Intro to **PL/SQL**

CS 341 Database Systems

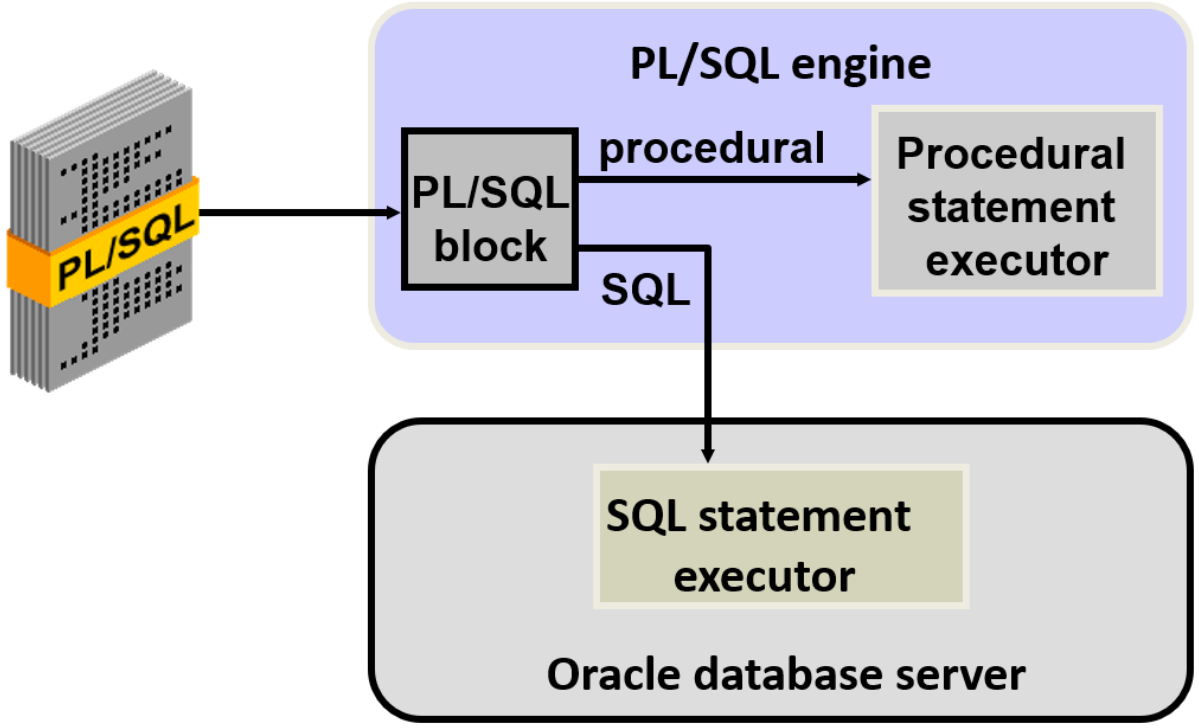
About PL/SQL

- PL/SQL:
  - Stands for “Procedural Language extension to SQL”
  - Oracle Corporation’s standard data access language for relational databases
  - Seamlessly integrates procedural constructs with SQL

What is PL/SQL?

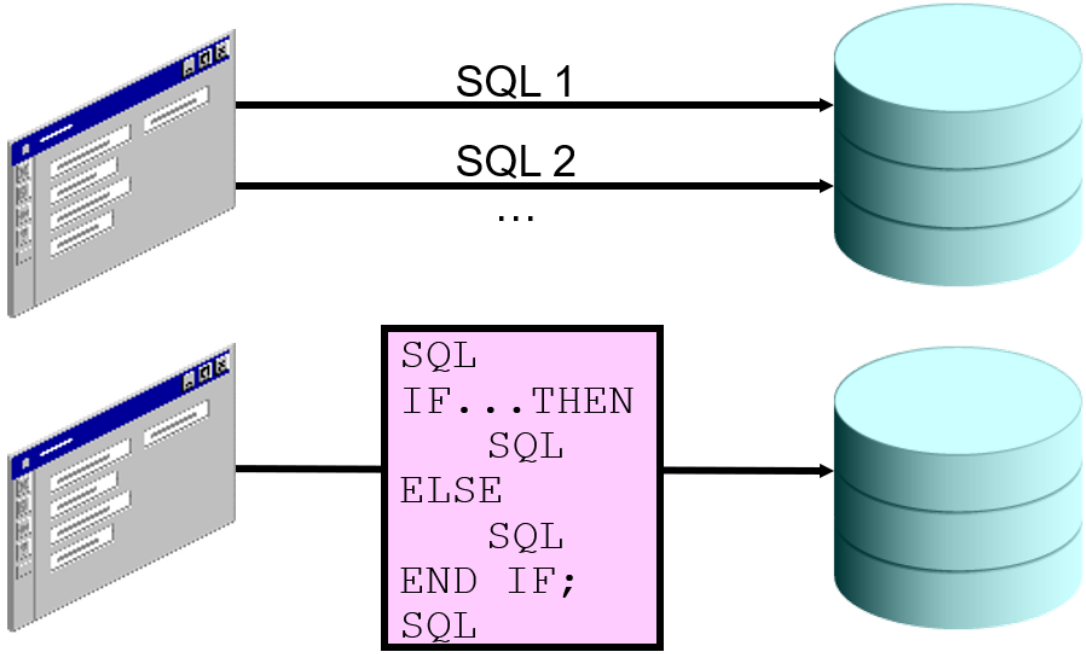
- PL/SQL stands for Procedural Language operating on or using SQL
- Combines the flexibility of SQL (4GL) with the power and configurability of the procedural constructs of a 3GL
- Extends SQL by adding 3GL constructs such as:
  - Variables and types (predefined and user defined)
  - Control Structures (IF-THEN-ELSE, Loops)
  - Procedures and functions
  - Object types and methods

PL/SQL Environment



Benefits of PL/SQL

- Integration of procedural constructs with SQL
- Improved performance

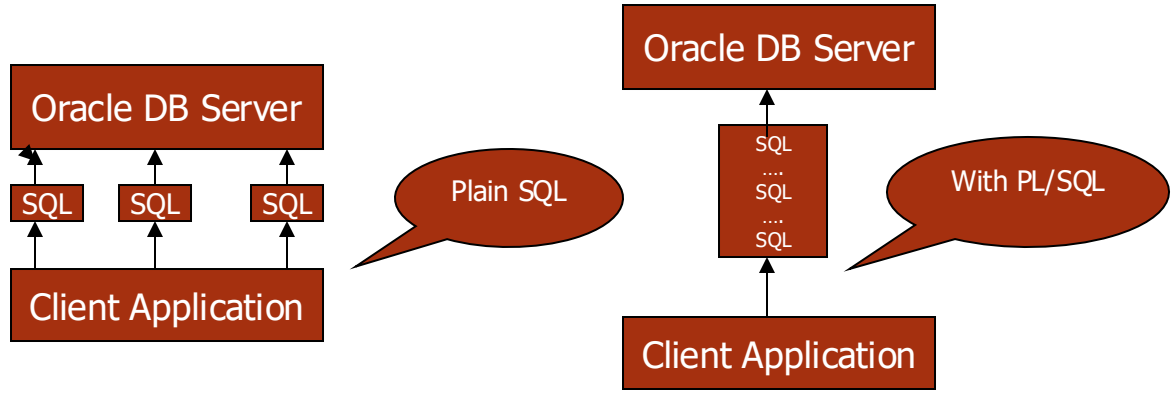


Benefits of PL/SQL

- Modularized program development
- Integration with Oracle tools
- Portability
- Exception handling

Client-Server Model

- SQL results in many network trips, one for each SQL statement
- PL/SQL permits several SQL statements to be bundled into a single block
- Results in fewer calls to database
  - Less network traffic
  - faster response time

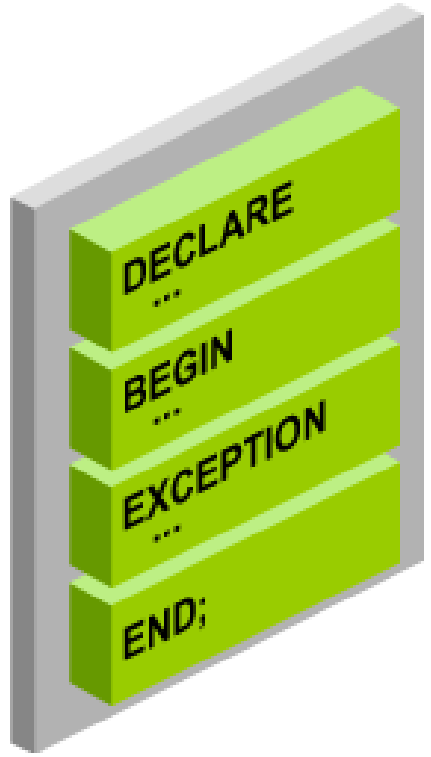


PL/SQL Block Structure

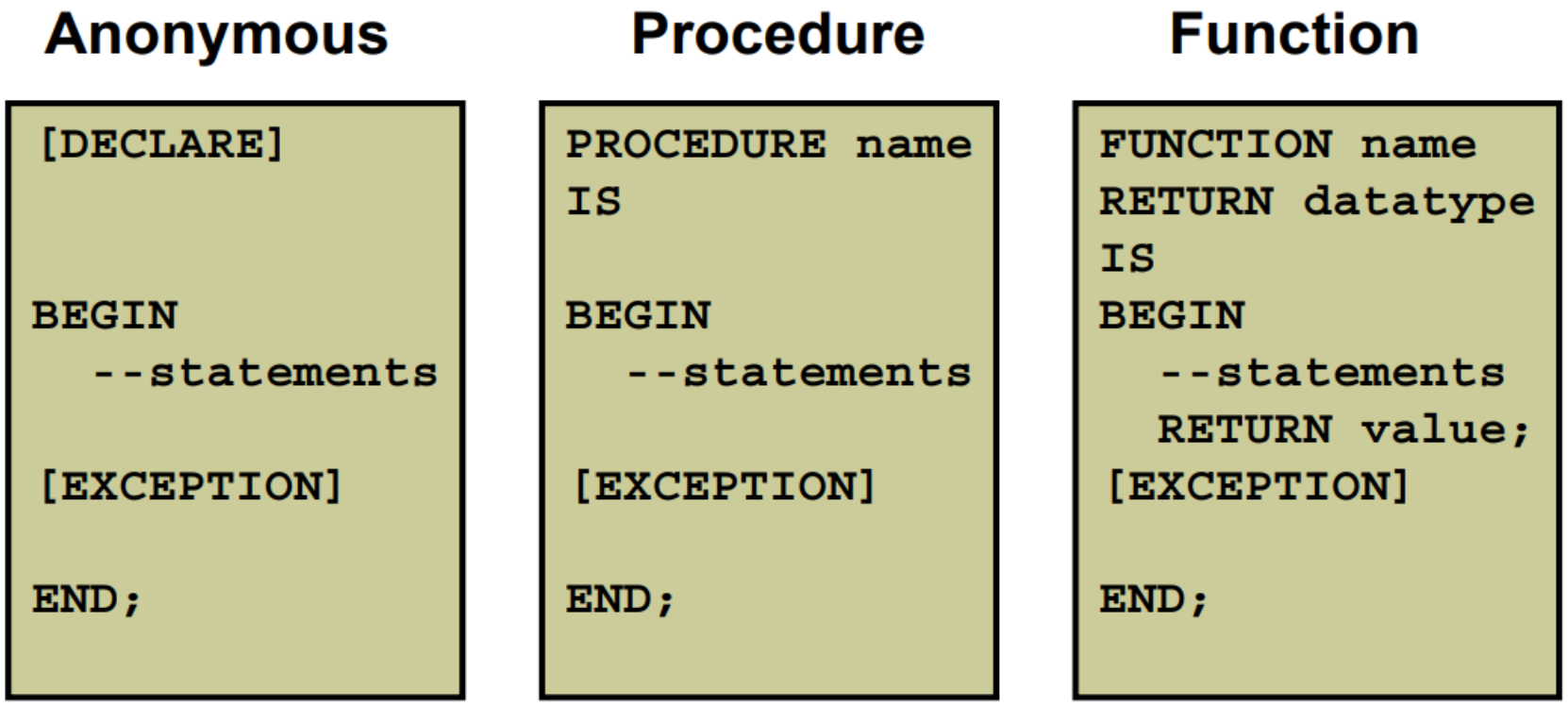
- Basic building block/unit of PL/SQL programs
  - Three possible sections of a block
    - Declarative section (optional)
    - Executable section (required)
      - Delimiters : BEGIN, END
    - Exception handling (optional)
- A block performs a logical unit of work in the program
- Blocks can be nested

PL/SQL Block Structure

- **DECLARE** (optional)
  - Variables, cursors, user-defined exceptions
- **BEGIN** (mandatory)
  - SQL statements
  - PL/SQL statements
- **EXCEPTION** (optional)
  - Actions to perform when errors occur
- **END**; (mandatory)



PL/SQL Block Types



Anonymous Block

```
DECLARE
f_name VARCHAR(20);

BEGIN
SELECT first_name INTO f_name FROM employees WHERE
employee_id=100;
END;
```

Stored Procedures

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(argument1 [mode1] datatype1,
  argument2 [mode2] datatype2,
  . . .)]
IS|AS
procedure_body;
```

Function

```
CREATE [OR REPLACE] FUNCTION function_name
 [(argument1 [mode1] datatype1,
  argument2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
function_body;
```

Test Output of PL/SQL Block

```
SET SERVEROUTPUT ON
…
DBMS_OUTPUT.PUT_LINE(' The First Name of the
Employee is ' || f_name);
…
```

```
SET SERVEROUTPUT ON

DECLARE
f_name VARCHAR(20);

BEGIN
SELECT first_name INTO f_name FROM employees WHERE
employee_id=100;
DBMS_OUTPUT.PUT_LINE(The First Name of the Employee is ' ||
f_name);
```

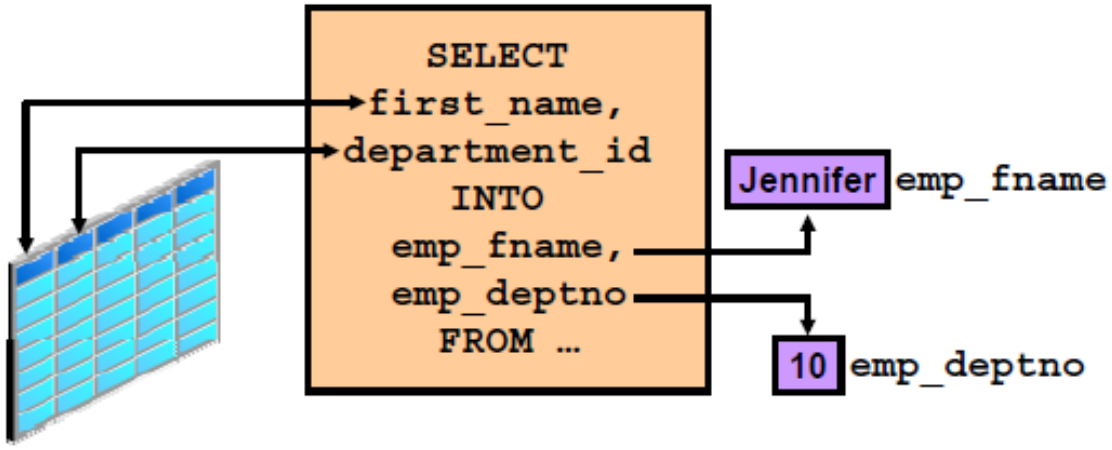
PL/SQL Constructs

- PL/SQL is based on Ada language constructs
  - Block Structure
  - Variables and Types
  - Conditionals
  - Looping Constructs
  - Cursors
  - Error Handling

Use of Variables

Variables can be used for:

- Temporary storage of data
- Manipulation of stored values
- Reusability



PL/SQL Variables and Types

- A variable is a named location in memory that:
  - can be read from
  - assigned a value in the program
  - 30 characters (length), no spaces, not case-sensitive
- Declared in the declarative section of the block
- Variables have a specific type associated with them
- Can be same type as database columns

Declaring and Initializing PL/SQL Variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
      [:= | DEFAULT expr];
```

Examples

```
DECLARE
  emp_hiredate    DATE;
  emp_deptno      NUMBER(2) NOT NULL := 10;
  location        VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

Assignment Operator

1

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
  Myname := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
END;
/
```

2

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20):= 'John';
BEGIN
  Myname := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
END;
/
```

Avoid Using column names as identifiers

```
DECLARE
  employee_id  NUMBER(6);
BEGIN
  SELECT     employee_id
  INTO       employee_id
  FROM       employees
  WHERE      last_name = 'Kochhar';
END;
/
```

%TYPE Attribute

- Is used to declare a variable according to:
  - A database column definition
  - Another declared variable
- Is prefixed with:
  - The database table and column
  - The name of the declared variable

Syntax

```
identifier      table.column_name%TYPE;
```

Examples

```
...
  emp_lname       employees.last_name%TYPE;
  balance         NUMBER(7,2);
  min_balance     balance%TYPE := 1000;
...
```

Consider the table:
PRODUCTS (**prod_id**, prod_name and prod_price).
Write a PL/SQL block that prints the name and price for product id=2.

Activity

```
SET serveroutput ON;
DECLARE
    v_prod_name  PRODUCTS.prod_name%TYPE;
    v_prod_price PRODUCTS.prod_price%TYPE;
BEGIN
    SELECT prod_name, prod_price
    INTO v_prod_name, v_prod_price
    FROM PRODUCTS
    WHERE prod_id = 2;

    DBMS_OUTPUT.PUT_LINE('The Product named'||v_prod_name||'is priced at'||v_prod_price);
END;
```

Activity

Bind Variables

- Created in the environment
- Also called host variables
- Created with the VARIABLE keyword
- Used in SQL statements and PL/SQL blocks
- Accessed even after the PL/SQL block is executed
- Referenced with a preceding colon

Example

```
VARIABLE emp_salary NUMBER
BEGIN
   SELECT salary  INTO :emp salary
   FROM  employees WHERE employee_id = 178;
END;
/
PRINT emp_salary
SELECT first_name, last_name FROM employees
WHERE salary=:emp_salary;
```

Substitution Variables

- Are used to get user input at run time
- Are referenced within a PL/SQL block with a preceding ampersand
- Are used to avoid hard-coding values that can be obtained at run time

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
  empno NUMBER(6):=&empno;
BEGIN
  SELECT salary  INTO :emp salary
  FROM  employees WHERE employee_id = empno;
END;
/
```

Nested Blocks

PL/SQL blocks can be nested.

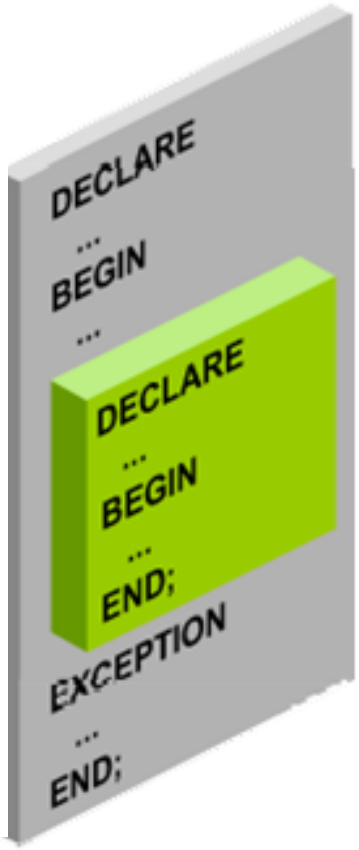
An executable section (BEGIN… END) can contain nested blocks.

An exception section can contain nested blocks.

Variables declared in a PL/SQL block are considered local to that block and global to all its subblocks.

Example

```
DECLARE
 outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
   inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
   DBMS_OUTPUT.PUT_LINE(inner_variable);
   DBMS_OUTPUT.PUT_LINE(outer_variable);
  END;
 DBMS_OUTPUT.PUT_LINE(outer_variable);
END;
/
```



Operators in PL/SQL

Operator	Operation
**	Exponentiation
+, -	Identity, negation
*, /	Multiplication, division
+, -, ||	Addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion

Indenting Code for Clarity

```
BEGIN
   IF x=0 THEN
      y:=1;
   END IF;
END;
/
```

```
DECLARE
   deptno        NUMBER(4);
   location_id   NUMBER(4);
BEGIN
   SELECT  department_id,
           location_id
   INTO    deptno,
           location_id
   FROM    departments
   WHERE   department_name
           = 'Sales';
...
END;
/
```

SELECT in PL/SQL

```
SELECT  select list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

- INTO clause is required
- Queries must return ONE row only.

```
SET SERVEROUTPUT ON
DECLARE
 fname VARCHAR2(25);
BEGIN
 SELECT first name INTO fname
 FROM employees WHERE employee_id=200;
 DBMS_OUTPUT.PUT_LINE(' First Name is : '||fname);
END;
/
```

```
DECLARE
 emp_hiredate    employees.hire_date%TYPE;
 emp_salary      employees.salary%TYPE;
BEGIN
  SELECT   hire date, salary
  INTO     emp_hiredate, emp_salary
  FROM     employees
  WHERE    employee_id = 100;
END;
/
```

Aggregate Functions

- Sum of the salary of all employees in a specified dept.

```
SET SERVEROUTPUT ON
DECLARE
   sum_sal   NUMBER(10,2);
   deptno    NUMBER NOT NULL := 60;
BEGIN
   SELECT  SUM(salary)  -- group function
   INTO sum_sal FROM employees
   WHERE  department_id = deptno;
   DBMS_OUTPUT.PUT_LINE ('The sum of salary is '
   || sum_sal);
END;
/
```

- NOT NULL to ensure the variable always receives a value.

INSERT in PL/SQL

- Insert new employee.

```
BEGIN
 INSERT INTO employees
  (employee_id, first_name, last_name, email,
   hire_date, job_id, salary)
   VALUES(employees_seq.NEXTVAL, 'Ruth', 'Cores',
   'RCORES',sysdate, 'AD_ASST', 4000);
END;
/
```

UPDATE in PL/SQL

- Increase salary of all employees who are stock clerks.

```
DECLARE
  sal_increase    employees.salary%TYPE := 800;
BEGIN
  UPDATE      employees
  SET         salary = salary + sal_increase
  WHERE       job_id = 'ST_CLERK';
END;
/
```

DELETE in PL/SQL

- Delete rows that belong to department 10 from the employees table

```
DECLARE
  deptno    employees.department id%TYPE := 10;
BEGIN
  DELETE FROM   employees
  WHERE  department_id = deptno;
END;
/
```

Session 02

PL/SQL

PL/SQL Conditionals

- PL/SQL supports conditional execution of statements
  - IF-THEN-ELSE
  - Nested IF-THEN-ELSE

IF Statement

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

IF Statement Examples

```
DECLARE
  myage number:=31;
BEGIN
  IF myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  END IF;
END;
/
```

PL/SQL procedure successfully completed.

```
SET SERVEROUTPUT ON
DECLARE
myage number:=31;
BEGIN
IF myage < 11
 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
 ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
END IF;
END;
/
```

I am not a child
PL/SQL procedure successfully completed.

IF ELSIF ELSE

```
DECLARE
myage number:=31;
BEGIN
IF myage < 11
 THEN
       DBMS_OUTPUT.PUT_LINE(' I am a child ');
   ELSIF myage < 20
     THEN
       DBMS_OUTPUT.PUT_LINE(' I am young ');
   ELSIF myage < 30
     THEN
       DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
   ELSIF myage < 40
     THEN
       DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
 ELSE
    DBMS_OUTPUT.PUT_LINE(' I am always young ');
END IF;
END;
/
```

I am in my thirties
PL/SQL procedure successfully completed.

Example: Conditional Statement

Step #1: Create table with appropriate datatype: **temp_table (message)**

Step #2:

```
DECLARE
     v_TotalEmployees NUMBER;
BEGIN
  SELECT COUNT(*)
    INTO v_TotalEmployees
    FROM Employees;
IF v_TotalEmployees = 0 THEN
    INSERT INTO temp_table (message)
      VALUES ('There are no employees
      registered');
  ELSIF v_TotalEmployees < 5 THEN
    INSERT INTO temp_table (message)
      VALUES ('There are only a few
      employees registered');
```

```
ELSIF v_TotalEmployees < 10 THEN
    INSERT INTO temp_table
      (message)
      VALUES ('There are a little more
      employees registered');
  ELSE
    INSERT INTO temp_table
      (message)
      VALUES ('There are many
      employees registered');
  END IF;
END;
/
```

CASE Expression

```
CASE selector
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
/
```

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DECLARE
   grade CHAR(1) := UPPER('&grade');
   appraisal VARCHAR2(20);
BEGIN
   appraisal  :=
      CASE grade
         WHEN 'A' THEN 'Excellent'
         WHEN 'B' THEN 'Very Good'
         WHEN 'C' THEN 'Good'
         ELSE 'No such grade'
      END;
DBMS_OUTPUT.PUT_LINE ('Grade: '|| grade || '
                        Appraisal ' || appraisal);
END;
/
```

PL/SQL Looping

- Looping Constructs
  - A loop allows execution of a set of statements repeatedly
  - Types of loops
    - Simple loop
    - Numeric For loop
    - While loop

LOOPs

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

Simple Loop

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Numeric For Loop

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

While Loop

Simple Loop VS Numeric FOR Loop

Simple Loop

```
DECLARE
  v_LoopCounter INTEGER := 1;
BEGIN
  LOOP
    INSERT INTO temp_table (num_col)
      VALUES (v_LoopCounter);
    v_LoopCounter := v_LoopCounter + 1;
    EXIT WHEN v_LoopCounter > 50;
  END LOOP;
END;
/
```

Numeric FOR Loop

```
BEGIN
  FOR v_LoopCounter IN 1..50 LOOP
    INSERT INTO temp_table
    (num_col)
      VALUES (v_LoopCounter);
  END LOOP;
END;
/
```

Simple LOOP

```
declare
     i number(2) := 1;
begin

     loop
         dbms_output.put_line(i);
         i := i + 1;
         exit when i > 10;
     end loop;

end;
```

WHILE

```
declare
     i number(2) := 1;
begin

     while  i <= 10
     loop
         dbms_output.put_line(i);
         i := i + 1;
     end loop;

end;
```

FOR LOOP

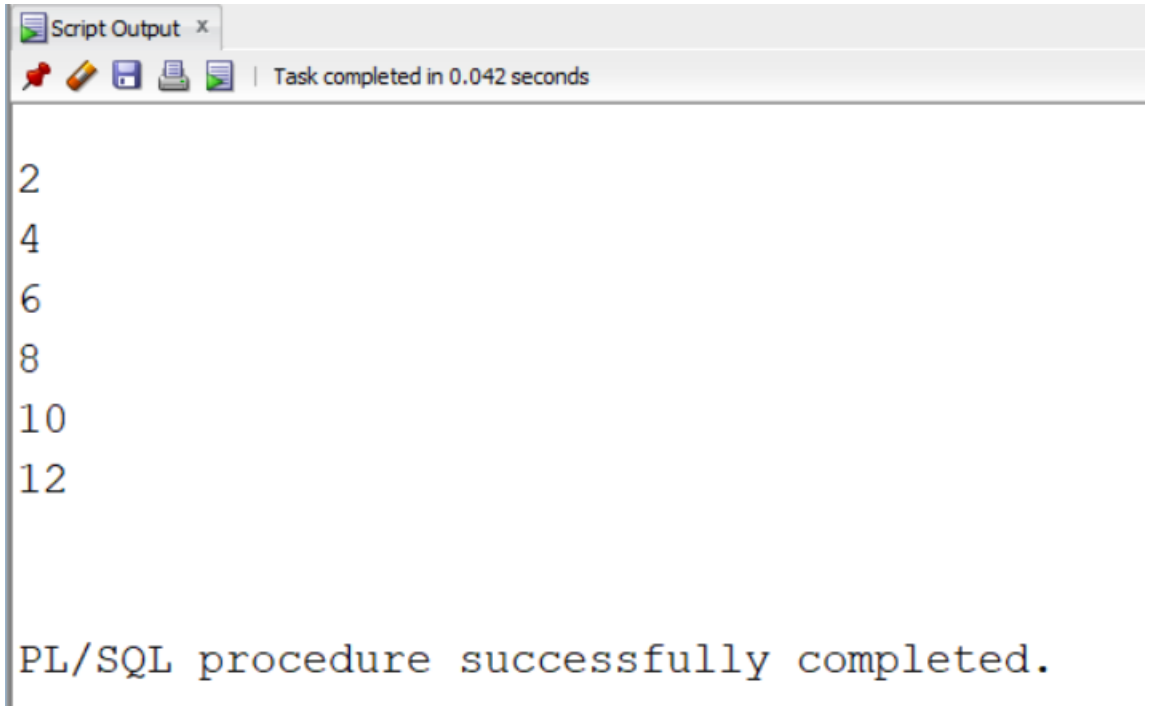
```
begin

    for i in 1..10
    loop
         dbms_output.put_line(i);
    end loop;

end;
```

```
begin

    for i in REVERSE 1..10
    loop
         dbms_output.put_line(i);
    end loop;

end;
```

Write a PL/SQL code to print multiples of 2 (e.g 2,4,6…12) using for loop.

Activity

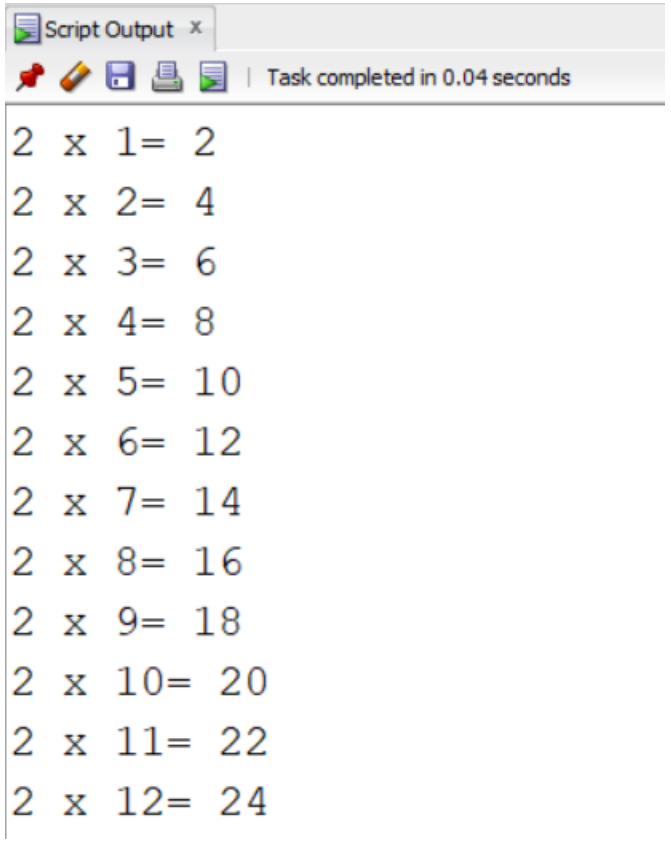
```sql
-- Q1: Print multiples of 2
Set serveroutput ON;
DECLARE
    mult2 number :=0;
BEGIN
    FOR i IN 1..6 LOOP
        mult2:= i*2;
        DBMS_OUTPUT.PUT_LINE(mult2);
    END LOOP;
END;
```



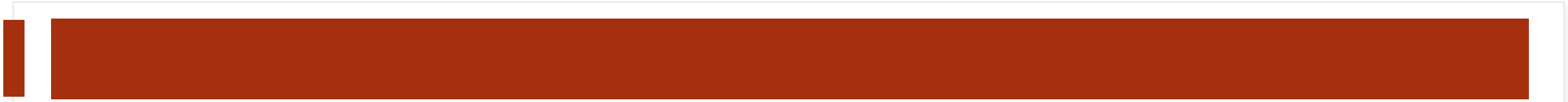
Write a PL/SQL code to print table of 2
(e.g
2x1=2,
2x2=4,
till 2x12=24)
using for loop.

Activity

```
-- Table of 2
Set serveroutput ON;
DECLARE
    mult2 number :=0;
BEGIN
    FOR i IN 1..12 LOOP
        mult2:= i*2;
        DBMS_OUTPUT.PUT_LINE('2 x '||i||'= '||mult2);
    END LOOP;
END;
```



What if we need to print tables for different numbers at different times, do we **repeat** the code and change the multiplier each time?



PL/SQL Procedures & Functions

- Subprograms
  - Special type of ‘block’ statements are Procedures or Functions in PL/SQL
  - can be compiled and stored in the database
    - Hence, called ‘Stored Procedure’
  - can be called from another PL/SQL block

Syntax

CREATE OR REPLACE PROCEDURE procedure_name (*parameter_name* [IN | OUT | IN OUT] *datatype*) **AS**

-- Variable declarations

BEGIN

-- Executable statements

-- Logic for the procedure

EXCEPTION

-- Exception handling (optional)

END procedure_name; *-- This is optional, is written for clarity of end of the procedure*

Create and Invoke the Procedure

```
...
CREATE TABLE dept AS SELECT * FROM departments;
CREATE PROCEDURE add_dept IS
 dept_id  dept.department_id%TYPE;
 dept_name  dept.department_name%TYPE;
BEGIN
 dept_id:=280;
 dept_name:='ST-Curriculum';
 INSERT INTO dept(department_id,department_name)
 VALUES(dept_id,dept_name);
 DBMS_OUTPUT.PUT_LINE(' Inserted '||
  SQL%ROWCOUNT ||' row ');
END;
/
```

```
BEGIN
 add_dept;
END;
/
SELECT department_id, department_name FROM
dept WHERE department_id=280;
```

Inserted 1 row
PL/SQL procedure successfully completed.

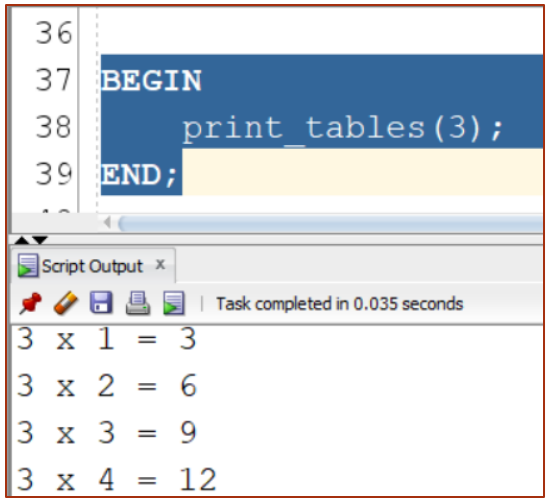
DEPARTMENT_ID	DEPARTMENT_NAME
280	ST-Curriculum

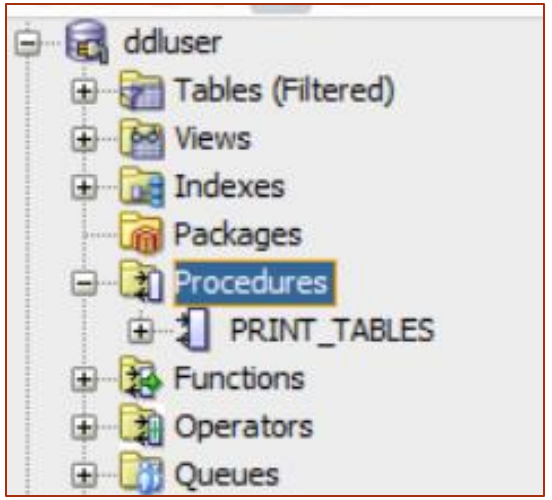
Write a stored procedure named print_tables which takes any number as an input and prints the table.
(hint: use the table of 2 code)

Activity

```
-- Stored Procedure
CREATE OR REPLACE PROCEDURE print_tables (num IN NUMBER) AS
    mult_result NUMBER := 0;
BEGIN
    FOR i IN 1..12 LOOP
        mult_result := i * num;
        DBMS_OUTPUT.PUT_LINE(num || ' x ' || i || ' = ' || mult_result);
    END LOOP;
END;
```

Print_Tables





Cursors

PL/SQL Cursor

- A cursor is a pointer to the private memory space allocated by the Oracle server.
- There are two types of cursors:
  - **Implicit:** Created and managed internally by the Oracle server to process SQL statements
  - **Explicit:** Explicitly declared by the programmer

SQL Cursor Attributes for Implicit Cursors

- Using SQL cursor attributes, you can test the outcome of your SQL statements.

SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement returned at least one row
SQL%NOTFOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement did not return even one row
SQL%ROWCOUNT	An integer value that represents the number of rows affected by the most recent SQL statement

SQL Cursor Attributes for Implicit Cursors

- Delete rows that have the specified employee ID from the employees table. Print the number of rows deleted.

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  empno employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM employees
  WHERE employee_id = empno;
  :rows deleted := (SQL%ROWCOUNT ||
                        ' row deleted.');
END;
/
PRINT rows_deleted
```

SQL and PL/SQL commands

Step #1 Create table students using appropriate datatypes.

(ID (Primary key), first_name, last_name, major)

Step#2

/* Update the Student's major, else create the student record, if not found */

```
Create or replace procedure Update_Student_Record
( p_NewMajor IN VARCHAR2,   p_FirstName IN VARCHAR2,  p_LastName IN VARCHAR2)
AS
BEGIN
  UPDATE students
    SET major = p_NewMajor
    WHERE first_name = p_FirstName
    AND last_name = p_LastName;
 IF SQL%NOTFOUND THEN
    INSERT INTO students (ID, first_name, last_name, major)
      VALUES (student_sequence.NEXTVAL, p_FirstName, p_LastName, p_NewMajor);
  END IF;
END;
/
```

Activity contd.

Consider the table:
PRODUCTS (**prod_id**, prod_name and prod_price). Write a procedure that prints the name and price for product id=2.

If the product with id=2 doesn't exist, output a suitable message.

```
SET serveroutput ON;
DECLARE
    v_prod_name  PRODUCTS.prod_name%TYPE;
    v_prod_price PRODUCTS.prod_price%TYPE;
BEGIN
    SELECT prod_name, prod_price
    INTO v_prod_name, v_prod_price
    FROM PRODUCTS
    WHERE prod_id = 2;

    DBMS_OUTPUT.PUT_LINE('The Product named'||v_prod_name||'is priced at'||v_prod_price);
END;
```

Activity (initial code)

SQL Cursor Attributes for Implicit Cursors

```
CREATE OR REPLACE PROCEDURE print_product_details AS
    v_prod_name  PRODUCTS.prod_name%TYPE;
    v_prod_price PRODUCTS.prod_price%TYPE;
BEGIN
    SELECT prod_name, prod_price
    INTO v_prod_name, v_prod_price
    FROM PRODUCTS
    WHERE prod_id = 2;
    -- Check if any row was found using SQL%NOTFOUND
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('No product found with prod_id = 2.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('The Product named ' || v_prod_name || ' is priced at ' || v_prod_price);
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred.');
END print_product_details;
```

PL/SQL Cursor

- A cursor creates a named context area as a result of executing an associated SQL statement
- Used to process multiple rows retrieved from the Database (aka array)
    - Permits the program to step through the multiple rows displayed by an SQL statement

PL/SQL Cursors

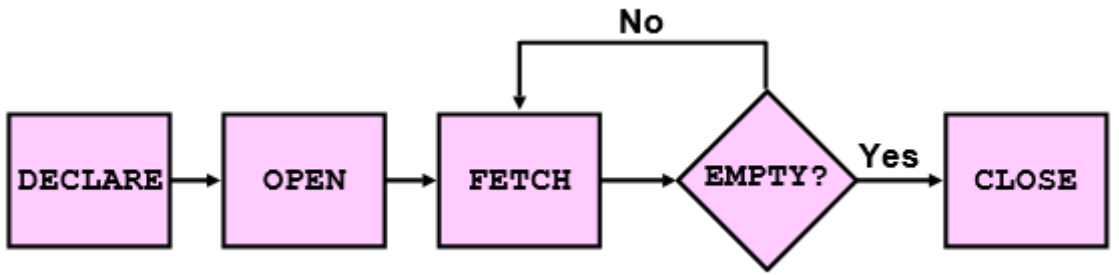
Syntax:

```
CURSOR cursor_name IS
     select_statement;
```

Examples

```
DECLARE
  CURSOR emp_cursor IS
  SELECT employee_id, last_name FROM employees
  WHERE department_id =30;
```

```
DECLARE
  locid NUMBER:= 1700;
  CURSOR dept_cursor IS
  SELECT * FROM departments
  WHERE location_id = locid;
...
```



- Create a named SQL area.
- Identify the active set.
- Load the current row into variables.
- Test for existing rows.
- Return to FETCH if rows are found.
- Release the active set.

Explicit Cursor Attributes

Obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the total number of rows returned so far

Cursor Fetch Loop VS Cursor FOR Loop

```
DECLARE
  v_FirstName VARCHAR2(20);
  v_LastName  VARCHAR2(20);
  CURSOR c_employees IS
    SELECT first_name, last_name
      FROM employees;
BEGIN
  OPEN c_employees;
  LOOP
    FETCH c_employees INTO v_FirstName, v_LastName;
    EXIT WHEN c_employees %NOTFOUND;
    dbms_output.put_line(v_firstname||'  ' ||v_lastName);
  END LOOP;
  CLOSE c_employees;
END;
```

```
SET SERVEROUTPUT ON
DECLARE
   CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
BEGIN
    FOR emp_record IN emp_cursor
     LOOP
      DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
      ||' ' ||emp_record.last_name);
     END LOOP;
END;
/
```

Example: Stored Procedure

```
CREATE OR REPLACE PROCEDURE PrintEmployees(JbRole IN varchar2)
AS
  CURSOR c_employees IS
    SELECT first_name, last_name
      FROM employees
      WHERE job_id  = JbRole;
BEGIN
  FOR v_EmployeeRec IN c_employees LOOP
    DBMS_OUTPUT.PUT_LINE(v_EmployeeRec.first_name || ' ' || v_EmployeeRec.last_name);
  END LOOP;
END;
/
```

Create a Stored Procedure

```
BEGIN
  PrintEmployees('IT_PROG');
END;
/
```

Uses the stored procedure

PL/SQL Error Handling

- Exception handling section permits the user to trap and respond to run-time errors
- Exceptions can be associated with
  - Predefined Oracle errors
  - User-defined errors

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

Predefined Exceptions

- Sample predefined exceptions:
  - NO_DATA_FOUND: Single row SELECT returned no data.
  - TOO_MANY_ROWS: Single-row SELECT returned more than one row.
  - INVALID_CURSOR: Illegal cursor operation occurred.
  - ZERO_DIVIDE: Attempted to divide by zero

```
SET SERVEROUTPUT ON
DECLARE
   lname VARCHAR2(15);
BEGIN
   SELECT last_name INTO lname FROM employees WHERE
   first_name='John';
   DBMS OUTPUT.PUT LINE ('John''s last name is : '
   ||lname);
EXCEPTION
   WHEN TOO_MANY_ROWS THEN
   DBMS_OUTPUT.PUT_LINE (' Your select statement
   retrieved multiple rows. Consider using a
   cursor.');
END;
/
```

```
WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
```

SQL Cursor Attributes for Implicit Cursors


```
CREATE OR REPLACE PROCEDURE print_product_details AS
    v_prod_name  PRODUCTS.prod_name%TYPE;
    v_prod_price PRODUCTS.prod_price%TYPE;
BEGIN
    SELECT prod_name, prod_price
    INTO v_prod_name, v_prod_price
    FROM PRODUCTS
    WHERE prod_id = 2;

    -- Check if any row was found using SQL%NOTFOUND
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('No product found with prod_id = 2.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('The Product named ' || v_prod_name || ' is priced at ' || v_prod_price);
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred.');
END print_product_details;
```

```
CREATE OR REPLACE PROCEDURE print_product_details AS
    v_prod_name  PRODUCTS.prod_name%TYPE;
    v_prod_price PRODUCTS.prod_price%TYPE;
BEGIN
    SELECT prod_name, prod_price
    INTO v_prod_name, v_prod_price
    FROM PRODUCTS
    WHERE prod_id = 2;

    DBMS_OUTPUT.PUT_LINE('The Product named ' || v_prod_name || ' is priced at ' || v_prod_price);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No product found with prod_id = 2.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred.');
END print_product_details;
```

Activity contd. - Exception Block

Example: Exception handler

STEP #1 **Create table with appropriate datatype:** Log_table
Attributes: (Code, Message, info)

STEP #2

```
DECLARE
  v_ErrorCode NUMBER;                    -- Code for the error
  v_ErrorMsg  VARCHAR2(200);             -- Message text for the error
  v_CurrentUser VARCHAR2(8);             -- Current database user
  v_Information VARCHAR2(100);           -- Information about the error
BEGIN
  /* Code which processes some data here */
  NULL;
EXCEPTION
  WHEN OTHERS THEN                       --- Construct similar to Select Case of VB
    v_ErrorCode       := SQLCODE;
    v_ErrorMsg        := SQLERRM;
    v_CurrentUser     := USER;
    v_Information     := 'Error encountered on ' || TO_CHAR(SYSDATE) ||
                                  ' by database user ' || v_CurrentUser;
    INSERT INTO log_table (code, message, info)
                VALUES (v_ErrorCode, v_ErrorMsg, v_Information);
END;
/
```

SQLCODE: Returns the numeric value for the errorcode

SQLERRM: Returns the message associated with the error number

Brainstorm Your Project

CS 341 Database Systems

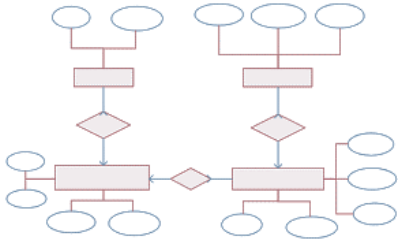
Brainstorming activity

Users/Use Cases

- Specify the key use cases and elaborate on how different types of users will interact with the application.

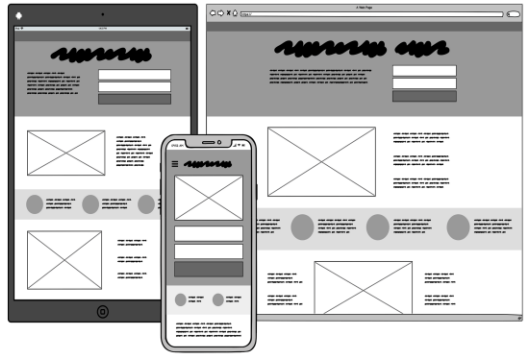
Business Rules

- Identify and state the fundamental business rules governing your chosen scenario.



ER Model

- Identify entities, attributes and relationships



Wireframes

- Sketch the wireframes for your application.