Subprogram

(1)

Subprograms are named PL/SQL blocks that can take parameters and can be invoked. Explain it.

Two Types

① Procedures — used to perform action

② Functions — to compute value

Like unnamed or anonymous PL/SQL Block, Subprgs too have a declarative part, execution part & option exception-handling part.

Subprograms → further two type

LOCAL & STORED TYPE.

Local procedures and fns. are created in declarative section of PL/SQL module, & are local to that module. They can be called anywhere in module's execution section.

Stored Subprograms

Store Subprograms or functions is a named PL/SQL code block that have been compiled and stored in the Oracle's engines system table (database).

* If error occurs during the compilation of procedure or fns, an invalid procedure or fn is created. Oracle displays message that it has been created with compilation errors but does not display error. To see these errors:-

```
Select * from user_errors;
   or
Show errors;
```

Advantages of Subprograms

* Extensibility → new prog modules w/o affecting existing.
* Modularity
* Reusability
* Maintainability
* Abstraction → ∵ only know what they do, but not how they do.
* Dummy Subprograms (Stubs) — It allows defering the definition of procedures & functions until we test and debug the main program.

Procedure	Function
A Subprog that can accept parameters, perform action & return parameters.	A subprg that can accept parameters, compute a value & return that value to a caller.
It may return no value.	Must return only one value.
Can not return value as direct o/p.	— can do.
Use it to perform action	— compute a value.

Procedures.

Local Procedure.

Declare

-- Declaration of Global Variables.

PROCEDURE name

[(arguments {IN, OUT, IN OUT} data type, ---)] {IS | AS}

[local declarations].

BEGIN

PL/SQL Subprogram body;

[EXCEPTION exception handlers] of Procedure

END [name];

BEGIN

-- Executable code

-- Code to call procedure

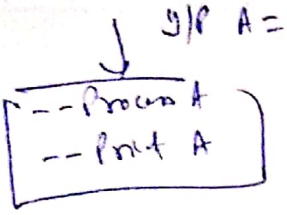
[EXCEPTION exception handlers for main Block;].

END;

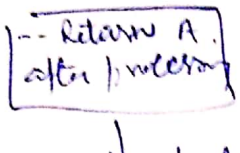
Here;

IN	OUT	IN OUT
- It is default	It must be specified	It must be specified
- Passes value of SP	- Returns values to caller.	- Passes initial values to SP & returns updated values to caller.
- It is a formal parameter acts like constant	- It is a formal parameter acts like an uninitialized variable.	- It is a formal parameter acts like initialized variable.
- It is a formal parameter can not be assigned a value.	- It is a formal Parameter can not be used in an expression & must be assigned a value.	- It is a formal parameter should be assigned a value
- It can be a constant initialized variable, literal or expression.	- It must be a variable	- It must be a variable.

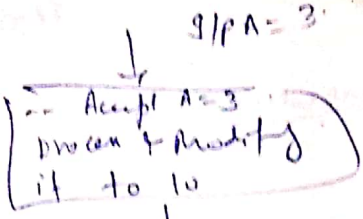
IN mode



OUT mode.



IN OUT mode



WAP to accept two nos & write add, sub, mult div. & return multiple values through arguments.

DECLARE

~~A, B, C, D, E~~ A NUMBER; same as B, C, D, E, F . . .)

PROCEDURE PROCESS (A IN NUMBER, B IN NUMBER, C OUT NUMBER,
D . . . . F OUT NUMBER) IS.

BEGIN

C := A+B; D := A-B; E := A*B; F := A/B;

END;

BEGIN

A := 'First No.';

B := 'Second No';

PROCESS (A, B, C, D, E, F);

DBPL ('Addition' || C);

(Sub D);

(Mult. E);

(Divi || F);

END;

Subprogram

(1) (1)

Consider:- procedure to raise salary, which accepts two parameters (3) empid & amount to be added to salary of employee. It increases the salary & update it to the database

```
DECLARE
    empno Number; amount Number;
    PROCEDURE rs(empid number, increase number) IS
        cur_sal Number(7,2);
        Sal_missing EXCEPTION;
    BEGIN
        SELECT Sal INTO cur_sal FROM emp WHERE empno = empid;
            IF cur_sal IS NULL THEN
                RAISE Sal_missing;
            ELSE
                UPDATE emp SET Sal = Sal + increase WHERE empno = empid;
            END IF;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_NAMS('NO Such Number');
            when Sal_missing THEN
            DBMS_OUTPUT_LINE('Salary IS NULL');

        END rs;
BEGIN
    empno := &empno
    Amount := &amount.
    rs(empno, amount);
END;
```

formal parameter

empno, amount
actual parameter.

Stored Procedure → ~~is a set of~~

To create a stored permanently in oracle DB, we use CREATE PROCEDURE statement, which can be executed interactively from SQL*PLUS.

CREATE OR REPLACE PROCEDURE procedurename
(argument {IN, OUT, INOUT} datatype ...) {IS | AS}
[local declarations];
BEGIN
PL/SQL ~~Begin~~ subprogram body;
[EXCEPTION
exception PL/SQL Block;]
END;

Calling a Procedure - can be called from any PL/SQL program by giving their names followed by parameters as shown

Ex: Create stored procedure that accepts 2 nos & return, add, sub, mul, div.

CREATE OR REPLACE PROCEDURE PROCESS (A IN NUMBER, B in number,
C OUT Number ..... F OUT number) IS
BEGIN
C:=A+B; D:=A-B; E:=A*B; F:=A/B;
END;

~~Call procedure code~~ PL/SQL Code to call procedure Process Created above

DECLARE
A Number; B ---------- F Number;
BEGIN
A:= &FirstNo; B = &Secondnumber;
PROCESS (A, B, C, D, E, F);
DBMS_OUTPUT.PUT_LINE('Addition is ' || C);
D
E
F.
END;

```
e:=&empno;
fire_employee(e);
END;
```

22.6 ACTUAL VERSUS FORMAL PARAMETERS

Subprograms pass information using parameters. There are two types of parameters.

- Formal Parameters
- Actual Parameters

Formal Parameters are declared in the parameter list of procedure or function definition.

For example, the following procedure declares two formal parameters named emp_id and increase:

```
PROCEDURE raise_salary (emp_id NUMBER, increase NUMBER) IS
```

Actual Parameters are referenced in a procedure or function call.

For example, the following procedure call lists two actual parameters named emp_num and amount:

```
raise_salary(emp_num, amount);
```

22.7 FUNCTIONS

A function is a subprogram that computes a value. Functions and procedures are structured alike, except that functions have a RETURN clause.

There are two types of functions:

- Local function
- Stored function

22.7.1 Syntax to define a function local to block

```
DECLARE
   FUNCTION function_name
   [(argument IN data type, ...)] RETURN datatype [IS | AS]
   [local declarations]
   BEGIN
   PL/SQL subprogram body;
   [EXCEPTION
```

5. **Accessories**

In this module the user can buy the various accessories like belts, ties, cufflinks, tie-nibs, watches, sunglasses etc. The accessories play an important part for creating whole outlook for a person. Without them any look would be incomplete.

6. **Wedding Collection**

Wedding is an auspicious day in one's life, so everyone wish to look astonishing on this important day. As clothes play an important part in one's personality. So we have provided a huge range of wedding collections.

7. **Searching**

We will be provided different types of searching for the user to easily select his choice of clothes.

- **By Price range**
  The user will be provided with the facility to select the clothes in the price range.
- **By Brand**
  The user will be provided with the facility to select the clothes of his selected brand.

8. **Shopping bag**

After clicking on the buy button the item will be added to the shopping bag so that the user can buy the apparels later on.

9. **Payment**

The user has the choice to pay the bill by cash on delivery (COD).

```
    exception handlers]
    END [name];
BEGIN
-- PL/SQL code to call function
[EXCEPTION
-- calling program exception handler]
END;
```

Here:

function_name is the function name to identify the function when it is called in the program

return datatype is used to declare the datatype of the return value of the function which can be any valid PL/SQL datatype.

declarations is optional declaration section where local variable, constant or cursor are placed.

> **Note: We cannot impose the NOT NULL constraint on a parameter, and we cannot specify a constraint on the datatype.**

RETURN Statement

The RETURN statement immediately completes the execution of a subprogram and returns control to the caller. Execution then resumes with the statement following the subprogram call. (Do not confuse the RETURN statement with the RETURN clause, which specifies the datatype of the result value in a function specification.)

A subprogram can contain several RETURN statements. Executing any of them completes the subprogram immediately. In procedures, a RETURN statement cannot contain an expression. The statement simply returns control to the caller before the normal end of the procedure is reached. However, in functions, a RETURN statement must contain an expression, which is evaluated when the RETURN statement is executed. The resulting value is assigned to the function identifier, which acts like a variable of the type specified in the RETURN clause. A function must contain at least one RETURN statement. Otherwise, PL/SQL raises the predefined exception PROGRAM_ERROR at run time.

Example 22.7. ***Write a PL/SQL code that calls a function to add two numbers.***

Solution:

```
DECLARE
   A NUMBER;
   B NUMBER;
```

```
        C NUMBER;
        FUNCTION ADDN ( A IN NUMBER, B IN NUMBER) RETURN NUMBER IS
        BEGIN
        C:=A+B;
        RETURN(C);
        END;
    BEGIN
        A:=&FIRSTNUMBER;
        B:=&SECONDNUMBER;
        C:=ADDN(A,B);
        DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);
    END;
```

Example 22.8. *A PL/SQL code, which call a function balance to return the balance of a specified bank account.*

Solution:

```
DECLARE
acc_no NUMBER;
bal NUMBER;
    FUNCTION balance (acct_id NUMBER) RETURN NUMBER IS
    acct_bal NUMBER;
    BEGIN
    SELECT bal INTO acct_bal FROM accts WHERE acct_no = acct_id;
    RETURN acct_bal;
    END balance;
BEGIN
    acc_no:=&acc_no;
    bal:=balance(acc_no);
    DBMS_OUTPUT.PUT_LINE('BALANCE IS'||bal);
END;
```

Example 22.9. *Consider a function which accepts a code and find the rate for the code from item table, quantity from sale table and return the product of rate*quantity. The PL/SQL block calls the function by passing codes from 1 to 5 and adds up the total values returned by the function and displays it.*

Solution:

```
DECLARE
 total NUMBER:=0;
 ctr NUMBER:=1;
 FUNCTION cal_value (code NUMBER) RETRUN NUMBER IS
        it_rate NUMBER;
        it_qty NUMBER;
 BEGIN
        SELECT rate INTO it_rate FROM item1 WHERE item_code=code;
        SELECT qty INTO it_qty FROM sale WHERE item_code=code;
        RETURN (it_rate*it_qty);
 EXCEPTION
 WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE('NO SUCH CODE');
 END;
BEGIN
 WHILE ctr<5 LOOP
 total:=cal_value(ctr)+total;
 ctr:=ctr+1;
 END LOOP;
 DBMS_OUTPUT.PUT_LINE(TOTAL);
END;
```

A function can use both the out parameter and return statement to return the multiple values to the calling subprogram which is illustrated in the following PL/SQL block.

Example 22.10. ***Write a PL/SQL code that to show that a function can use both the out parameter and return statement to return the multiple values to the calling subprogram.***

Solution:

```
DECLARE
    A NUMBER;
    B NUMBER;
    C NUMBER;
    D NUMBER;
```

```
FUNCTION ADDN ( A IN NUMBER, B IN NUMBER, C OUT NUMBER) RETURN NUMBER IS
    BEGIN
    C:=A+B;
    D:=A-B;
    RETURN(D);
    END;
BEGIN
    A:=&FIRSTNUMBER;
    B:=&SECONDNUMBER;
    D:=ADDN(A,B,C);
    DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);
    DBMS_OUTPUT.PUT_LINE('SUBTARCTION IS'||D);
END;
```

22.7.2 Syntax for Stored Function

```
CREATE OR REPLACE FUNCTION function_name
    (argument IN DATATYPE,...)
    RETURN datatype {IS | AS}
    [local declarations]
BEGIN
    PL/SQL subprogram body;
[EXCEPTION
    exception PL/SQL block;]
END;
```

Example 22.11. ***Create a Stored function that accepts two numbers and return addition of passed values.***

Solution:

```
CREATE OR REPLACE FUNCTION ADDN( A IN NUMBER, B IN NUMBER)
    RETURN NUMBER IS
    C NUMBER;
BEGIN
    C:=A+B;
    RETURN(C);
END;
```

Example 22.12. *A PL/SQL code to call the function ADDN created in example 22.10.*

Solution:

```
DECLARE
    A NUMBER;
    B NUMBER;
    C NUMBER;
BEGIN
    A:=&FIRSTNUMBER;
    B:=&SECONDNUMBER;
    C:=ADDN(A,B);
    DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);
END;
```

Example 22.13. *A Stored function that accepts department number and return total salary of that department.*

Solution:

```
CREATE OR REPLCAE FUNCTION SALARY (dept NUMBER) RETURN
NUMBER IS
    s NUMBER;
BEGIN
    Select sum(sal) into s from emp where dep' =dept;
    RETURN(s);
END;
```

Example 22.14. *A PL/SQL code to call example 22.13.*

Solution:

```
DECLARE
    D NUMBER;
BEGIN
    D:=&DEPTNO;
    SAL:=SALARY(D);
    DBMS_OUTPUT.PUT_LINE(
END;
```

ALTER PACKAGE pa

22.8. DICTIONARY INFORMATION ABOUT STORED SUBPROGRAMS

The information about all the stored procedure and functions can be obtained by using the USER_OBJECTS internal table, which has the following structure:

Name	Null?	Type
OBJECT_NAME		VARCHAR2(128)
SUBOBJECT_NAME		VARCHAR2(30)
OBJECT_ID		NUMBER
DATA_OBJECT_ID		NUMBER
OBJECT_TYPE		VARCHAR2(19)
CREATED		DATE
LAST_DDL_TIME		DATE
TIMESTAMP		VARCHAR2(19)
STATUS		VARCHAR2(7)
TEMPORARY		VARCHAR2(1)
GENERATED		VARCHAR2(1)
SECONDARY		VARCHAR2(1)

In order to see the details of all the stored functions we can issue the following query on USER_OBJECTS table:

```
SQL>SELECT OBJECT_NAME, OBJECT_TYPE, STATUS FROM USER_OBJECTS WHERE
OBJECT_TYPE='PROCEDURE';
```

The output is:

OBJECT_NAME	OBJECT_TYPE	STATUS
CHECK_UPDATE	PROCEDURE	VALID

In order to see the source code of the stored subprogram we can use USER_SOURCE, which contain the following information:

Name	Null?	Type
NAME	NOT NULL	VARCHAR2(30)
TYPE		VARCHAR2(12)
LINE	NOT NULL	NUMBER
TEXT		VARCHAR2(4000)

22.9 PACKAGE

Packages are groups of procedures, functions, variables, constants, cursors, exceptions and SQL statements grouped together into a single unit. The tool used to create a package is SQL*Plus. A package once written and debugged is compiled and stored in Oracle's system tables held in an Oracle Database. All users who have executed permissions on the Oracle Database can use the package. Unlike the stored programs the package itself cannot be called.

Components of an Oracle Package:

A package has usually two components:

- Package specification
- Package body

Package Specifications : A package's specification declares the memory variables, constants, exceptions, cursors and subprograms that are available for use.

Syntax:

```
CREATE OR REPLACE PACKAGE package_name AS
FUNCTION function_name (list of arguments) RETURN data type;
PROCEDURE procedure_name (list of arguments);
End package_name;
```

Package Body : The body of a package contains the definition of public objects that are declared in the specification. It implements the specifications. Unlike package specification, the package body can contain subprogram bodies.

Syntax:

```
CREATE OR REPLACE PACKAGE BODY package_name AS
FUNCTION function_name (list of arguments) RETURN data type {IS|AS}
[local declaration of function]
BEGIN
-- code of function
[EXCEPTION
-- exception handler of function]
END function_name;
PROCEDURE procedure_name (list of arguments) {IS|AS}
[local declaration of procedure]
```

```
BEGIN
-- code of procedure
[EXCEPTION
-- exception handler of procedure]
END procedure_name;
[...similarly other functions and procedures]
END package_name;
```

Steps to Create Packages

The first step to create a package is to create its specification. The specification declares the objects that are contained in the body of the package. A package is created using Oracle's SQL *Plus interactive tool. After the specification is created, the body of the package to be created. The body of the package is a collection of detailed definitions of the objects that were declared in the specification.

Referencing Package Contents

The package contents can be referenced from database triggers, stored subprograms or PL/SQL blocks. To reference a package's subprogram and objects, we must use dot notation.

Syntax for Dot Notation:

Package_name.type_name

Package_name.object_name

Package_name.subprogram_name

In this syntax

Package_name is the name of the declared package.

Type_name is the name of the type that is user defined, such as record.

Object_name is the name of the constant or variable that is declared by the user.

Sub_program is the name of the procedure or function contained in the package body.

For example to reference a procedure addn in package bank we can use.

Bank.addn(list_of_arguments);

22.10 ADVANTAGES OF PACKAGES

Packages offer the following advantages:

Modularity

Packages encapsulate related types, objects, procedures, functions, variables, constants, cursors and exceptions together in a named PL/SQL module.

Easier Application Design

A package specification can be coded and complied without its body which increases the designer productivity.

Information Hiding

By hiding implementation details in package body, better security is provided to th system. It also allows granting of privileges efficiently.

Added Functionality

A package's public variables and cursors persist for the duration of the session. Therefo all cursors and procedures that execute in this environment can share them.

Better Performance

Packages improve performance by loading multiple objects into memory at one Therefore, subsequent calls to related subprograms in the package require no Input/Outp

Example 22.15: ***Create a package specification for the Insert_Oper, Retrieve Update_Oper and Delete_Oper whose description is given below:***

Insert_Oper: Procedure to insert record in emp table, values are passed to procedu as input arguments.

Retrieve: Procedure to retrieve record on the basis of empno. Here empno is passed procedure as input argument and returns name and salary as output argument.

Update_Oper: Function to return number of record updated, we supply value of deptn and increased amount and it will add the amount to salary for all the employees in th department.

Delete_Oper: Function to delete the record of employee on the basis of empno passe to the function and it return 'Y' and 'N' according the result.

Solution:

```
CREATE OR REPLACE PACKAGE OPERATION AS
        PROCEDURE INSERT_OPER(ENO NUMBER, NAME VARCHAR, JOB
        VARCHAR,SAL NUMBER, DNO NUMBER);
        PROCEDURE RETRIEVE(ENO IN NUMBER, NAME OUT VARCHAR,
        SAL OUT NUMBER);
        FUNCTION UPDATE_OPER(DNO NUMBER,AMOUNT NUMBER)
        RETURN NUMBER;
        FUNCTION DELETE_OPER(ENO NUMBER) RETURN CHAR;
END;
```

Example 22.16: *Create a package body for the Insert_Oper, Retrieve, Update_Oper and Delete_Oper as described above.*

Solution:

```
CREATE OR REPLACE PACKAGE BODY OPERATION IS
 PROCEDURE INSERT_OPER(ENO NUMBER, NAME VARCHAR, JOB
 VARCHAR,SAL NUMBER, DNO NUMBER) AS
 BEGIN
        INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DEPTNO) VALUES(ENO,
 NAME, JOB,SAL, DNO);
 END INSERT_OPER;
 PROCEDURE RETRIEVE(ENO IN NUMBER; NAME OUT VARCHAR, SAL
 OUT NUMBER) IS
 BEGIN
 SELECT ENAME, SAL INTO NAME,SAL FROM EMP WHERE EMPNO=ENO;
 END RETRIEVE;
 FUNCTION UPDATE_OPER(DNO NUMBER, AMOUNT NUMBER) RETURN
 NUMBER IS
       N NUMBER;
   BEGIN
       UPDATE EMP SET SAL=SAL+AMOUNT WHERE DEPTNO=DNO;
       N:=SQL%ROWCOUNT;
       RETURN(N);
 END UPDATE_OPER;
 FUNCTION DELETE_OPER(ENO NUMBER) RETURN CHAR IS
 BEGIN
      DELETE EMP WHERE EMPNO=ENO;
      IF SQL%FOUND THEN
           RETURN('Y');
      ELSE
           RETURN('N');
      END IF;
END DELETE_OPER;
END OPERATION;
```

Example 22.17: *A PL/SQL code to call Insert_Oper procedure to insert a record in emp table.*

Solution:

```
DECLARE
        ENO NUMBER;
        NAME VARCHAR(20);
        JOB VARCHAR(20);
        SAL NUMBER;
        DNO NUMBER;
BEGIN
        ENO:=&EMPLOYEE_NUMBER;
        NAME:=&EMPLOYEE_NAME;
        JOB:=&EMPLOYEE_JOB;
        SAL:=&EMPLOYEE_SALARY;
        DNO:=&EMPLOYEE_DEPTNO;
        OPERATION.INSERT_OPER(ENO, NAME, JOB, SAL, DNO);
END;
```

Example 22.18 : *A PL/SQL code to call retrieve procedure to retrieve a record in emp table.*

Solution:

```
DECLARE
        EMPNO NUMBER;
        NAME VARCHAR(20);
        SAL NUMBER;
BEGIN
        EMPNO:=&EMPNO_TO_SEARCH;
        OPERATION.RETRIEVE(ENO, NAME, SAL);
        DBMS_OUTPUT.PUT_LINE(ENO||NAME||SAL);
END;
```

Example 22.19: *A PL/SQL code to call update_oper function to get number of record updated.*

Solution:

```
DECLARE
        DEPTNO NUMBER;
        AMOUNT NUMBER;
        REC_NO NUMBER;
BEGIN
        DEPTNO:=&DEPTNO_TO_UPDATE;
        AMOUNT:=&AMOUNT_TO_INCREASE;
        REC_NO:=OPERATION.UPDATE_OPER(DEPTNO,AMOUNT)
        DBMS_OUTPUT.PUT_LINE(REC_NO||'RECORD UPDATED');
END;
```

Example 22.20: *A PL/SQL code to call delete_oper function to get 'Y' or 'N' according to success of delete operation.*

Solution:

```
DECLARE
        EMPNO_TO_DELETE NUMBER;
        RESULT CHAR;
BEGIN
        EMPNO_TO_DELETE:=&EMPNO_TO_DELETE;
        RESULT:=OPERATION.DELETE_OPER(EMPNO_TO_DELETE);
        DBMS_OUTPUT.PUT_LINE('RECORD DELETED: '||RESULT);
END;
```

Alterations to an Existing Package

To recompile a package, use the ALTER PACKAGE command with the compile keyword. It eliminates the need for any implicit run time recompilation and prevents any associated runtime compilation errors and performance overhead. It is common to explicitly compile a package after modifications to the package. Recompiling a package recompiles all objects defined within the package. Recompiling does not change the definition of the package or any of its objects.

Syntax:

```
ALTER PACKAGE package_name COMPILE PACKAGE;
```

To recompile just the body of a package use the following syntax:

Syntax:

```
ALTER PACKAGE package_name COMPILE BODY;
```

Θ (Theta Join) → Condition on common-domain compatible column in both tables. (Θ — any comparison operator).

Employee-Product.

Name	Product
R	Pen
R	Pen
R	Pencil
S	Pen

Product-Customer.

C.Prod.	Cust.
Pen	K
Pen	Su
Pencil	Su

Emp-Prod-cust.

Name	Prod	C.Prod	Cust.
R	Pen	Pen	K
R	Pen	Pen	Su
R	Pen	Pencil	Su
S	Pen	Pen	K
S	Pen	Pen	Su

$R \bowtie_F S$ (F is of form $R.a_i \; \theta \; S.b_i$ where θ (<, ≤, >, ≥, =, ≠).

* If both Tables have same Common column then one of the Common column has to be Renamed in resultant table to Preserve the uniqueness of the names in its header part.

Above Example is Equi Join where Θ (restriction) is that the product attribute values in Emp-Prod should be equal to the product attribute values in product-Cust. It can be written as:-

$(\Pi_{Name, Product}(Emp\_Product)) \bowtie_{Product = C.Product} (\Pi_{C\_Prod, Cust}(Prod.Cust))$

---

* If Θ restriction of equal to was not there then Θ join would be Cartesian product operation of two tables followed by a restriction operation on the resultant table ⋈

Natural Join :- Variation of equi-join where this dupl... of Column values is eliminated by taking a proje... of a table which includes only one of the duplicated Columns.

Emp_Prod_Cost

Name	Product	Cost
R	Pen	K
R	Pen	Rs
J	Pencil	Rs
B	Pen	K
S	Pen	Rs.

∴ We say Join is composite operation of Cartesian product then Selection (restriction) and Projection operator.

Above all examples were inner join. The fact that S makes rubber is not recorded in any of the resultant tables from the joins, because the joining values must exist in both tables.

* If it requires that the value existing in any one table must also appear in the resultant table (O/P) then the Selection is Outer join.

outer Join of above tables would be.

Emp name	Product name	Customer
R	Pen	R
R	Pen	Su
S	Pen.	L
S	Pen	Su.
R	Pencil.	Su.
Sp.	Rubber	— .

Above case is left outer Join.

Semi Join ▷ : Defines a relation that contains the tuples of R that participate in the join of R with S

It is equivalent to ~~Selection~~ Projection of Join.

$$R \triangleright_F S = \Pi_A (R \bowtie_F S).$$

Division operator divides a dividend relation A of degree m+n by a divisor relation B of degree n, and produces a resultant relation of degree m. If

Cursors

Cursor → Work area used by oracle Engine to execute SQL statements & store information.

- PL/SQL allows to name these work areas and to access this stored info.

- The data stored in Cursor is known as Active Data Set.
- All the rows returned by query are stored in the Cursor at the Server and will be as displayed at the client end.

Types:

1) Implicit Cursor (2) Explicit Cursor.

↳ implicitly declared for all SQL statements
- opened, Managed by Oracle Engine
- their operations Maintained automatically.

Explicit

↳ User defined.
- declared explicitly in Declaration section.
- Manipulated through specific statements within executable block.
- needs to be opened before reading/processing, afterward closed.
- Cursor marks the ~~active~~ current position in an active set.

Cursor Attributes (Both Implicit & Explicit).

% Is open. → true if Cr open else false.

% found → true if record successfully fetched from open Cr.

% NOT found → " " " was not " " " "

% Row Count → Returns no. of records processed from cursor.

Implicit Cursor Handling.

Implicit Cursors named "SQL" are declared implicitly for all SQL statements.

* Its attrib. are used to access info. about the status of last insert, update, delete, or single-row select statement.

→ Precede attrib name with "SQL" (implicit cr. name).

* Implicit cr. attrib are used in procedural statements but not in SQL statements.

SQL % Isopen → always false ∵ oracle automatically closes it after executing SQL statement.

SQL % found → True if most recently executed DML statement was successful.

SQL % Not found → ——— was not successful.

SQL % RowCount → Returns no. of rows affected by an INS, UP, Delete or single row select statements.

```
Delete EMP where empno = & empno
If SQL%NOTFound THEN
    DBMS_OPL ('Record Not Deleted');
Else
    DOPL ('Record deleted')
END If ;
END ;
```

eg 2 Delete ——————————

```
If SQL%FOUND THEN
    DOPL ('Record deleted');
Else
    DOPL ('Record not deleted');
END If ;
END ;
```

eg 3

```
Declare
    N Number
Begin
    Delete emp where deptno = & deptno ;
    N := SQL%ROWCOUNT ;
    DOPL ('Total no. of records deleted' || N);
END ;
```

Explicit Cursor Handling.

Steps

1. Declare a cursor
2. Open it
3. Fetch data from Cursor one row at a time into memory variable
4. Process data held in memory variable as required using a loop.
5. Exit from loop
6. Close the cursor.

Declaration

CURSOR <Cursor-name> IS <Select_statement>;

* <Select_statement> includes most of the usual clauses, but INTO clause is not allowed.

Opening: open <cursor_name>;

used within executable part., also establishes an active set of rows. When Cur open, points to its first row in active data set.

Fetching of individual Row

Fetch <Cname> INTO <Var_names>;

For each column value returned by query associated with the cursor, there must be corresponding variable in the INTO list, with matched data type.

With each fetch Cur move its ptr to next row in Active set, hence each fetch gives diff row.

Explicit Cursor Attributes

Each cursor that the user defines has four attributes. These are:

Attributes	Description
%ISOPEN	Evaluates to TRUE, if the cursor is opened, otherwise evaluates to FALSE.
%FOUND	Evaluates to TRUE when last fetch succeeded.
%NOTFOUND	Evaluates to TRUE, if the last fetch failed, i.e. no more rows are left.
%ROWCOUNT	Returns the number of rows fetched

There are four attributes of explicit cursor for obtaining status information about a cursor. When appended to the cursor name these attributes let the user access useful information about the execution of a Multi row query.

Example21.4: *Consider a PL/SQL code to display the empno, ename, job, of employees of department number 10.*

Solution:

```
DECLARE
 CURSOR C1 IS SELECT EMPNO, ENAME, JOB FROM EMP WHERE
 DEPTNO=10;
 REC C1%ROWTYPE;  /* rec is a row type variable for cursor c1 record,
                                containing empno, ename and job*/
BEGIN
 OPEN C1;
 LOOP
 FETCH C1 INTO REC;
 EXIT WHEN C1%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.EMPNO);
 DBMS_OUTPUT.PUT_LINE('ENAME '||REC.ENAME);
 DBMS_OUTPUT.PUT_LINE('JOB '||REC.JOB);
 END LOOP
 CLOSE C1;
END;
```

Example21.5: *Consider a PL/SQL code to display the employee number and name of top 5 highest paid employees.*

Solution:

```
DECLARE
        EMPNAME EMP.ENAME%TYPE;
        EMPSAL EMP.SAL%TYPE;
        CURSOR TEMP1 IS SELECT ENAME, SAL FROM EMP ORDER BY
 SAL DESC;
BEGIN
        OPEN TEMP1;
        LOOP
        FETCH TEMP1 INTO EMPNAME, EMPSAL;
        EXIT WHEN TEMP1%ROWCOUNT>5 OR TEMP1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(EMPNAME || EMPSAL);
        END LOOP;
        CLOSE TEMP1;
 END;
```

Example21.6: Consider a PL/SQL code to increase the salary of employee according to following rule:

Salary of department number 10 employees increased by 1000.

Salary of deportment number 20 employees increased by 500.

Salary of department number 30 employees increased by 800.

Store the employee number, old salary and new salary in a table temp having three columns empno, old and new.

Solution:

```
DECLARE
        OLDSAL NUMBER;
        NEWSAL NUMBER;
        CURSOR C1 IS SELECT * FROM EMP;
        REC C1%ROWTYPE;
 BEGIN
        OPEN C1;
        LOOP
            FETCH C1 INTO REC;
```