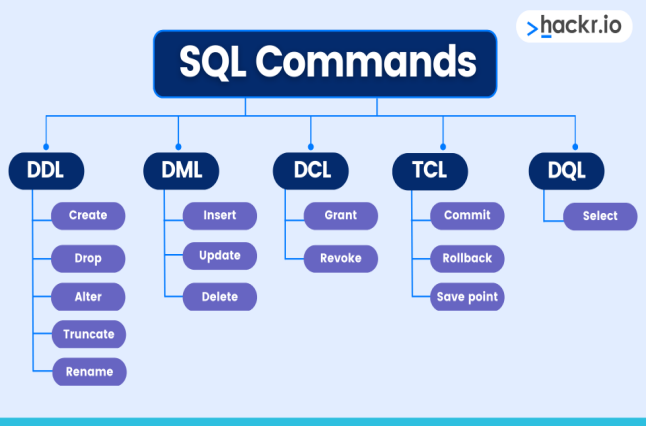
UNIT 3

SRM Institute of Science and Technology 1

**SQL commands, Views, Case Study: PL-SQL,**

****SRM Institute of Science and Technology 2

DDL COMMANDS

• 1. Data Definition Language (DDL)

• DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.

• All the command of DDL are auto-committed that means it permanently save all the changes in the database.

• Here are some commands that come under DDL:

• CREATE

• ALTER

• DROP

• TRUNCATE

• **a. CREATE** It is used to create a new table in the database.

• **Syntax:**

1. CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]); **b. DROP:** It is used to delete both the structure and record stored in the table. **Syntax**

DROP TABLE table\_name;

**Example**

DROP TABLE EMPLOYEE;

•

**c. ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute. **Syntax:**

To add a new column in the table

ALTER TABLE table\_name ADD column\_name COLUMN-definition; To modify existing column in the table:

ALTER TABLE table\_name MODIFY(column\_definitions....);

**EXAMPLE**

ALTER TABLE STU\_DETAILS ADD(ADDRESS VARCHAR2(20)); ALTER TABLE STU\_DETAILS MODIFY (NAME VARCHAR2(20));

**d. TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

**Syntax:**

TRUNCATE TABLE table\_name;

**Example:**

TRUNCATE TABLE EMPLOYEE;

DML COMMANDS

• 2. Data Manipulation Language

• DML commands are used to modify the database. It is responsible for all form of changes in the database. • The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

• Here are some commands that come under DML: • INSERT

• UPDATE

• DELETE

**a. INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table. **Syntax:**

1.INSERT INTO TABLE\_NAME

2.(col1, col2, col3,.... col N)

3.VALUES (value1, value2, value3, .... valueN);

Or

1.INSERT INTO TABLE\_NAME

2.VALUES (value1, value2, value3, .... valueN);

**For example:**

1.INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

**b. UPDATE:** This command is used to update or modify the value of a column in the table.

**Syntax:**

1.UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [W HERE CONDITION]

**For example:**

1.UPDATE students

2.SET User\_Name ='Sonoo'

3.WHERE Student\_Id = '3'

**c. DELETE:** It is used to remove one or more row from a table.

**Syntax:**

1.DELETE FROM table\_name [WHERE condition]; **For example:**

1.DELETE FROM javatpoint

2.WHERE Author="Sonoo";

3. Data Control Language

DCL commands are used to grant and take back authority from any database user. Here are some commands that come under DCL:

•Grant

•Revoke

• **a. Grant:** It is used to give user access privileges to a database.

• **Example**

1.GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_ USER, ANOTHER\_USER;

• **b. Revoke:** It is used to take back permissions from the user.

• **Example**

1.REVOKE SELECT, UPDATE ON MY\_TABLE FROM US ER1, USER2;

4.Transaction Control Language

• TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

• These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

• Here are some commands that come under TCL: •COMMIT

•ROLLBACK

•SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database. • **Syntax:**

COMMIT;

• **Example:**

1. DELETE FROM CUSTOMERS

2. WHERE AGE = 25;

3. COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

• **Syntax:**

ROLLBACK;

• **Example:**

1. DELETE FROM CUSTOMERS

2. WHERE AGE = 25;

3. ROLLBACK;

• **c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

1.SAVEPOINT SAVEPOINT\_NAME;

5. Data Query Language

• DQL is used to fetch the data from the database.

• It uses only one command:

• SELECT

• **a. SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause. • **Syntax:**

1. SELECT expressions

2. FROM TABLES

3. WHERE conditions;

• **For example:**

1. SELECT emp\_name

2. FROM employee

3. WHERE age > 20;

PL-SQL CASE STUDY

• Let's create a PL/SQL case study with an example. • In this case study, let's assume we are building a simple library management system.

• We want to create a stored procedure that calculates the late fee for books that are returned past their due date. • Here's the scenario:

1.We have a table named book\_loans with columns book\_id, due\_date, and return\_date.

2.We need to calculate the late fee for each book based on the number of days it is overdue.

3.The late fee is $0.50 per day for each book. Let's write a PL/SQL stored procedure for this case:

-- Create the table for book loans

CREATE TABLE book\_loans (

book\_id INT,

due\_date DATE,

return\_date DATE

);

-- Insert some sample data

INSERT INTO book\_loans VALUES (1, TO\_DATE('2024-01-20', 'YYYY-MM-DD'), TO\_DATE('2024-01-25', 'YYYY-MM-DD'));

INSERT INTO book\_loans VALUES (2, TO\_DATE('2024-01-15', 'YYYY-MM-DD'), TO\_DATE('2024-01-22', 'YYYY-MM-DD'));

INSERT INTO book\_loans VALUES (3, TO\_DATE('2024-01-10', 'YYYY-MM-DD'), TO\_DATE('2024-01-18', 'YYYY-MM-DD'));

-- Create a stored procedure to calculate late fees

CREATE OR REPLACE PROCEDURE calculate\_late\_fee AS

CURSOR book\_cursor IS

SELECT book\_id, due\_date, return\_date

FROM book\_loans;

v\_book\_id book\_loans.book\_id%TYPE;

v\_due\_date book\_loans.due\_date%TYPE;

v\_return\_date book\_loans.return\_date%TYPE;

v\_days\_late NUMBER;

v\_late\_fee NUMBER := 0.50;

BEGIN

OPEN book\_cursor;

LOOP

FETCH book\_cursor INTO v\_book\_id, v\_due\_date, v\_return\_date; EXIT WHEN book\_cursor%NOTFOUND;

-- Calculate the number of days late

v\_days\_late := v\_return\_date - v\_due\_date;

-- If the book is returned late, calculate and display the late fee IF v\_days\_late > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Book ID ' || v\_book\_id || ' is ' || v\_days\_late || ' days late.');

DBMS\_OUTPUT.PUT\_LINE('Late Fee: $' || v\_days\_late \* v\_late\_fee); ELSE

DBMS\_OUTPUT.PUT\_LINE('Book ID ' || v\_book\_id || ' was returned on time.'); END IF;

END LOOP;

CLOSE book\_cursor;

END calculate\_late\_fee;

/

-- Execute the stored procedure

EXEC calculate\_late\_fee

Explanation:

1.We create a table book\_loans to store information about book loans.

2.We insert some sample data into the table.

3.We create a stored procedure calculate\_late\_fee that uses a cursor to iterate through each book loan record. 4.Inside the loop, we calculate the number of days the book is overdue.

5.If the book is overdue, we print a message with the book ID, days late, and the late fee.

6.If the book is returned on time, we print a message indicating that.

**Constraints with examples**

• SQL constraints are used to specify rules for the data in a table.

• Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

• Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

•NOT NULL - Ensures that a column cannot have a NULL value

•UNIQUE - Ensures that all values in a column are different

•PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

•FOREIGN KEY - Prevents actions that would destroy links between tables

•CHECK - Ensures that the values in a column satisfies a specific condition

•DEFAULT - Sets a default value for a column if no value is specified

•CREATE INDEX - Used to create and retrieve data from the database very quickly

SQL NOT NULL CONSTRAINT

• SQL NOT NULL on CREATE TABLE

• Example

• CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL, FirstName varchar(255) NOT NULL Age int

);

• SQL NOT NULL on ALTER TABLE

• ALTER TABLE Persons

ALTER COLUMN Age int NOT NULL;

**2. UNIQUE –**

This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.

For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID.

CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20)

);

**3. PRIMARY KEY –**

• Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints. •

A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.

CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20),

PRIMARY KEY(ID)

);

**4. FOREIGN KEY –**

A foreign key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.

Consider the two tables as shown below:

**O\_ID ORDER\_NO C\_ID**

**1 2253 3**

**2 3325 3**

**3 4521 2**

**4 8532 1**

**C\_ID NAME ADDRESS**

**1 RAMESH DELHI**

**2 SURESH NOIDA**

**3 DHARMESH GURGAON**

As we can see clearly that the field C\_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.

Syntax:

CREATE TABLE Orders

(

O\_ID int NOT NULL,

ORDER\_NO int NOT NULL,

C\_ID int, PRIMARY KEY (O\_ID),

FOREIGN KEY (C\_ID)

REFERENCES Customers(C\_ID)

)

**(i) CHECK –**

Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.

For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18 ). That is, the user will not be allowed to enter any record in the table with AGE < 18. Check constraint in detail

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int NOT NULL CHECK (AGE >= 18)

);

**(ii) DEFAULT –**

This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.

For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

CREATE TABLE Student

( ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int DEFAULT 18

);

**JOINS AND ITS TYPES**

• **SQL Join** statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

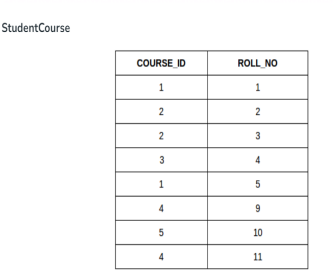
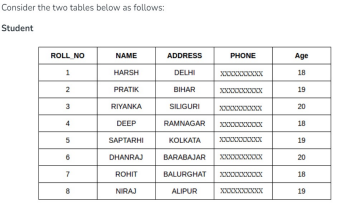
• INNER JOIN

• LEFT JOIN

• RIGHT JOIN

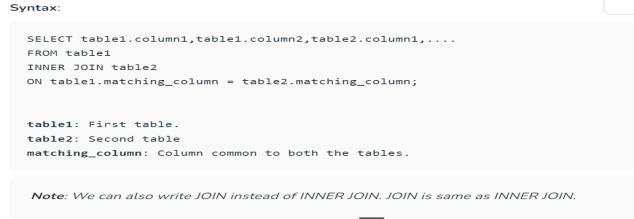
• FULL JOIN

• NATURAL JOIN

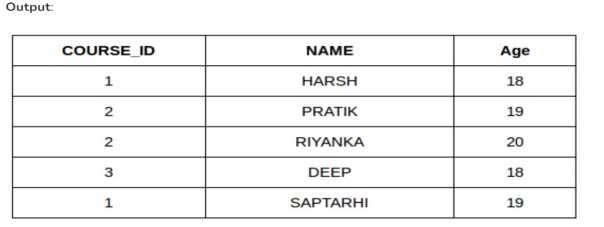
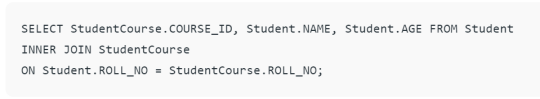


**A. INNER JOIN**

• The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

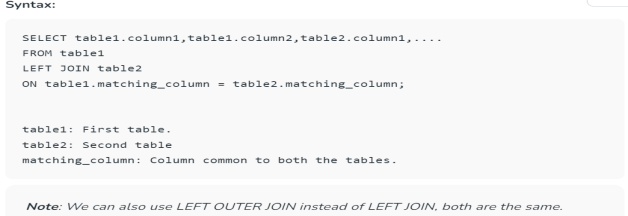


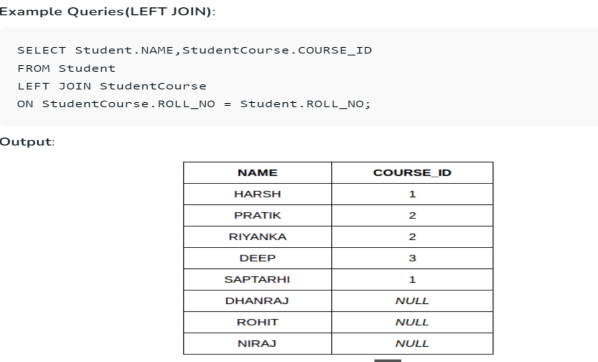
**Example Queries(INNER JOIN)**

This query will show the names and age of students enrolled in different courses.

**B. LEFT JOIN**

• This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.





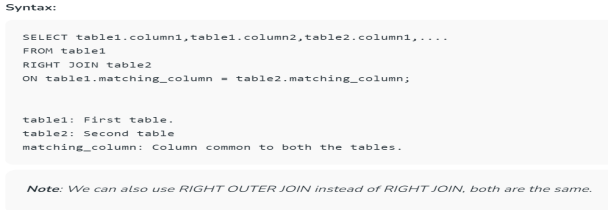
• **C. RIGHT JOIN**

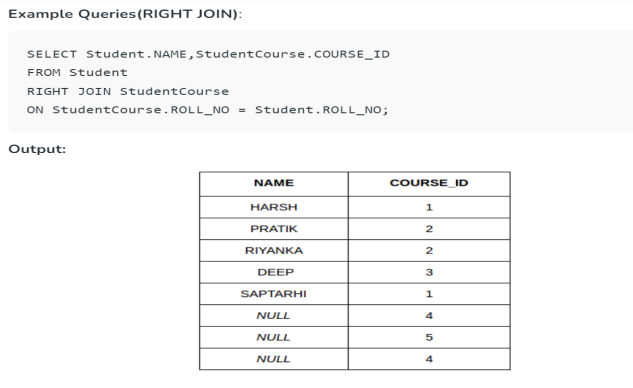
• RIGHT JOIN is similar to LEFT JOIN.

• This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.

• For the rows for which there is no matching row on the left side, the result-set will contain *null*.

• RIGHT JOIN is also known as RIGHT OUTER JOIN.





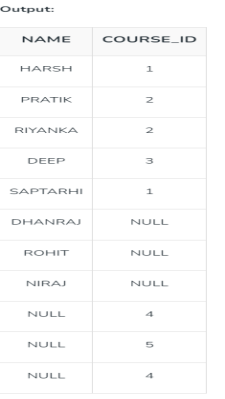
• **D. FULL JOIN**

• FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN.

• The result-set will contain all the rows from both tables.

• For the rows for which there is no matching, the result-set will contain *NULL* values.





• **E. Natural join (?)**

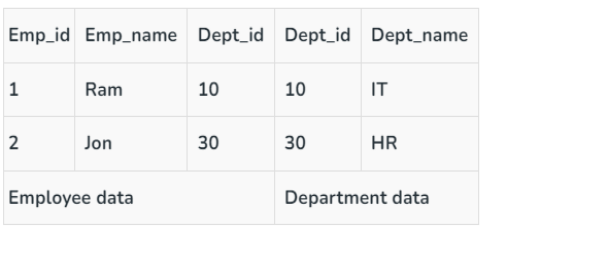
• Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

• Both table must have at list one common column with same column name and same data type.

• The two table are joined using Cross join.

• DBMS will look for a common column with same name and data type Tuples having exactly same values in common columns are kept in result.





**SET OPERATIONS**

• The SQL Set operation is used to combine the two or more SQL SELECT statements.

• Types of Set Operation

1.Union

2.UnionAll

3.Intersect

4.Minus

**1.Union**

• The SQL Union operation is used to combine the result of two or more SQL SELECT queries.

• In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.

• The union operation eliminates the duplicate rows from its resultset.

**Syntax**

1.SELECT column\_name FROM table1

2.UNION

3.SELECT column\_name FROM table2;









**3. Intersect**

• It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.

• In the Intersect operation, the number of datatype and columns must be the same.

• It has no duplicates and it arranges the data in ascending order by default.



**4. Minus**

• It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.

• It has no duplicates and data arranged in ascending order by default.

**Sub queries-Nested Queries**

• A Subquery is a query within another SQL query and embedded within the WHERE clause.

• **Important Rule:**

• A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.

• You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

• A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.

• Subqueries are on the right side of the comparison operator. • A subquery is enclosed in parentheses.

• In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

1. Subqueries with the Select Statement

• SQL subqueries are most frequently used with the Select statement. **Syntax**

****

****

2. Subqueries with the INSERT Statement

• SQL subquery can also be used with the Insert statement.

• In the insert statement, data returned from the subquery is used to insert into another table.In the subquery, the selected data can be modified with any of the character, date functions.



3. Subqueries with the UPDATE Statement

• The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.



• This would impact three rows, and finally, the EMPLOYEE table would have the following records.



4. Subqueries with the DELETE Statement

• The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.





**Views**

****

• Views in SQL are considered as a virtual table. A view also contains rows and columns.

• To create the view, we can select the fields from one or more tables present in the database.

• A view can either have specific rows based on certain condition or all the rows of a table.



1. Creating view

• A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

**Syntax:**

**CREATE VIEW** view\_name **AS**

**SELECT** column1, column2.....

**FROM** table\_name

**WHERE** condition;

• **2. Creating View from a single table**

• In this example, we create a View named DetailsView from the table Student\_Detail.



**3. Creating View from multiple tables**

• View from multiple tables can be created by simply include multiple tables in the SELECT statement.

• In the given example, a view is created named MarksView from two tables Student\_Detail and Student\_Marks. QUERY

**CREATE VIEW** MarksView **AS**

**SELECT** Student\_Detail.**NAME**, Student\_Detail.ADDRESS, Student\_Marks.MARKS **FROM** Student\_Detail, Student\_Mark

**WHERE** Student\_Detail.**NAME** = Student\_Marks.**NAME**;



4. Deleting View

A view can be deleted using the Drop View statement. **DROP VIEW** view\_name; - SYNTAX



Fundamentals of PL/SQL

• Full-featured programming language

• Interpreted language

• Execute using Oracle 10*g* utilities

• SQL\*Plus

• Forms Builder

• Combines SQL queries with procedural commands • Reserved words

65

PL/SQL Variables and Data Types

• Variable names must follow the Oracle naming standard (Example: current\_s\_id, not $current\_s\_id) • Strongly typed language

• Explicitly declare each variable including data type before using variable

• Variable declaration syntax:

*variable\_name data\_type\_declaration*; • Default value always NULL

66

Scalar Variables

• Reference single value such as number, date, string • Data types correspond to Oracle 10*g* database data types • VARCHAR2 

• CHAR

• DATE

• NUMBER

• PL/SQL has other data types that do not correspond to database data types

67

Composite Variables

• Data object made up of multiple individual data elements

• Data structure contains multiple scalar variables • Composite variable data types include:

• RECORD (multiple scalar values similar to a table’s record)

A

R

• TABLE (tabular structure with multiple columns and R

rows)

A

• VARRAY (**v**ariable-sized **array**. Tabular structure that

Y

can expand or contract based on data values)

68

Reference Variables

• Directly reference specific database column or row • Assume data type of associated column or row • %TYPE data declaration syntax:

*variable\_name tablename.fieldname*%TYPE; • %ROWTYPE data declaration syntax:

*variable\_name tablename*%ROWTYPE;

LOB Data Type

• Must be manipulated using programs in DBMS\_LOB package

69

PL/SQL Program Blocks

• Declaration section

• Optional 

• Execution section

• Required

• Exception section

• Optional

• Comment statements

⮚Enclosed within /\* and \*/ for

several lines’ comments

⮚-- for single line comments /\* Script: Student register Purpose: to enroll students

in class \*/

-- Script: Student register

-- Purpose: to enroll

students

70

PL/SQL Arithmetic Operators in Describing Order of Precedence



• Parentheses are used to force PL/SQL interpreter to evaluate operations in a certain order

total\_hours\_worked - 40 \* over\_time\_rate

(total\_hours\_worked – 40) \* over\_time\_rate

Questions: 2 \* 2 \*\* 2 = ? 100 / 2 \* 5 = ?

71

Assignment Statements

• Assigns value to variable • Operator: :=

DECLARE

variable1 NUMBER := 0; variable2 NUMBER := 0; BEGIN

variable2 := variable1 +1;

END;

• Syntax: *variable\_name* := *value*; • String literal within single quotation mark • Examples:

current\_s\_first\_name := ‘Tammy’;

current\_student\_ID NUMBER := 100;

Q: What is

• Result of adding a value to a NULL value is another the final

NULL value

• DEFAULT keyword can be used instead of assignment operator

value of variable2?

72

Displaying PL/SQL Program Output in SQL\*Plus

• PL/SQL output buffer

• Memory area on database server

• Stores program’s output values before they are displayed to user

• Default buffer size is 2000 bytes

• Should increase size if you want to display more than a few lines in SQL Plus to avoid buffer overflow error • Syntax: SET SERVEROUTPUT ON SIZE *buffer\_size* • Example: SET SERVEROUTPUT ON SIZE 4000

73

Displaying PL/SQL Program Output in SQL\*Plus (continued)

• DBMS\_OUTPUT

• is an Oracle built-in package

• Consists of a set of programs for processing output

• PUT\_LINE is the DBMS\_OUTPUT procedure for displaying output

• Syntax: DBMS\_OUTPUT.PUT\_LINE('*display\_text*'); • Example: DBMS\_OUTPUT.PUT\_LINE(current\_s\_first); • Displays maximum of 255 characters of text data • If try to display more than 255 characters, error occurs

74

Writing a PL/SQL Program

• Write PL/SQL program in Notepad or another text editor

• Indenting commands within each section is a good programming practice. **Will loose points if code is not indented**

• Copy and paste program commands from text editor into SQL\*Plus

• Press Enter after last program command • Type front slash ( **/** )

• Then press Enter again

75

PL/SQL Program Commands

76

PL/SQL Data Conversion Functions WHERE O\_DATE = TO\_DATE (‘29/05/2006’,

• Implicit data conversions

‘DD/MM/YYYY’)

WHERE O\_DATE =

• Interpreter automatically converts value from one data

‘29/05/2006’

type to another

• If PL/SQL interpreter unable to implicitly convert value error occurs

• Explicit data conversions

• Convert variables to different data types

• Using data conversion functions

77

Manipulating Character Strings

• Concatenating

• Joining two separate strings

• Operator: || (i.e. double bar)

• Syntax: *new\_string* := *string1* || *string2*; • Example: s\_fullname := s\_first || s\_last; • Parse

• Separate single string consisting of two data items separated by commas or spaces

s\_fullname := s\_first ||‘ ’|| s\_last; **Variable Data type Valu**

**e**

Bldg\_code VARCHAR2 LH Room\_num VARCHAR2 101 Room\_capacity NUMBER 150

room\_message := bldg\_code || ‘ Room ’ || room\_num || ‘ has ’ || TO\_CHAR(room\_capacity) || ‘seats.’;

**Question**: Write down the value of room\_message after the above Assignment statement is executed.

78

Removing Blank Leading and Trailing Spaces from Strings

• LTRIM function

• Remove blank leading spaces

• *string* := LTRIM(*string\_variable\_name*); • RTRIM function

• Remove blank trailing spaces

• *string* := RTRIM(*string\_variable\_name*);

DECLARE

s\_address CHAR(20) := ‘951 Raimbow Dr’;

BEGIN

s\_address := RTRIM(s\_address);

END;

**Questions:** How many characters will be removed from the string assigned to the s\_address variable when the RTRIM function in the avove PL/SQL block is executed

79

Finding the Length of Character Strings

• LENGTH function syntax

• *string\_length* := LENGTH(*string\_variable\_name*); • Example:

• code\_length as NUMBER(3):= LENGTH(bldg\_code);

• Q1: What will be the value of code\_length if bldg\_code’s value is ‘CR’?

• Q2: What will be the value of code\_length if bldg\_code’s value is ‘BUS ’?

80