRIDEN_USER	SWRIDEN_DA
12340	857834585	Brown	Julie	K		20-DEC-05	TRAIN_ORA101	TRAINING
12340	876536782	Brown	Julie	K	I	20-DEC-05	TRAIN_ORA101	TRAINING
12341	882993466	Smith	Robert	E		20-DEC-05	TRAIN_ORA101	TRAINING
12342	853954312	Johnson	Peter	S		20-DEC-05	TRAIN_ORA101	TRAINING
12343	845672112	Jones	Sandy	J	N	20-DEC-05	TRAIN_ORA101	TRAINING
12343	845672112	Jones-Erickson	Sandy	J		20-DEC-05	TRAIN_ORA101	TRAINING
12344	892568211	Erickson	Ralph	L		20-DEC-05	TRAIN_ORA101	TRAINING
12345	878549991	Erickson	Susan	T		20-DEC-05	TRAIN_ORA101	TRAINING
12346	843853339	White	Nancy	Carol		20-DEC-05	TRAIN_ORA101	TRAINING
12347	822874301	Marx	Joan	Elizabeth		20-DEC-05	TRAIN_ORA101	TRAINING
12348	862100933	Clifford	Stephanie	Geena		20-DEC-05	TRAIN_ORA101	TRAINING
12349	881337923	Serum	Tracy	Paige		20-DEC-05	TRAIN_ORA101	TRAINING
12350	817253082	Dukes	Michelle	Q		20-DEC-05	TRAIN_ORA101	TRAINING
12350	817253082	Vaughn	Michelle	Q	N	20-DEC-05	TRAIN_ORA101	TRAINING
12351	029348721	Dukes	Frederick	C	I	20-DEC-05	TRAIN_ORA101	TRAINING
12351	029348721	Jordan	Marcus	I	N	20-DEC-05	TRAIN_ORA101	TRAINING
12351	872109834	Dukes	Frederick	C		20-DEC-05	TRAIN_ORA101	TRAINING
12352	825110988	Johnson	Jeremy	P		20-DEC-05	TRAIN_ORA101	TRAINING
12353	861232200	Johnson	Ian	G		20-DEC-05	TRAIN_ORA101	TRAINING
12354	831603288	Bristow	Brandon	A		20-DEC-05	TRAIN_ORA101	TRAINING
12355	855231118	McNair	Tracy	A		20-DEC-05	TRAIN_ORA101	TRAINING
12355	855231118	Goode	Tracy	A	N	20-DEC-05	TRAIN_ORA101	TRAINING
12356	832092865	Miner	Christopher	U		20-DEC-05	TRAIN_ORA101	TRAINING
12357	832763321	Roberson	Devon	O		20-DEC-05	TRAIN_ORA101	TRAINING
12357	832736321	Roberson	Devon	O	I	20-DEC-05	TRAIN_ORA101	TRAINING
12359	853084511	Peterson	Amy	H		20-DEC-05	TRAIN_ORA101	TRAINING
12358	8486565	Jameson	Jennifer	W		28-DEC-05	TRAIN_ORA101	TRAINING

Section O: Table Descriptions and Contents

Lesson: SWRREGS

Jump to TOC

Description

```
SQL> DESC SWRREGS   --Student course registration table.

Name                                      Null?    Type
----------------------------------------- -------- -------------
SWRREGS_PIDM                              NOT NULL NUMBER(8)
SWRREGS_TERM_CODE                         NOT NULL VARCHAR2(6)
SWRREGS_CRN                               NOT NULL NUMBER(5)
SWRREGS_GPA                                        NUMBER(4,2)
SWRREGS_ACTIVITY_DATE                     NOT NULL DATE
```

Section O: Table Descriptions and Contents

Lesson: SWRREGS (Continued)

Jump to TOC

Contents

```
SQL> SELECT * FROM SWRREGS;

SWRREGS_PIDM    SWRREGS_TERM_CODE SWRREGS_CRN          SWRREGS_GPA      SWRREGS_ACTIVITY_DATE
--------------- ----------------- -------------------- ---------------- ----------------------
12340           200608            10001                3.2              25-SEP-07
12349           200608            10001                3.4              25-SEP-07
12349           200701            10007                3.1              25-SEP-07
12346           200702            10001                2.6              25-SEP-07
12348           200702            10001                2.9              25-SEP-07
12343           200702            10001                3                25-SEP-07
12340           200602            10007                2.1              25-SEP-07
12340           200702            10015                2.8              25-SEP-07
12340           200602            10017                                 25-SEP-07
12340           200602            10004                                 25-SEP-07
12340           200702            10008                2                25-SEP-07
12340           200702            10005                3                25-SEP-07
12340           200602            10009                3.2              25-SEP-07
12340           200602            10005                3.1              25-SEP-07
12341           200608            10014                3.2              25-SEP-07
12341           200701            10018                2.4              25-SEP-07
12341           200602            10023                1.2              25-SEP-07
12341           200602            10024                3.1              25-SEP-07
12342           200602            10021                                 25-SEP-07
12342           200602            10020                                 25-SEP-07
12342           200602            10013                2.6              25-SEP-07
12342           200602            10008                3.3              25-SEP-07
12342           200608            10011                3.3              25-SEP-07
12342           200608            10002                3.4              25-SEP-07
12342           200701            10033                2.3              25-SEP-07
12342           200701            10006                2.8              25-SEP-07
12342           200701            10009                2.9              25-SEP-07
12342           200602            10006                2.3              25-SEP-07
12343           200602            10008                3.8              25-SEP-07
12343           200701            10009                4                25-SEP-07
12343           200602            10011                1.6              25-SEP-07
12344           200602            10012                                 25-SEP-07
12344           200608            10019                                 25-SEP-07
12344           200602            10004                3.6              25-SEP-07
12345           200701            10003                2.2              25-SEP-07
12345           200702            10004                2.8              25-SEP-07
12345           200702            10011                2.8              25-SEP-07
12345           200602            10003                3.9              25-SEP-07
12345           200602            10009                3                25-SEP-07
12345           200701            10002                2.8              25-SEP-07
12346           200702            10016                1.1              25-SEP-07
12346           200602            10005                                 25-SEP-07
12346           200701            10015                3.5              25-SEP-07
12346           200701            10024                3.6              25-SEP-07

44 rows selected
```

Section O: Table Descriptions and Contents

Lesson: SWRSTDN

Jump to TOC

Description

```
SQL> DESC SWRSTDN  – Student standing table.

 Name                           Null?    Type
------------------------------- -------- ------------
SWRSTDN_PIDM                    NOT NULL NUMBER(8)
SWRSTDN_STDN_CODE               NOT NULL VARCHAR2(2)
SWRSTDN_DATE                    NOT NULL DATE
SWRSTDN_ACTIVITY_DATE           NOT NULL DATE
```

Contents

```
SQL> SELECT * FROM SWRSTDN;

SWRSTDN_PIDM SW SWRSTDN_D SWRSTDN_A
------------ -- --------- ---------
       12340 GR 23-FEB-05 23-FEB-05
       12341 GR 23-FEB-05 23-FEB-05
       12342 GH 23-FEB-05 23-FEB-05
       12343 PB 20-DEC-05 20-DEC-05
       12344 SS 20-DEC-05 20-DEC-05
       12345 GS 20-DEC-05 20-DEC-05
       12346 HS 20-DEC-05 20-DEC-05

7 rows selected.
```

Section O: Table Descriptions and Contents

Lesson: SWRTEST

Jump to TOC

Description

```
SQL> DESC SWRTEST  – Student test score table.

 Name                           Null?    Type
------------------------------- -------- --------------
SWRTEST_PIDM                    NOT NULL NUMBER(8)
SWRTEST_TEST_DATE               NOT NULL DATE
SWRTEST_SAT_VERBAL                       NUMBER(3)
SWRTEST_SAT_MATH                         NUMBER(3)
SWRTEST_ACTIVITY_DATE           NOT NULL DATE
```

Contents

```
SQL> SELECT * FROM swrtest;

SWRTEST_PIDM SWRTEST_T SWRTEST_SAT_VERBAL SWRTEST_SAT_MATH SWRTEST_A
------------ --------- ------------------ ---------------- ---------
       12340 01-MAR-05                550              480 31-OCT-05
       12341 03-MAR-05                530              580 31-OCT-05
       12342 13-FEB-05                660              520 31-OCT-05
       12341 08-JUN-05                590              610 20-NOV-05
       12343 03-FEB-05                530              420 11-OCT-05
       12345 10-DEC-05                590              620 31-OCT-05
       12346 20-NOV-05                630              590 31-OCT-05
       12346 10-DEC-05                520              460 18-DEC-05
       12347 30-NOV-05                                 550 15-DEC-05
       12345 30-NOV-05                500                 15-DEC-05
       12344 05-DEC-05                                    08-DEC-05

11 rows selected.
```

Section O: Table Descriptions and Contents

Lesson: SWVCRSE

Jump to TOC

Description

```
SQL> DESC SWVCRSE  --Course validation table.

 Name                           Null?    Type
 ------------------------------ -------- -----------------
 SWVCRSE_CRN                    NOT NULL NUMBER(5)
 SWVCRSE_DESC                   NOT NULL VARCHAR2(30)
 SWVCRSE_CREDIT_HOURS                    NUMBER(3)
 SWVCRSE_ACTIVITY_DATE          NOT NULL DATE
```

Contents

```
SQL> SELECT * FROM SWVCRSE;

SWVCRSE_CRN SWVCRSE_DESC                    SWVCRSE_CREDIT_HOURS SWVCRSE_A
----------- ------------------------------ -------------------- ---------
      10001 Writing                                           3 20-DEC-05
      10002 European History                                  2 20-DEC-05
      10003 Algebra                                           4 20-DEC-05
      10004 Physics                                           3 20-DEC-05
      10005 Biology                                           3 20-DEC-05
      10006 Zoology                                           3 20-DEC-05
      10007 Philosophy                                        3 20-DEC-05
      10008 Psychology                                        3 20-DEC-05
      10009 Calculus                                          4 20-DEC-05
      10010 Literature                                        2 20-DEC-05
      10011 Anthropology                                      4 20-DEC-05
      10012 Statistics                                        3 20-DEC-05
      10013 Oil Painting                                      2 20-DEC-05
      10014 Pottery                                           2 20-DEC-05
      10015 Speech                                            3 20-DEC-05
      10016 C Programming                                     4 20-DEC-05
      10017 Management Information Systems                    3 20-DEC-05
      10018 Tennis                                            2 20-DEC-05
      10019 Golf                                              2 20-DEC-05
      10020 Swimming                                          2 20-DEC-05
      10021 Economics                                         3 20-DEC-05
      10022 Accounting                                        3 20-DEC-05
      10023 Geometry                                          4 20-DEC-05
      10024 Photography                                       2 20-DEC-05

24 rows selected.
```

Section O: Table Descriptions and Contents

Lesson: SWVSTDN

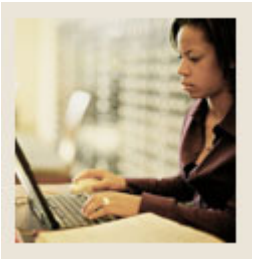
Jump to TOC

Description

```
SQL> DESC SWVSTDN  – Validation table for standing codes.

 Name                           Null?    Type
------------------------------- -------- ----------------
SWVSTDN_CODE                    NOT NULL VARCHAR2(2)
SWVSTDN_DESC                             VARCHAR2(30)
SWVSTDN_ACTIVITY_DATE           NOT NULL DATE
```

Contents

```
SQL> SELECT * FROM SWVSTDN;

SW SWVSTDN_DESC                    SWVSTDN_A
-- ------------------------------- ---------
GS Good Standing                   20-DEC-05
HS Honor Student                   20-DEC-05
PB Probation                       20-DEC-05
SS Suspended                       20-DEC-05
GR Graduate                        20-DEC-05
GH Graduate with honors            20-DEC-05

6 rows selected.
```



Section O: Table Descriptions and Contents

Lesson: SWVTERM

Jump to TOC

Description

```
SQL> DESC SWVTERM   – Validation table for term code.

 Name                                  Null?    Type
------------------------------------- -------- ----------------
SWVTERM_TERM_CODE                           NOT NULL VARCHAR2(6)
SWVTERM_DESC                                     VARCHAR2(30)
SWVTERM_ACTIVITY_DATE                 NOT NULL DATE
```

Contents

```
SQL> SELECT * FROM SWVTERM;

SWVTERM_TERM_CODE SWVTERM_DESC                   SWVTERM_ACTIVITY_DATE
----------------- ------------------------------ ---------------------
----
200501            Spring Semester 2005           25-SEP-07
200502            Spring Semester 2005           25-SEP-07
200502            Spring Semester 2005           25-SEP-07
200505            Summer Semester 2005           25-SEP-07
200508            Fall Semester 2005             25-SEP-07
200602            Spring Semester 2006           25-SEP-07
200605            Summer Semester 2006           25-SEP-07
200608            Fall Semester 2006             25-SEP-07
200608            Fall Semester 2006             25-SEP-07
200701            Spring Semester 2007           25-SEP-07
200702            Spring Semester 2007           25-SEP-07
200705            Summer Semester 2007           25-SEP-07
200708            Fall Semester 2007             25-SEP-07
200801            Spring Semester 2008           25-SEP-07
200805            Summer Semester 2008           25-SEP-07
200808            Fall Semester 2008             25-SEP-07
200901            Spring Semester 2009           25-SEP-07
200905            Summer Semester 2009           25-SEP-07
200908            Fall Semester 2009             25-SEP-07

19 rows selected
```

Section O: Table Descriptions and Contents

Lesson: TWRACCD

Jump to TOC

Description

```
SQL> DESC TWRACCD   --Account detail table.

 Name                           Null?    Type
------------------------------- -------- ----------------
TWRACCD_PIDM                    NOT NULL NUMBER(8)
TWRACCD_TERM_CODE               NOT NULL VARCHAR2(6)
TWRACCD_DETC_CODE               NOT NULL VARCHAR2(4)
TWRACCD_TRANS_TYPE              NOT NULL VARCHAR2(1)
TWRACCD_BILL_DATE               NOT NULL DATE
TWRACCD_AMOUNT                  NOT NULL NUMBER(7,2)
TWRACCD_BALANCE                          NUMBER(7,2)
TWRACCD_PAID_DATE                        DATE
TWRACCD_ACTIVITY_DATE           NOT NULL DATE
```

Section O: Table Descriptions and Contents

Lesson: TWVDETC

Jump to TOC

Description

```
SQL> DESC TWVDETC – Detail code validation table.
 Name                                      Null?    Type
----------------------------------------- -------- ---------------
TWVDETC_CODE                              NOT NULL VARCHAR2(4)
TWVDETC_DESC                                       VARCHAR2(30)
TWVDETC_ACTIVITY_DATE                     NOT NULL DATE
```

Contents

```
SQL> SELECT * FROM TWVDETC;

TWVD TWVDETC_DESC                      TWVDETC_A
---- ------------------------------ ---------
TUIT Tuition Charges                   20-DEC-05
BOOK Book Charges                      20-DEC-05
DORM Dorm Charges                      20-DEC-05
MEAL Meal Plan Charges                 20-DEC-05
LABS Lab Fee Charges                   20-DEC-05
CHEK Check Payment                     20-DEC-05
CASH Cash Payment                      20-DEC-05
CRED Credit Card Payment               20-DEC-05
FAID Financial Aid Payment             20-DEC-05

9 rows selected.
```

Section P: Related Files

Lesson: Overview

Jump to TOC

Introduction

This section provides listings for some important data files.

Section contents

Section P: Related Files

Lesson: Create_exercise_tables.sql

Jump to TOC

Listing

```
set feedback 6 echo off term on pause off
-- ********************************************************
-- Changed columns to use proper Banner naming conventions
-- Changed swbaddr to swraddr to be closer to Banner standard
-- DW 12/2005
-- Consolidated scripts for use with OR101, OR102 and OR106
-- ********************************************************
--prompt Setting up tables...please wait.
--
--PERSON INDENTIFICATION TABLE
--
DROP TABLE SWRIDEN;
CREATE TABLE SWRIDEN
(SWRIDEN_PIDM              NUMBER(8) NOT NULL,
 SWRIDEN_ID          VARCHAR2(9) NOT NULL,
 SWRIDEN_LAST_NAME   VARCHAR2(25) NOT NULL,
 SWRIDEN_FIRST_NAME         VARCHAR2(15),
 SWRIDEN_MI          VARCHAR2(15),
 SWRIDEN_CHANGE_IND         VARCHAR2(1),
 SWRIDEN_ACTIVITY_DATE      DATE NOT NULL,
 SWRIDEN_USER_ID            VARCHAR2(30),
 SWRIDEN_DATA_ORIGIN        VARCHAR2(30));
--
--
INSERT INTO SWRIDEN VALUES
(12340,'857834585','Brown','Julie','K',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12340,'876536782','Brown','Julie','K','I',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12341,'882993466','Smith','Robert','E',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12342,'853954312','Johnson','Peter','S',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12343,'845672112','Jones','Sandy','J','N',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12343,'845672112','Jones-Erickson','Sandy','J',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12344,'892568211','Erickson','Ralph','L',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12345,'878549991','Erickson','Susan','T',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12346,'843853339','White','Nancy','Carol',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12346,'843853339','LeBlanc','Nancy','C','N',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12347,'822874301','Marx','Joan','Elizabeth',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12348,'862100933','Clifford','Stephanie','Geena',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12349,'881337923','Serum','Tracy','Paige',NULL,SYSDATE,USER,'TRAINING');
```

```
INSERT INTO SWRIDEN VALUES
(12350,'817253082','Dukes','Michelle','Q',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12350,'817253082','Vaughn','Michelle','Q','N',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12351,'029348721','Dukes','Frederick','C','I',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12351,'029348721','Jordan','Marcus','I','N',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12351,'872109834','Dukes','Frederick','C',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12352,'825110988','Johnson','Jeremy','P',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12353,'861232200','Johnson','Ian','G',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12354,'831603288','Bristow','Brandon','A',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12355,'855231118','McNair','Tracy','A',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12355,'855231118','Goode','Tracy','A','N',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12356,'832092865','Miner','Christopher','U',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12357,'832763321','Roberson','Devon','O',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12357,'832736321','Roberson','Devon','O','I',SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12358,'8486565','Jameson','Jennifer','W',NULL,SYSDATE,USER,'TRAINING');
INSERT INTO SWRIDEN VALUES
(12359,'853084511','Peterson','Amy','H',NULL,SYSDATE,USER,'TRAINING');
--
--GENERAL PERSON TABLE
--
DROP TABLE SWBPERS;
CREATE TABLE SWBPERS
(SWBPERS_PIDM              NUMBER(8)  not null,
 SWBPERS_SSN        VARCHAR2(9),
 SWBPERS_BIRTH_DATE        DATE,
 SWBPERS_MRTL_CODE  VARCHAR2(1),
 SWBPERS_SEX        VARCHAR2(1),
 SWBPERS_CONFID_IND        VARCHAR2(1),
 SWBPERS_ACTIVITY_DATE     DATE NOT NULL,
 SWBPERS_USER_ID           VARCHAR2(30),
 SWBPERS_DATA_ORIGIN       VARCHAR2(30));
--
--
INSERT INTO SWBPERS VALUES
(12340,'585442212',to_date('02-AUG-1973','DD-MON-YYYY'),'S','F','Y',SYSDATE-50,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12341,'682082678',to_date('12-NOV-1970','DD-MON-YYYY'),'M','M','N',SYSDATE-10,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12343,'555444412',to_date('22-SEP-1973','DD-MON-YYYY'),'S','F','Y',SYSDATE-15,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12344,'198767345',to_date('30-OCT-1973','DD-MON-YYYY'),'S','M','N',SYSDATE-12,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12345,'955433412',to_date('05-JAN-1984','DD-MON-YYYY'),'W','F','N',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12346,'643091257',to_date('15-FEB-1971','DD-MON-YYYY'),'D','F','N',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
```

```
(12348,'231560987',to_date('25-MAR-1941','DD-MON-YYYY'),'M','F','N',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12350,'340541234',to_date('01-APR-1976','DD-MON-YYYY'),'W','F','N',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12352,'189054387',to_date('19-MAY-1978','DD-MON-YYYY'),'D','M','N',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12353,'035341098',to_date('29-JUN-1965','DD-MON-YYYY'),'M','M','N',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12355,'608321875',to_date('04-JUL-1960','DD-MON-YYYY'),'W','F','Y',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12357,'430896512',to_date('27-NOV-1962','DD-MON-YYYY'),'M','M','Y',SYSDATE-13,USER,
'TRAINING');
INSERT INTO SWBPERS VALUES
(12359,'318760932',to_date('31-DEC-1964','DD-MON-YYYY'),'D','F','Y',SYSDATE-13,USER,
'TRAINING');
--
--PERSON ADDRESS TABLE
--
DROP TABLE SWRADDR;
CREATE TABLE SWRADDR
(SWRADDR_PIDM              NUMBER(8)   NOT NULL,
 SWRADDR_ATYP_CODE         VARCHAR2(2) NOT NULL,
 SWRADDR_STREET_LINE1      VARCHAR2(30)NOT NULL,
 SWRADDR_STREET_LINE2      VARCHAR2(30),
 SWRADDR_CITY              VARCHAR2(20),
 SWRADDR_STAT_CODE  VARCHAR2(3),
 SWRADDR_ZIP        VARCHAR2(10),
 SWRADDR_PHONE_AREA        VARCHAR2(3),
 SWRADDR_PHONE_NUMBER      VARCHAR2(7),
 SWRADDR_ACTIVITY_DATE     DATE        NOT NULL,
 SWRADDR_USER_ID           VARCHAR2(30),
 SWRADDR_DATA_ORIGIN       VARCHAR2(30));
--
--
INSERT INTO SWRADDR VALUES
(12340,'MA','506 Brown St.',NULL,'West Chester','PA','19380','610','5624789',SYSDATE-
35,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12341,'PR','210 Pine St.',NULL,'San Francisco','CA','94104-
2702','415','7954323',SYSDATE-36,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12342,'PR','PO BOX 1035','1200 Elm
Ln.','Lexington','KY','40223','859','5624789',SYSDATE-38,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12343,'P1','23 Market St.',NULL,'West Chester','PA','19382','610','3246734',SYSDATE-
18,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12344,'MA','18 Chestnut Rd.',NULL,'New Orleans','LA','70112','504','6743213',SYSDATE-
14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12345,'PR','1821 Canal St.',NULL,'New York','NY','10012','212','6452399',SYSDATE-
14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12347,'P1','21087 Streetville Rd.','Apt.5B',
'Fayetteville','AR','40012','870','6743213',SYSDATE-14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12350,'MA','2300 Leaffall Rd.',NULL,'Dallas','TX','75202','469','6743213',SYSDATE-
14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
```

```
(12352,'PR','3 Ceasar Palace Ln.','2nd
Fl.','Reno','NV','89502','775','6743213',SYSDATE-14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12358,'MA','27 Macchiato Blvd.','15th
Fl.','Seattle','WA','98108','206','6743213',SYSDATE-14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12356,'P1','1 Titans Way',null,
'Nashville','TN','37213','615','5654300',SYSDATE-14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12355,'P1','43 W. Murray St.',null,
'Bartlett','NH','03812','603','6743213',SYSDATE-14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12348,'P1','21087 Streetville Rd.',null,
'Chicago','IL','60605','312','6743213',SYSDATE-14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12349,'P1','359 Mustang Ave.',null,
'Dearborn','MI','48124','679','6743213',SYSDATE-14,USER,'TRAINING');
INSERT INTO SWRADDR VALUES
(12999,'P1','7427 Cornell Pl.',null,
'Wesmont','VA','23456','211','1234567',SYSDATE-14,USER,'TRAINING');
--
--TELEPHONE NUMBER VIEW
CREATE OR REPLACE VIEW SWVTELE
(SWVTELE_PIDM, SWVTELE_NAME, SWVTELE_PHONE)
AS
SELECT SWRIDEN_PIDM,
       SWRIDEN_LAST_NAME||', '||SWRIDEN_FIRST_NAME||' ' || SWRIDEN_MI,
       '('||SWRADDR_PHONE_AREA||') '||SUBSTR(SWRADDR_PHONE_NUMBER,1,3)||'-'||
       SUBSTR(SWRADDR_PHONE_NUMBER,4,4)
FROM SWRIDEN, SWRADDR
WHERE SWRIDEN_PIDM = SWRADDR_PIDM
AND SWRIDEN_CHANGE_IND IS NULL;
--
--ACCOUNT TRANSACTION TABLE
--
DROP TABLE TWRACCD;
CREATE TABLE TWRACCD
(TWRACCD_PIDM             NUMBER(8) NOT NULL,
 TWRACCD_TERM_CODE  VARCHAR2(6) NOT NULL,
 TWRACCD_DETC_CODE  VARCHAR2(4) NOT NULL,
 TWRACCD_TRANS_TYPE        VARCHAR2(1) NOT NULL,
 TWRACCD_BILL_DATE  DATE NOT NULL,
 TWRACCD_AMOUNT            NUMBER(7,2) NOT NULL,
 TWRACCD_BALANCE           NUMBER(7,2),
 TWRACCD_PAID_DATE  DATE,
 TWRACCD_ACTIVITY_DATE     DATE NOT NULL);
--
--
INSERT INTO TWRACCD VALUES
(12340,'200602','TUIT','C',sysdate-130,1500.50,1500.50,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12340,'200602','BOOK','C',sysdate-145,300.20,300.20,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12340,'200608','BOOK','P',sysdate-312,700.00,-700.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12341,'200608','TUIT','C',sysdate-320,1100.00,1100.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12341,'200602','DORM','C',sysdate-310,500.00,500.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12341,'200602','CHEK','P',sysdate-90,1000.00,-1000.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12342,'200602','TUIT','C',sysdate-120,1300.00,1200.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12342,'200602','LABS','C',SYSDATE-120,50.00,50.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
```

```
(12342,'200602','MEAL','C',SYSDATE-220,800.50,800.50,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12343,'200602','TUIT','C',SYSDATE-300,800.00,800.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12343,'200602','FAID','P',SYSDATE-300,1100.00,-1100.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12344,'200702','TUIT','C',SYSDATE-215,750.00,750.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12344,'200702','BOOK','C',SYSDATE-210,400.00,400.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12344,'200702','LABS','C','10-SEP-2006',120.00,120.00,'1-NOV-2006',SYSDATE);
INSERT INTO TWRACCD VALUES
(12344,'200701','MEAL','C',SYSDATE-10,900.00,900.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12344,'200602','DORM','C','7-OCT-2006',1000.00,1000.00,'27-NOV-2006',SYSDATE);
INSERT INTO TWRACCD VALUES
(12344,'200701','CASH','P',SYSDATE-14,800.00,-800.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12344,'200701','CRED','P',SYSDATE-33,400.50,-400.50,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12345,'200701','TUIT','C',SYSDATE-25,300.00,300.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12342,'200602','MEAL','C',SYSDATE-90,400.50,400.50,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12346,'200701','TUIT','C',SYSDATE-17,900.00,900.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12346,'200701','LABS','C',SYSDATE-5,50.00,50.00,null,SYSDATE);
INSERT INTO TWRACCD VALUES
(12346,'200701','CHEK','P',SYSDATE-6,950.00,-950.00,null,SYSDATE);
--
-- TRANSACTION TYPE DETAIL CODE TABLE
DROP TABLE TWVDETC;
CREATE TABLE TWVDETC
(TWVDETC_CODE VARCHAR2(4) NOT NULL,
 TWVDETC_DESC VARCHAR2(30),
 TWVDETC_ACTIVITY_DATE DATE NOT NULL);
--
INSERT INTO TWVDETC VALUES ('TUIT', 'Tuition Charges', SYSDATE);
INSERT INTO TWVDETC VALUES ('BOOK', 'Book Charges', SYSDATE);
INSERT INTO TWVDETC VALUES ('DORM', 'Dorm Charges', SYSDATE);
INSERT INTO TWVDETC VALUES ('MEAL', 'Meal Plan Charges', SYSDATE);
INSERT INTO TWVDETC VALUES ('LABS', 'Lab Fee Charges', SYSDATE);
INSERT INTO TWVDETC VALUES ('CHEK', 'Check Payment', SYSDATE);
INSERT INTO TWVDETC VALUES ('CASH', 'Cash Payment', SYSDATE);
INSERT INTO TWVDETC VALUES ('CRED', 'Credit Card Payment', SYSDATE);
INSERT INTO TWVDETC VALUES ('FAID', 'Financial Aid Payment', SYSDATE);

--
-- TERM CODE TABLE
--
DROP TABLE SWVTERM;
CREATE TABLE SWVTERM
(SWVTERM_TERM_CODE  VARCHAR2(6) NOT NULL,
 SWVTERM_DESC              VARCHAR2(30),
 SWVTERM_ACTIVITY_DATE     DATE NOT NULL);
--
INSERT INTO SWVTERM VALUES ('200501', 'Spring Semester 2005', SYSDATE);
INSERT INTO SWVTERM VALUES ('200502', 'Spring Semester 2005', SYSDATE);
INSERT INTO SWVTERM VALUES ('200502', 'Spring Semester 2005', SYSDATE);
INSERT INTO SWVTERM VALUES ('200505', 'Summer Semester 2005', SYSDATE);
INSERT INTO SWVTERM VALUES ('200508', 'Fall Semester 2005', SYSDATE);
INSERT INTO SWVTERM VALUES ('200602', 'Spring Semester 2006', SYSDATE);
INSERT INTO SWVTERM VALUES ('200605', 'Summer Semester 2006', SYSDATE);
INSERT INTO SWVTERM VALUES ('200608', 'Fall Semester 2006', SYSDATE);
INSERT INTO SWVTERM VALUES ('200608', 'Fall Semester 2006', SYSDATE);
```

```
INSERT INTO SWVTERM VALUES ('200701', 'Spring Semester 2007', SYSDATE);
INSERT INTO SWVTERM VALUES ('200702', 'Spring Semester 2007', SYSDATE);
INSERT INTO SWVTERM VALUES ('200705', 'Summer Semester 2007', SYSDATE);
INSERT INTO SWVTERM VALUES ('200708', 'Fall Semester 2007', SYSDATE);
INSERT INTO SWVTERM VALUES ('200801', 'Spring Semester 2008', SYSDATE);
INSERT INTO SWVTERM VALUES ('200805', 'Summer Semester 2008', SYSDATE);
INSERT INTO SWVTERM VALUES ('200808', 'Fall Semester 2008', SYSDATE);
INSERT INTO SWVTERM VALUES ('200901', 'Spring Semester 2009', SYSDATE);
INSERT INTO SWVTERM VALUES ('200905', 'Summer Semester 2009', SYSDATE);
INSERT INTO SWVTERM VALUES ('200908', 'Fall Semester 2009', SYSDATE);
--
-- CLASS REGISTRATION TABLE
--
DROP TABLE SWRREGS;
CREATE TABLE SWRREGS
(SWRREGS_PIDM NUMBER(8) NOT NULL,
 SWRREGS_TERM_CODE VARCHAR2(6) NOT NULL,
 SWRREGS_CRN NUMBER(5) NOT NULL,
 SWRREGS_GPA NUMBER(4,2),
 SWRREGS_ACTIVITY_DATE DATE NOT NULL);
--
INSERT INTO SWRREGS VALUES(12340,'200608',10001,3.2,SYSDATE);
INSERT INTO SWRREGS VALUES(12349,'200608',10001,3.4,SYSDATE);
INSERT INTO SWRREGS VALUES(12349,'200701',10007,3.1,SYSDATE);
INSERT INTO SWRREGS VALUES(12346,'200702',10001,2.6,SYSDATE);
INSERT INTO SWRREGS VALUES(12348,'200702',10001,2.9,SYSDATE);
INSERT INTO SWRREGS VALUES(12343,'200702',10001,3.0,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200602',10007,2.1,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200702',10015,2.8,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200602',10017,NULL,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200602',10004,NULL,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200702',10008,2.0,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200702',10005,3.0,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200602',10009,3.2,SYSDATE);
INSERT INTO SWRREGS VALUES(12340,'200602',10005,3.1,SYSDATE);
INSERT INTO SWRREGS VALUES(12341,'200608',10014,3.2,SYSDATE);
INSERT INTO SWRREGS VALUES(12341,'200701',10018,2.4,SYSDATE);
INSERT INTO SWRREGS VALUES(12341,'200602',10023,1.2,SYSDATE);
INSERT INTO SWRREGS VALUES(12341,'200602',10024,3.1,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200602',10021,NULL,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200602',10020,NULL,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200602',10013,2.6,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200602',10008,3.3,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200608',10011,3.3,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200608',10002,3.4,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200701',10033,2.3,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200701',10006,2.8,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200701',10009,2.9,SYSDATE);
INSERT INTO SWRREGS VALUES(12342,'200602',10006,2.3,SYSDATE);
INSERT INTO SWRREGS VALUES(12343,'200602',10008,3.8,SYSDATE);
INSERT INTO SWRREGS VALUES(12343,'200701',10009,4.0,SYSDATE);
INSERT INTO SWRREGS VALUES(12343,'200602',10011,1.6,SYSDATE);
INSERT INTO SWRREGS VALUES(12344,'200602',10012,NULL,SYSDATE);
INSERT INTO SWRREGS VALUES(12344,'200608',10019,NULL,SYSDATE);
INSERT INTO SWRREGS VALUES(12344,'200602',10004,3.6,SYSDATE);
INSERT INTO SWRREGS VALUES(12345,'200701',10003,2.2,SYSDATE);
INSERT INTO SWRREGS VALUES(12345,'200702',10004,2.8,SYSDATE);
INSERT INTO SWRREGS VALUES(12345,'200702',10011,2.8,SYSDATE);
INSERT INTO SWRREGS VALUES(12345,'200602',10003,3.9,SYSDATE);
INSERT INTO SWRREGS VALUES(12345,'200602',10009,3.0,SYSDATE);
INSERT INTO SWRREGS VALUES(12345,'200701',10002,2.8,SYSDATE);
INSERT INTO SWRREGS VALUES(12346,'200702',10016,1.1,SYSDATE);
INSERT INTO SWRREGS VALUES(12346,'200602',10005,NULL,SYSDATE);
INSERT INTO SWRREGS VALUES(12346,'200701',10015,3.5,SYSDATE);
INSERT INTO SWRREGS VALUES(12346,'200701',10024,3.6,SYSDATE);
```

```
--
-- COURSE TABLE
--
DROP TABLE SWVCRSE;
CREATE TABLE SWVCRSE
(SWVCRSE_CRN        NUMBER(5) NOT NULL,
 SWVCRSE_DESC             VARCHAR2(30) NOT NULL,
 SWVCRSE_CREDIT_HOURS     NUMBER(3),
 SWVCRSE_ACTIVITY_DATE    DATE NOT NULL);
--
INSERT INTO SWVCRSE VALUES (10001, 'Writing', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10002, 'European History', 2,SYSDATE);
INSERT INTO SWVCRSE VALUES (10003, 'Algebra', 4,SYSDATE);
INSERT INTO SWVCRSE VALUES (10004, 'Physics', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10005, 'Biology', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10006, 'Zoology', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10007, 'Philosophy', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10008, 'Psychology', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10009, 'Calculus', 4,SYSDATE);
INSERT INTO SWVCRSE VALUES (10010, 'Literature', 2,SYSDATE);
INSERT INTO SWVCRSE VALUES (10011, 'Anthropology', 4,SYSDATE);
INSERT INTO SWVCRSE VALUES (10012, 'Statistics', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10013, 'Oil Painting', 2,SYSDATE);
INSERT INTO SWVCRSE VALUES (10014, 'Pottery', 2,SYSDATE);
INSERT INTO SWVCRSE VALUES (10015, 'Speech', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10016, 'C Programming', 4,SYSDATE);
INSERT INTO SWVCRSE VALUES (10017, 'Management Information Systems',3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10018, 'Tennis', 2,SYSDATE);
INSERT INTO SWVCRSE VALUES (10019, 'Golf', 2,SYSDATE);
INSERT INTO SWVCRSE VALUES (10020, 'Swimming', 2,SYSDATE);
INSERT INTO SWVCRSE VALUES (10021, 'Economics', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10022, 'Accounting', 3,SYSDATE);
INSERT INTO SWVCRSE VALUES (10023, 'Geometry', 4,SYSDATE);
INSERT INTO SWVCRSE VALUES (10024, 'Photography', 2,SYSDATE);
--
-- STUDENT STANDING TABLE
--
DROP TABLE SWRSTDN;
CREATE TABLE SWRSTDN
(SWRSTDN_PIDM NUMBER(8) NOT NULL,
 SWRSTDN_STDN_CODE VARCHAR2(2) NOT NULL,
 SWRSTDN_STDN_DATE DATE NOT NULL,
 SWRSTDN_ACTIVITY_DATE DATE NOT NULL);
--
INSERT INTO SWRSTDN VALUES(12340,'GR',SYSDATE-300,SYSDATE-300);
INSERT INTO SWRSTDN VALUES(12341,'GR',SYSDATE-300,SYSDATE-300);
INSERT INTO SWRSTDN VALUES(12342,'GH',SYSDATE-300,SYSDATE-300);
INSERT INTO SWRSTDN VALUES(12343,'PB',SYSDATE,SYSDATE);
INSERT INTO SWRSTDN VALUES(12344,'SS',SYSDATE,SYSDATE);
INSERT INTO SWRSTDN VALUES(12345,'GS',SYSDATE,SYSDATE);
INSERT INTO SWRSTDN VALUES(12346,'HS',SYSDATE,SYSDATE);
--
-- STUDENT STANDING CODE TABLE
--
DROP TABLE SWVSTDN;
CREATE TABLE SWVSTDN
(SWVSTDN_CODE VARCHAR2(2) NOT NULL,
 SWVSTDN_DESC VARCHAR2(30),
 SWVSTDN_ACTIVITY_DATE DATE NOT NULL);
--
INSERT INTO SWVSTDN VALUES ('GS', 'Good Standing', SYSDATE);
INSERT INTO SWVSTDN VALUES ('HS', 'Honor Student', SYSDATE);
INSERT INTO SWVSTDN VALUES ('PB', 'Probation', SYSDATE);
INSERT INTO SWVSTDN VALUES ('SS', 'Suspended', SYSDATE);
INSERT INTO SWVSTDN VALUES ('GR', 'Graduate', SYSDATE);
```

```
INSERT INTO SWVSTDN VALUES ('GH', 'Graduate with honors', SYSDATE);
--
-- STUDENT TEST TABLE
DROP TABLE SWRTEST;
CREATE TABLE SWRTEST
(SWRTEST_PIDM NUMBER(8) NOT NULL,
 SWRTEST_TEST_DATE DATE NOT NULL,
 SWRTEST_SAT_VERBAL NUMBER(3),
 SWRTEST_SAT_MATH NUMBER(3),
 SWRTEST_ACTIVITY_DATE DATE NOT NULL);
--
INSERT INTO SWRTEST VALUES(12340,'01-MAR-2006',550,480,SYSDATE-50);
INSERT INTO SWRTEST VALUES(12341,'03-MAR-2006',530,580,SYSDATE-50);
INSERT INTO SWRTEST VALUES(12342,'13-FEB-2006',660,520,SYSDATE-50);
INSERT INTO SWRTEST VALUES(12341,'08-JUN-2006',590,610,SYSDATE-30);
INSERT INTO SWRTEST VALUES(12343,'03-FEB-2006',530,420,SYSDATE-70);
INSERT INTO SWRTEST VALUES(12345,SYSDATE-10,590,620,SYSDATE-50);
INSERT INTO SWRTEST VALUES(12346,SYSDATE-30,630,590,SYSDATE-50);
INSERT INTO SWRTEST VALUES(12346,SYSDATE-10,520,460,SYSDATE-2);
INSERT INTO SWRTEST VALUES(12347,SYSDATE-20,null,550,SYSDATE-5);
INSERT INTO SWRTEST VALUES(12345,SYSDATE-20,500,null,SYSDATE-5);
INSERT INTO SWRTEST VALUES(12344,SYSDATE-15,null,null,SYSDATE-12);
--
-- Base Student Table
--
DROP TABLE SWBSTDN;
CREATE TABLE SWBSTDN
(SWBSTDN_PIDM                   NUMBER(8)     NOT NULL,
 SWBSTDN_TERM_CODE_EFF          VARCHAR2(6)   NOT NULL,
 SWBSTDN_STST_CODE              VARCHAR2(2)   NOT NULL,
 SWBSTDN_LEVL_CODE              VARCHAR2(2),
 SWBSTDN_ACTIVITY_DATE          DATE          NOT NULL,
 SWBSTDN_DATA_ORIGIN            VARCHAR2(30),
 SWBSTDN_USER_ID                VARCHAR2(30));
--
INSERT INTO SWBSTDN VALUES
(12341,'200110','AS','UG',to_date('15-MAR-2000','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12343,'200020','AS','GR',to_date('15-FEB-1999','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12345,'200230','IS','UG',to_date('15-JUL-2001','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12347,'200230','AS','MD',to_date('15-JUL-2001','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12349,'200110','GS','UG',to_date('15-MAR-2000','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12351,'200340','AS','GR',to_date('15-FEB-2002','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12353,'200340','IS','GR',to_date('15-FEB-2002','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12355,'200410','AS','UG',to_date('15-FEB-2003','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12357,'200420','AS','LW',to_date('15-FEB-2003','DD-MON-YYYY'),USER,'TRAINING');
INSERT INTO SWBSTDN VALUES
(12359,'200410','IS','UG',to_date('15-FEB-2003','DD-MON-YYYY'),USER,'TRAINING');
--
--
DROP TABLE SWRCMNT;
CREATE TABLE SWRCMNT
(SWRCMNT_PIDM               NUMBER(8)     NOT NULL,
 SWRCMNT_CMTT_CODE          VARCHAR2(3)   NOT NULL,
 SWRCMNT_TEXT               VARCHAR2(4000),
 SWRCMNT_USER_ID            VARCHAR2(30),
 SWRCMNT_ACTIVITY_DATE      DATE          NOT NULL,
 SWRCMNT_DATA_ORIGIN          VARCHAR2(30));
```

```
--
INSERT INTO swrcmnt VALUES
(12340,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12341,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12342,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12343,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12344,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12345,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12346,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12347,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12348,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12349,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12350,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12351,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12352,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12353,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12354,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12355,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12356,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12357,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12358,'100','HELLO',user,sysdate,'TRAINING');
INSERT INTO swrcmnt VALUES
(12359,'100','HELLO',user,sysdate,'TRAINING');
--
-- Base Employee Table
--
DROP TABLE PWBEMPL;
CREATE TABLE PWBEMPL
(PWBEMPL_PIDM                    NUMBER(8)     NOT NULL,
PWBEMPL_EMPL_STATUS              VARCHAR2(1)   NOT NULL,
PWBEMPL_ECLS_CODE                VARCHAR2(2)   NOT NULL,
PWBEMPL_FIRST_HIRE_DATE          DATE          NOT NULL,
PWBEMPL_CURRENT_HIRE_DATE        DATE          NOT NULL,
PWBEMPL_TERM_DATE                DATE,
PWBEMPL_ACTIVITY_DATE            DATE          NOT NULL,
PWBEMPL_USER_ID                  VARCHAR2(30),
PWBEMPL_DATA_ORIGIN              VARCHAR2(30));
--
INSERT INTO PWBEMPL VALUES
(12340,'A','01',to_date('06-APR-1974','DD-MON-YYYY'),to_date('31-DEC-1954','DD-MON-
YYYY'),  null,sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12342,'A','01',to_date('31-JAN-1954','DD-MON-YYYY'),to_date('31-JAN-1954','DD-MON-
YYYY'),  null,sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12344,'T','01',to_date('17-FEB-1954','DD-MON-YYYY'),to_date('17-FEB-1954','DD-MON-
YYYY'),  to_date('15-SEP-2004','DD-MON-YYYY'),sysdate-30,USER,'TRAINING');
```

```
INSERT INTO PWBEMPL VALUES
(12346,'A','01',to_date('22-JUN-1999','DD-MON-YYYY'),to_date('22-JUN-1999','DD-MON-
YYYY'),  null,sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12348,'A','01',to_date('02-JUN-2000','DD-MON-YYYY'),to_date('02-JUN-2000','DD-MON-
YYYY'),  null,sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12350,'A','01',to_date('20-JUL-2001','DD-MON-YYYY'),to_date('20-JUL-2001','DD-MON-
YYYY'),  null,sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12352,'T','01',to_date('14-JUL-2002','DD-MON-YYYY'),to_date('14-JUL-2002','DD-MON-
YYYY'), to_date('15-JUL-2002','DD-MON-YYYY'),sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12354,'A','01',to_date('26-OCT-1997','DD-MON-YYYY'),to_date('26-OCT-1997','DD-MON-
YYYY'),  null,sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12356,'T','01',to_date('01-NOV-1995','DD-MON-YYYY'),to_date('01-NOV-1995','DD-MON-
YYYY'),  to_date('01-MAY-1999','DD-MON-YYYY'),sysdate-30,USER,'TRAINING');
INSERT INTO PWBEMPL VALUES
(12358,'A','01',to_date('15-DEC-1992','DD-MON-YYYY'),to_date('15-DEC-1992','DD-MON-
YYYY'),  null,sysdate-30,USER,'TRAINING');
--
--
DROP TABLE TEMP;
CREATE TABLE TEMP
(COL1 NUMBER(8),
 COL2 VARCHAR2(15),
 COL3 DATE,
MESSAGE VARCHAR2(60));
--
DROP TABLE HIGH_VERBAL;
CREATE TABLE HIGH_VERBAL
(HIGH_VERBAL_PIDM NUMBER(8) NOT NULL,
 HIGH_VERBAL_VERBAL_SCORE NUMBER(3),
 HIGH_VERBAL_TEST_DATE DATE NOT NULL);
--
DROP TABLE HIGH_MATH;
CREATE TABLE HIGH_MATH
(HIGH_MATH_PIDM NUMBER(8) NOT NULL,
 HIGH_MATH_MATH_SCORE NUMBER(3),
 HIGH_MATH_TEST_DATE DATE NOT NULL);
--
DROP TABLE SWRRANK;
CREATE TABLE SWRRANK
(SWRRANK_PIDM NUMBER(8) NOT NULL,
 SWRRANK_CLASS_RANK NUMBER(5),
 SWRRANK_ACTIVITY_DATE DATE NOT NULL);
--
DROP TABLE SWRIDEN_HISTORY;
CREATE TABLE SWRIDEN_HISTORY
(SWRIDEN_HIST_PIDM NUMBER(8) NOT NULL,
 SWRIDEN_HIST_ID VARCHAR2(9) NOT NULL,
 SWRIDEN_HIST_LAST_NAME VARCHAR2(25) NOT NULL,
 SWRIDEN_HIST_FIRST_NAME VARCHAR2(15),
 SWRIDEN_HIST_MI VARCHAR2(15),
 SWRIDEN_HIST_CHANGE_IND VARCHAR2(1),
 SWRIDEN_HIST_ACTIVITY_DATE DATE NOT NULL);
--
DROP TABLE LOB_TABLE;
--
--prompt All done. Please alert the instructor of any errors other than ORA-00942.
set echo on pause on
```

Section P: Related Files

Lesson: swriden.ctl

Jump to TOC

Listing

```
LOAD DATA
INFILE 'swriden.dat'
BADFILE 'swriden.bad'
DISCARDFILE 'swriden.dsc'
APPEND
INTO TABLE swriden
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(swriden_pidm             SEQUENCE(MAX, 1),
 swriden_activity_date    SYSDATE,
 swriden_id               CHAR,
 swriden_last_name        CHAR,
 swriden_first_name       CHAR,
 swriden_mi               CHAR,
 swriden_change_ind       CHAR)
```

Section P: Related Files

Lesson: swriden.dat

Jump to TOC

Listing

```
443223344,Robertson,Robert,R,
433256789,Thompson,Thomas,T,I
433234566,Thompson,Thomas,T,
66755334533,Appleton,Apple,A,N
667553345,Thompson,Apple,A,
657890007,Jackson,Steve,D,
543678890,Hedges,Mike,R
453678923,Palm,Royal,Z
```

Release Date

Jump to TOC

This workbook was last updated on 10/22/2007.