**STRUCTURED QUERY LANGUAGE**

Trainer : Thamarai Selvam Date : 24-JUN-2015

Time : 11.15am –12.45pm

**INTRODUCTION:**

* Data 🡪 Collection of raw facts.
* Information 🡪 Processed data.
* Database 🡪 collection of related data.
* DBMS 🡪 Software that facilitates storage ,manipulation,& retrieval of data.

**TYPES OF DBMS :**

1. RDBMS [E.F.codd]
2. ORDBMS(object RDBMS)
3. OORDBMS (Object Oriented RDMS)🡪(oracle)

**CATEGORIES OF SQL :**

1. Data definition language[DDL]
2. Data manipulation language[DML]
3. Data query language
4. Transaction data control language[TCL/ DCL]

**DATA DEFINITION LANGUAGE [**DDL**]:**

* CREATE
* ALTER
* DROP
* RENAME
* TRUNCATE

**CREATE :**

**Syntax🡪**

CREATE table <table name> (

)

**EXAMPLES :**

* CREATE table emp

(

name varchar2(20)

age number(2)

salary number(10,2)

DOJ date

);

Alter table emp 🡺 ADD MODE

**Add**(

DOB date,

City varchar (30)

);

Alter table emp 🡺 MODIFY MODE

**Modify** (name varchar2(40);

)

* Drop table emp;
* Rename <old\_table\_name> to <new\_table\_name> ;

Rename emp to employee;

* Truncate emp;

**DATA INTEGRITY CONSTRAINTS :**

* IT refers to the correctness of data.

**Types of IC :**

1. Entity IC 🡪 PRIMARY & UNIQUE KEY.
2. Domain IC 🡪 CHECK & NOT NULL
3. Referential IC 🡪 FOREIGN KEY

* Primary attribute 🡪 Distinct & doesn’t allows NULL
* unique attribute 🡪 Distinct & allow NULL

**DATA MANIPULATION LANGUAGE :**

* INSERT 🡪to insert a data into a table
  + - Full row insert 🡪 values to be supplied for all columns.
    - Partial row insert 🡪 values to be supplied for the required columns.
* UPDATE
* DELETE
* SELECT 🡪 used to retrieve columns.
* \* 🡪 used to retrieve all the columns.

**EXAMPLE :**

Select column1,column2,column3,…….

From<table-name1>,<table-name2>…….;

Select eno, ename, deptno

From emp,dept;

Select \*

From emp;

* We should qualify the column name with table name,
* SELECT statement is followed by column name.
* FROM statement is followed by table name.
* SELECT comes with 🡪
  + - * 1. Where clause
        2. Order by
        3. Group by
        4. having
* WHERE CLAUSE 🡪 used to **filter rows**
  + - * 1. OR
        2. AND
        3. LIKE
        4. BETWEEN…..AND [inclusive of boundary condition]
        5. IN
        6. IS
        7. NOT
        8. NULL
* Pattern Matching 🡪 regular expression[used to describe a set of things]
* Regular matching symbols🡪
  + - 1. % 🡪 zero or more occurences of a character.
      2. \_ 🡪 single occurence of a character.

**EXAMPLE :**

1. Select ename

From emp

Where ename like ‘s%n’

1. Select \* ename

From emp

Where phone is not null

1. Select salary , salary+salary\*0.2 salary \_increment

From emp;

1. Select distinct designation [to remove duplicates]

From emp;

1. Select \* [arrange the output in specific order]

From emp;

Order by eno,ename;

* ‘\_\_\_\_’ 🡪 display names which has four characters.

Trainer : Thamarai Selvam Date : 29-JUN-2015

Time : 3.12pm –04.10pm

* Group by 🡪 used to summarize the rows & it is always followed by column name
* Having clause 🡪used to filter groups.[must be followed by **group by clause**].
* Where clause 🡪 used to filter rows.
* In 🡪 used as a set member operator.

**AGGREGATE FUNCTION** 🡪 a group function returns a single result based on many rows & columns

* It is valid only when used with SELECT.
  + - AVG
    - COUNT
    - MAX
    - MIN
    - STDDEV
    - SUM
    - VARIANCE

**EXAMPLE :**

* Select count (phone) [excludes NULL value]

From emp;

* Select count (\*) [includes NULL value]

From emp;

* Select eno, sum(sal)

From emp;

**SCALAR /NUMBER FUNCTIONS :**

1. ABS(n)
2. CEIL(n)
3. FLOOR(n)
4. MOD(m,n)
5. POWER(m,n)
6. ROUND(n,[m])
7. TRUNC(n,[m])
8. SIGN(n)
9. SQRT(n)
10. GREATEST(N1,N2,N3,N4……)
11. LEAST(N1,N2,N3,N4……)
12. NVL

**EXAMPLE** :

* Select abs (-70)

From dual; 70

* Select \*

From dual;

* Select ceil (71.3)

From dual; 72

* Select floor (71.3)

From dual; 71

* Select round (71.3)

From dual; 71

**CHARACTER FUNCTION :**

1. ||
2. INITCAP (char)
3. LENGTH (char)
4. LOWER (char)
5. UPPER (char)
6. LTRIM/RTRIM (char)
7. SUBSTR (string, starting position, length)
8. RPAD/LPAD (string1, length, string2)
9. SOUNDEX (string)

**EXAMPLE :**

* + Select ‘Good’ | | ‘Noon’

From dual ; Good Noon

* + Select initcap(‘sriram’)

From dual ; Sriram

* + Select length(‘Manish’)

From dual ; 6

* + Select ltrim(‘ Manish’)

From dual ; Manish

* + Select rtrim(‘Manish ’)

From dual ; Manish

* + Select substr (‘Manish’, 2,5)

From dual ; anis

* + Select rpad(‘Manish’, 8,0)

From dual ; Manish00

* + Select lpad(‘Manish’, 7,’!’)

From dual ; !Manish

**DATE FUNCTIONS :**

1. SYSDATE
2. ADD\_MONTHS (d,n)
3. LAST\_DAY (d)
4. MONTHS\_BETWEEN (d1,d2)
5. NEXT\_DAY (d, string)

**CONVERSION FUNCTIONS :**

1. TO\_CHAR
2. TO\_DATE
3. TO\_NUMBER

**EXAMPLE :**

* + - * Select to\_char(1234)

From dual;

* + - * Select to\_number(‘1234’)

From dual; 4

* + - * Select to\_char(‘abc’)

From dual; error

**DATA CONTROL COMMANDS :**

* Used to control the access to the database and the data
  + - * GRANT
      * REVOKE

**PREVILAGES :**

1. SELECT
2. INSERT
3. UPDATE
4. DELETE

**EXAMPLE :**

* + Grant select, delete on emp , dept to user1, user2;
  + Grant all on emp, dept tp user1,user2;

**GRANT/ REVOKE**

* In a multi User environment Oracle allows the users to share their objects with others.
* GRANT : To give privileges on one’s object to other users
* REVOKE : To take away the given privileges
* PRIVILEGES – Select, Insert, Update, Delete

**SYNTAX:**

* + Grant All on <table-name> to <user1>, <user2>,…
  + Revoke All on <table-name> from <user1>

**TRANSACTION CONTROL COMMANDS :**

* COMMIT – Commits all DML transactions into the database[**permanent**]
* ROLLBACK – Cancels all DML transactions performed.
  + - Either from the beginning of a session or
    - after a previous commit.
* SAVEPOINT – Acts as a bookmark.

**EXAMPLE :**

* Sql>……

……….. [commited commands cannot be rollback]

Sql>rollback;

………

Sql>commit;

Sql>savepoint t1;

………

Sql>savepoint t2;

………

Sql>rollback;

Trainer : Thamarai Selvam Date : 29-JUN-2015

Time : 4.40pm –06.10pm

**JOINS** :

* JOINS 🡪 required to execute multiple queries.
* Inner-most query will be executed first.
* A join is a query which combines rows from two or more tables.
* Are used to retrieve data from multiple tables.
* The query’s select list can contain columns from any of these tables.
* If the tables joined, have column names in common, then the column names should be qualified with the table name or the table alias.

Types of Joins

* 1. Equi Join
  2. Non Equi Join
  3. Self Join
  4. Outer Join

**EXAMPLE :**

* Select cust\_name

From customer\_master\_table

Where representative

( select emp\_code

From emp\_master\_table

Where design =’representative’

And branch\_code =

( select branch\_code

From branch\_master

Where city =’coimbatore’) );

1. Simple sub-query
2. Correlated sub-query

|  |
| --- |
| 1. **Display the managers of Coimbatore branch :**  * Select emp\_code , ename   From emp\_master\_table  Where designation =’manager’ and  Branch\_code =  ( select branch\_code  From branch\_master  Where city=’coimbatore’); |

|  |
| --- |
| 1. **Display the managers of Coimbatore and Madurai branch :**  * Select emp\_code , ename   From emp\_master\_table  Where designation =’manager’ and  Branch\_code **in**  ( select branch\_code  From branch\_master  Where city=’coimbatore’ or city =’madurai ’); |

|  |
| --- |
| 1. **Display the employees reporting to madras branch manager :**  * Select emp\_name , ename   From emp\_master\_table  Where boss in  ( select emp\_code  From emp\_master\_table  Where designation =’manager’  And branch\_code in  ( select branch\_code  From branch\_master  Where city =’madras’) ) ; |

**CORRELATED SUBQUERY :**

* An inner query which has a reference to the outer query and gets executed for every row returned by the outer query is called the **correlated sub-query**
* Inner query executed for every row returned by outer query.

**DISADVANTAGES :**

* Proceesing time will be more
* Performance overhead

**EXAMPLE :**

SELECT eno, emp\_name, salary

FROM emp E

WHERE salary > ( SELECT avg(salary)

FROM emp

WHERE deptno= E.deptno)

* The table alias ‘E’ is called the correlation identifier.
* A column from the outer query is referred in the inner query.

Trainer : Thamarai Selvam Date : 30-JUN-2015

Time : 11.46am –12.45pm

* PL /SQL 🡪 Procedural Language Structured query language

**ADVANTAGES :**

* Support for SQL
* Higher productivity
* Better performance
* Portability
* Integration with oracle

Types of PL/SQL BLOCK :

1. Anonymous PL/SQL Block[compiled on every execution]
2. Named PL/SQL Block (stored procedures or database objects)[**pre compiled form]**
   * 1. Procedure
     2. Function
     3. Trigger
     4. Package[database]

**PARTS OF PL /SQL :**

1. Declarative part,
2. Executable part
3. Exception handling part

**SYNTAX :**

Declare

Declarative section

Begin

Executable section

[Exception]

Exception section

End

|  |
| --- |
| EXAMPLE :   * To compute maximum salary of an employee   declare  m number (7,2);  begin  select max(sal) into m  from emp\_master\_table  dbms\_output.put\_line(‘maximum salary=’ | | m);  End;  / \*//forward slash is used to execute anonymous block//\*  Set server output on; |

**ORDER OF EXECUTION :**

1. From
2. Where[condition]
3. group by
4. order by
5. Select

* Cursor 🡪it is an handle to the Temporary memory[working]area

1. Implicit cursor.
2. Explicit cursor.

**PL/SQL CHARACTER SET:**

* A-Z, a-z, 0 – 9, special characters
* PL/SQL contains groups of characters known a lexical units. They are
  + Identifiers (total, a, b)
  + Literals (4.5, 7.8, 90, ‘Hai’, ’12-May-90’)
  + Comments ( /\*, --)
  + Simple symbols (<, > , ;, +, -)
  + Compound symbols ( := , ||, << >>)

|  |
| --- |
| **Example**  declare  v\_eno number(4) := 1000;  v\_ename varchar2(10) := ‘Thomas’;  v\_dept varchar2(10) := ‘Training’;  begin  INSERT into emp  values (v\_ename, v\_eno, v\_dept);  COMMIT;  end; |

|  |
| --- |
| := 🡪Assignment operator  Constant 🡪 to declare a constant value |

* DML 🡪 we should write COMMIT statement
* DDL 🡪 no need for COMMIT statement

**CONTROL STRUCTURES :**

1. Conditional control
2. Iterative Control
3. Sequential control

**CONDITIONAL CONTROL:**

* Syntax:

If (Condition) Then

Statements

Elsif (Condition) Then

Statements

Elsif (Condition) Then

Statements

End if

|  |
| --- |
| **EXAMPLE:**  declare  v\_empno number(4) := 1000;  v\_dept number(2);  v\_increment number(10,2);  begin  SELECT deptno  INTO v\_dept  FROM emp WHERE eno = v\_empno;  if v\_dept = 10 then  v\_increment := 1000;  elsif v\_dept = 20 then  v\_increment := 500;  else v\_increment := 250;  end if;  UPDATE emp SET salary = salary + v\_increment  WHERE eno = v\_empno;  COMMIT;  end; |

**ITERATIVE CONTROL**

1. Simple loop
2. For Loop
3. While loop

**SIMPLE LOOP :**

SYNTAX:

Loop

Statements

Statements

Exit when < condition >

statements

End loop

|  |
| --- |
| **Example:**  declare  i number(2) := 1;  begin  loop  dbms\_output.put\_line('Hello');  i := i + 1;  exit when ( i > 5 );  end loop;  end; |

**WHILE LOOP:**

* Syntax:

While <condition>

Loop

Statements

Statements

statements

End loop

|  |
| --- |
| EXAMPLE :  declare  i number(2) := 1;  begin  while ( i < 5 )  loop  dbms\_output.put\_line('Hello');  i := i + 1;  end loop;  end; |

**FOR LOOP**

* Syntax:

For counter in [Reverse] lb..ub, step

Loop

Statements

Statements

End Loop

|  |
| --- |
| **EXAMPLE :**  for i in 1...5  loop dbms\_output.put\_line(i);  end loop: 1 2 3 4 5  for i in 1...5,2  loop dbms\_output.put\_line(i);  end loop: 1 3 5  for i in reverse 1...5,1  loop dbms\_output.put\_line(i);  end loop: 5 4 3 2 1 |

**SEQUENTIAL CONTROL :**

* Syntax:

…………….

……………….

Goto label;

…………….

…………………

<<label>>

Trainer : Thamarai Selvam Date : 01-JUL-2015

Time : 4.42pm –05.42pm

|  |
| --- |
| Exercise :   * **Write a function to return a minimum salary for a given city**   Create or replace function Min\_sal(c varchar)  Return number  Is  M number (7,2);  Begin  Select min(sal) into m from  Emp\_master\_table  Wher  Branch\_code =  (select branch\_code from  Branch\_master\_table  Where city=’c’ );  Return m;  End;  Declare  C varchar(20);  V number (7,2);  Begin  C =’&Enter city’;  V=min\_sal(c)  Dbms\_output.put\_line(‘minimum salary=’ | | v); |

|  |
| --- |
| Exercise :  **6)develop a function to return the strength of the employees for a given branch code:**  Declare  Bcode number(2);  V = number(2);  Begin  Bcode=’&Enter branch code’;  V=Emp\_strength(bcode);  Dbms\_output.put\_line(‘strength of the employee=’ | | v);  End;  Create or replace function Min\_sal(c varchar)  Return number  Is  N number(5);  Begin  Select count(emp\_code) into n  From emp\_master  Where branch\_code=b;  Return n;  End;  / |

**FUNCTIONS :**

* A function is a subprogram that is used to compute and return a value. The syntax for creating a function is given below.

**Syntax 🡪**

CREATE or REPLACE function < func\_name >(Parameter-list)

RETURN datatype

Is / as

(local declarations)

Begin

Executable statements

[Exception]

Exception handlers

End;

* A function can take only IN parameters [Arguments]
* A function can take any number of IN parameters”

|  |
| --- |
| EXAMPLE:   * **List the strength of the employee for a given department number**   CREATE or REPLACE function DeptCount(p\_dno number)  RETURN number  is  n number(4);  Begin  SELECT count(empno)  INTO n  FROM employee  WHERE deptno = p\_dno;  RETURN n;  End; |

**TYPES OF PARAMETERS :**

1. IN parameter

**PARAMETER MODES :**

1. IN
2. OUT
3. INOUT

|  |  |
| --- | --- |
| EXERCISE :  6**)Develop a procedure to get employee address, city, pincode who has the second highest salary for a given branch code**  Declare  Bcode number(2);  V = number(2);  Begin  Bcode=’&Enter branch code’;  Emp\_details (bcode,a,c,p);  Dbms\_output.put\_line(‘employee details are as follows =’);  Dbms\_output.put\_line(‘employee address =’ | | a);  Dbms\_output.put\_line(‘employee city =’ | | c);  Dbms\_output.put\_line(‘employee pincode =’ | | p);  End;  Create or replace procedure Emp\_Details (bcode in number , a out varchar, c out varchar,p out number)  Is  Begin  Select address , city, pincode into a,c,p  From emp\_master  Where salary =  (Select max (salary)  From emp  Where branch\_code =bcode and  salary < (select max(salary) from emp Where branch\_code =bcode ));  End;  /   |  | | --- | | * **To get the First highest salary**   Select max (salary)  From emp  Where salary < (select max(salary) from emp);   * **To get the second highest salary**   Select max (salary)  From emp  Where branch\_code =bcode and  salary < (select max(salary) from emp Where branch\_code =bcode )); | |

Trainer : Thamarai Selvam Date : 02-JUL-2015

Time : 10.20am –pm

**ERROR HANDLING :**

* Error condition in PL/SQL is termed as exceptions
  1. Predefined Exception
  2. User–defined Exceptions
* An exception is raised when an error occurs. In case of an error the control is immediately transferred to the exception handling section
* Predefined exceptions are raised automatically
* User defined exceptions are raised explicitly using RAISE

**PREDEFINED EXCEPTION :**

* NO\_DATA\_FOUND
* TOO\_MANY\_ROWS
* INVALID\_CURSOR
* CURSOR\_ALREADY\_OPEN
* DUP\_VAL\_ON\_INDEX
* ZERO\_DIVIDE
* OTHERS

**USER DEFINED EXCEPTION :**

* A user-defined exception should be declared , raised explicitly by a RAISE statement and handled in the exception handling section.

SYNTAX:

*exception-name* EXCEPTION; 🡨 Declaration

RAISE *exception-name*; 🡨 Executable stmt

when *exception-name then* 🡨 Exception handler

*……………*

*……………* ;

|  |
| --- |
| **EXAMPLE :**  declare  v\_eno emp.eno%type);  v\_ename emp.ename%type;  v\_dept emp.deptno%type;  v\_salary emp.salary%type;  sal\_chk exception;  begin  v\_eno := &empno;  v\_ename := &empname;  v\_dept := &deptid;  v\_salary := &sal; |

* Cursor 🡪it is an handle to the Temporary memory[working]area

1. Implicit cursor.
2. Explicit cursor.

CREATING & DECLARING A CURSOR :

Cursor <cursor-name> is query

Cursor C is select from emp;

* %rowtype 🡪complete record will be stored

**CURSOR ATTRIBUTES** :

* %NOTFOUND – Returns a boolean. True if the recently executed SQL statement did not process a row.
* %FOUND – Returns a boolean. True if the recently executed SQL statement processed a row.
* %ISOPEN – Returns a boolean. True if the cursor is open. Cannot be checked for implicit or SQL cursors.
* %ROWCOUNT – Returns the number of rows processed by the recently executed SQL statement.

**EXPLICIT CURSOR :**

* Syntax:

Open <cursor-name>

Fetch <cursor-name> into <variable-name>

Close <cursor-name>

|  |
| --- |
| Example :   * To give an increment to a salary   declare  CURSOR empcur is  SELECT \* from emp;  v\_emprow empcur%rowtype;  v\_incr number(10,2);  begin  OPEN empcur;  loop  FETCH empcur into v\_emprow;  EXIT WHEN empcur%notfound;  if v\_emprow.dept = 'trg' then  v\_incr := 1000;  elsif emprow.dept = 'admin' then  v\_incr := 500;  else  v\_incr := 250;  end if;  UPDATE emp  SET salary = salary + v\_incr  WHERE eno = emprow.eno;  end loop;  CLOSE empcur;  dbms\_output.put\_line('Table updated');  end;  / |

**CURSOR FOR LOOP:**

**SYNTAX:**

For <recordname> in <cursorname>

Loop

…….

End loop

**DECLARING CURSOR FOR LOOP :**

Declare

Cursor c is select sal from emp;

V\_emprec c%rowtype;

For v\_emprec in c

Loop

End loop;

Open c;

Fetch c into v

Close c;

%notfound

%found

%isopen

%rowcount

Trainer : Thamarai Selvam Date : 06-JUL-2015

Time : 04.50pm –pm

**DATABASE TRIGGERS** :

* A database trigger is a stored procedure that is fired in response to specific data manipulation events like INSERT, UPDATE or DELETE issued against the associated table.

**Purpose of database triggers ?**

* To carry out a series of tasks related to the event
* To generate data automatically
* To enforce complex integrity constraints
* To customize complex security authorizations
* To audit data modification
* To maintain replicate tables

**BASE TABLE** :

* It refers to the table on mhich an insert/update/delete is performed.

**TRIGGERING TABLE :**

* As a consequence of an action on base table, Trigger is fired on a table is called as TRIGGERING TABLE.

**TRIGGER TIMING :**

* After/before deleting the record.

**TYPES OF TRIGGER:**

1. Row level trigger
   * Action takes place at row level
2. Statement level trigger
   * Action takes place at the entire table.

**TRIGGER COMPONENTS**

* 1. Triggering Event
  2. Trigger constraint ( optional )
  3. Trigger action

**TYPES OF TRIGGERS**

* 1. Row level triggers 🡨 Fires for every affected row
  2. Statement level triggers 🡨 Fires once for a DML statement

**TRIGGER TIMINGS**

* 1. Before triggers 🡨 Executed before the triggering event
  2. After triggers 🡨 Executed after the triggering event

**SYNTAX :**

CREATE or REPLACE trigger < *trigger\_name* >

BEFORE / AFTER

INSERT / UPDATE / DELETE

On < *table\_name* >

[ For each ROW ]

[ WHEN condition ]

Declare

< Declarations >

Begin

< Executable statements >

Exception

< Exception handlers >

End;

**TRIGGER ATTRIBUTES/ PSEUDO COLUMNS:**

1. :old.columnname
2. :new.columnname

* Also called as correlation names
* Are available for row level triggers only
* Pseudo records are available for the triggering table only and are similar to the rows of the triggering table.
* :old.triggering table’s column\_name
* refers to the old value of the column
* :new.triggering table’s column\_name
* refers to the new value of the column

|  |
| --- |
| **EXAMPLE :**  **1)To reduces the stock of the delivered item, to the extent of the delivered quantity.**  CREATE or REPLACE Trigger DeliveryChk  AFTER INSERT ON Delivery  FOR EACH ROW  BEGIN  UPDATE item\_master  SET qty = qty - :new.del\_qty  WHERE item\_cd = :new.item\_cd;  END; |

|  |
| --- |
| EXAMPLE :   1. Insertions to Employee Master can be made only on Monday and Tuesday from 10 AM to 1 PM.   create or replace trigger emp\_master\_trig  before  insert on emp\_master\_table  declare  d char(3);  t number(2);  begin  d:=to\_char(sysdate, ‘dy’);  t:=to\_number(sysdate,’ hh24’);  if d not in(lower(‘mon’),lower(‘tue’)) then  raise\_application\_error(-20000,’Access denied’);  else  if t<10 or t>13 then  raise\_application\_error(-20000,’Access denied’);  end if;  end if;  end;  / |

* raise\_application\_error 🡪 used to signal some error messages.

|  |
| --- |
| **FORMAT :**  raise\_application\_error(-number, error message);  -20000 to -29999 |

|  |
| --- |
| **EXAMPLE :**   1. **When an order is deleted cascade the operation to the order child**.   create or replace trigger order\_del\_trig  after  delete on order\_table  for each row  begin  delete from sales where order\_id =:old.order\_id;  end;  /  delete from order\_table  where order\_id=1; |

**PACKAGES:**

* A package is a database object which is an encapsulation of related PL/SQL types, subprograms, cursors, exceptions, variables and constants
* A Package consists of two parts :
  1. **Package Specification / Package Header**
     1. Contains declarations of variables, constants,
     2. PL/SQL types, exceptions, cursors & subprogram prototypes.
  2. **Package Body**

Contains the implementation of cursors, subprograms defined in the package specification

* Packages can be created using the following commands
  1. CREATE package command
  2. CREATE package Body command

**SYNTAX:**

CREATE package < *pack\_name* >

IS / AS

<declarations>

End < *pack\_name* >;

CREATE package body < *pack\_name* >

IS / AS

<Local declarations>

Begin

(executable statements)

End < *pack\_name* >;

|  |
| --- |
| EXAMPLE :  create or replace package emp\_pack  is  procedure emp\_details(bcode,a,c,p);  function min\_sal (c char(25)) return number;  end emp\_pack;  create or replace package body emp\_pack  is  begin  procedure emp\_details(bcode,a,c,p);  is  ….  begin  ….  end;  function min\_sal (c char(25)) return number  is  ….  begin  ….  end;  end emp\_pack;  emp\_pack.min\_sal(…) |