UNIT – IV

STRUCTURED QUERY LANGUAGE

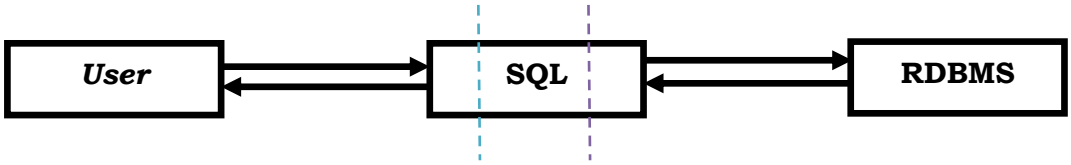
Introduction to SQL:

- The language, Structured English Query Language (SEQUEL) was developed by IBM to use E.F. Codd's relational model.
- It is developed by IBM in 1970's.
- Later it is known as SQL.
- SQL stands for structure query language.
- Structure means some standards.
- Query means question or requesting.
- Language means communication.

SQL is a collection of following languages.

1. Data definition language (DDL)
2. Data manipulation language (DML)
3. Transaction control language (TCL)
4. Data control language (DCL)

- It is heart of the RDBMS.
- It is database understandable language.
- It is 4th generation language.
- It is a powerful data manipulation language.
- It allows as interface between user and RDBMS



Characteristics of SQL:

1. SQL usage by its nature is extremely flexible. It uses a free form syntax that gives the user the ability to write SQL statements in a easy way.
2. Each SQL command is analyzed by the RDBMS before execution, to check for proper syntax and to process the command.
3. Unlike certain programming languages, there is no need to start SQL statements in a particular column or be finished in a single line.

Define Literal. Explain briefly different types of literals.

Literals are values that represent fixed value or constant. They can be used to supply values to database system and can also be used in value expressions. Literals can be classified into different types.

Text Literal: Text Literals contain characters. Text enclosed within single quotes. They have a maximum length of 4000 bytes.

Example:

- 'gm informatics'
- 'My friend'
- '15-jan-2018'

Integer Literal: Integer literals contain numerals up to 38 digits long. They do not contain decimal separators.

Example:

- 99
- -561

Number Literals: Number literals are the literals containing scientific notation, and decimal separators.

Example

- 99
- -69.421
- 23E-11

Boolean Literal: Boolean literals are predefine values that take either TRUE, FALSE OR NULL values. The TRUE and FALSE values are the outcome of 'if' and Loop statements. The Boolean values are assigned only to Boolean variables.

Write about data types in SQL.

Data Type: Data types will decide what kind of values need to be hold into the variables. Data types are used for to allocate memory for given data. The various data types available in SQL are as follows.

Character Data type: Character data types stores characters and strings up to 32k size. It can hold letters, numbers or binary data. Following are some data types of character data type.

Char

- It is used for to store alphanumeric values.

- It is used for to store fixed length characters.
- Default size is one byte.
- Maximum size is 256 bytes.

Varchar2

- It is also used for to store alphanumeric values.
- It is used for to store variable length strings.
- No default size.
- Maximum size is 4000 bytes.

Varchar

- It is also used for alphanumeric values.
- It is used for to store variable length strings.
- No default size.
- Maximum size is 2000 bytes.

Number: Number data type stores integer and floating point values.. It has the following format.

- Number (L) for integer
- Number (L,D) L specifies the length and D specifies the number of decimal places within the length.
- The maximum size is 38 digits
- The default size is 38 digits.

5. Date

- It is used for to store both date and time
- It allocate 7 fixed bytes of memory
- The default format of date is DD-MM-YY
- The default time zero hour's i.e. 12:00:00 AM.

Long: Long data type stores variable- length character strings. It stores up to 32760 bytes of data. At any instant, only one long column can be defined per table. Long columns cannot be used sub queries, functions, expressions, WHERE clauses or indexes. It is possible to insert a LONG value a LONG column.

Raw Data type: RAW data type stores fixed – length binary data with a maximum size of 2000 bytes. It can hold graphic characters or digitized pictures of up to 32k. The format of RAW data type is as follows:

RAW (size)

Where, size is an integer literal ranging from 1 to 32767. RAW column can hold 2KB of data. Hence, it is not possible to insert RAW value into RAW column.

Long Raw Data type: LONG RAW data type stores binary data like graphics characters or digitized pictures. It can hold maximum of 2GB data. Hence it allows large set of binary data.

LOB: large object data type stores unstructured information like manage file, video file, and audio file etc. its storage capacity up to 4GB.

Explain the concept of different types of operators in SQL.

Operator: An operator is a special kind of symbol which performs specific task.

Mainly there are 5 types of operators are available in SQL. They are

1. Arithmetic operators
2. Comparison operators(Relational)
3. Logical operators
4. Special operators
5. Set operators

***1. Arithmetic operators*:** Arithmetic operators are used to perform calculations based on number values. An arithmetic expression consists of column name with number data type and on arithmetic operator connecting them. The arithmetic operators are

Operator	Meaning	Syntax
+	Addition	a+b
-	Subtraction	a-b
*	Multiplication	a*b
/	Division	a/b
%	Modulo division	a%b
^	Raise to power	a^b

Display eno, ename, sal and sal plus commission.

Sql>Select eno,ename,sal, comm, sal+comm from emp;

Display after adding salary with commission is multiplied by 100

Sql>select eno, 100*(sal+comm) from emp;

2.Comparison operators: (Relational): Comparison operators are used in condition to compare one expression with another. The comparison operators are:

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> OR !=	Not Equal to

3. Logical operators: The logical operators are used to combine the result of two or more conditions top produce a single result. The logical operators are

And, or, not.

Operator	Meaning
AND	logical AND
OR	logical OR
NOT	logical NOT

Example:

Sql> select *from emp where eno=3 and sal=8000;

The above example displays all employee details from emp table provided both the conditions.

***4. Special operators*:** SQL allows the use of special operators in conjunction with the where clause. These operators include: between, in, like, is null, any, all, some, not in, not like, not between and is not null.

1. BETWEEN
2. IN
3. LIKE
4. IS NULL
5. EXISTS

***Between operator*:** It is used for to define range of values and it is check whether an attribute value within a range or not.

1. Display details of employees whose salaries ranging from 2000 to 5000.

Sql> select *from emp where sal between 2000 and 5000;

In operator: It is used to define list of values.

1. Display details of employees who are working in 10, 20 and 40 departments.

Sql> select *from emp where deptno in(10,20,40);

2. Display details of employees harini, maheswari

Sql> select *from emp where ename in('harini', 'maheswari');

Like operator: Sometimes we may not know the exact value for searching. We can select the rows that match a pattern by using like condition. The character pattern matching operator is referred to as a wild card search. Two symbols (%, _) can be used to construct the search strings.

- % It matches one or more characters.
- _ It matches any single character.

1. Display details of employees whose name starting with S.

Sql> select * from emp where ename like 's%';

2. Display details of employees who are receiving 4 digit salary.

Sql> select *from emp where sal '_ _ _ _ ';

***Is null operator*:** It is used for to test nulls.

1. Display details of employee who are not receiving commission.

Sql> select *from emp where comm is null;

EXISTS:

EXISTS command is used to execute a query based on the results of another query. If the sub-query returns at least a row, then the main query will be executed.

Syntax:

EXISTS (Sub Query)

Example:

Select EName, Job From Emp Where Exists(Select DName From Dept);

Select sid,sname,course from student where exists(Select cdesc from course);

Other operators:

ANY Operator:
We use the ANY operator in a WHERE clause to compare a value with any of the values in a list. ANY evaluates to true if the result of an inner query contains at least one row that satisfies the condition.
We must place an =, <>, <, >, <=, or >= operator before ANY.
Ex: SELECT * FROM employee WHERE salary > ANY (2000, 3000, 4000);

ALL Operator:
We use the ALL operator in a WHERE clause to compare a value with all of the values in a list. ALL evaluates to true when each value of inner query satisfies the condition.
We must place an =, <>, <, >, <=, or >= operator before ALL.
Example**:** SELECT * FROM employee WHERE salary > ALL (2000, 3000, 4000);

SOME Operator:

SOME operator compares a scalar value with a single-column set of values. SOME and ANY are equivalent.
Example: Select * From Employee Where salary=some (2750, 3000, 4000);

Explain about relational Set operators with examples:

Set operators combine two or more queries into a single result. The data type of the corresponding columns must be the same. The different set operators are:

1. Union
2. Union all
3. Intersect
4. Minus

***Union*:** The operator is useful when it is required to draw information from more than one table that have the same structure. It returns rows of the first query plus the rows of the second query after eliminating duplicate rows.

EMP

ENAME	SAL	DEPTNO
Madhu	20000	10
Kala	12000	20
Murali	12000	20

.

EMPS

ENAME	SAL	DEPTNO
Madhu	20000	10
Mahi	6000	20
Harini	5000	30

Sql> select * from emp union select * from emps;

ENAME	SAL	DEPTNO
Madhu	20000	10
Kala	12000	20
Murali	12000	20
Mahi	6000	20
Harini	5000	30

Union all: Union all is set operator used to retrieve the information from multiple data base tables without using join condition including the duplicate rows i.e. union all doesn't restrict duplicate rows.

Display details of employees **emp** and **emps** tables.

Sql> select *from emp union all select *from emps;

ENAME	SAL	DEPTNO
Madhu	8000	10
Madhu	8000	10
Kala	12000	20
Murali	12000	20
Mahi	6000	20
Harini	5000	30

Intersect: - Intersect returns only the rows those are common to both queries.

Display the common jobs of department 10 and 30

Sql> select *from emp intersect select *from emps;

ENAME	SAL	DEPTNO
Madhu	20000	10

Minus: This operator returns all the rows in the first table minus rows in the second table. In other words, it returns the rows present in the first table but not present in the second table.

Display the employees who exist in emp but not the emps table.

Sql> select *from emp minus select *from emps;

ENAME	SAL	DEPTNO
Madhu	8000	10
Kala	12000	20
Murali	12000	20

Write About Various Types Of Sql Commands

SQL commands can be categorized into several types based on their functionality in database operations:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Transaction Control Language (TCL)
4. Data Control Language (DCL)

All the above languages are referred as sublanguages of SQL.

DDL commands:

These are used to define various database objects i.e. tables, views, indexes etc. We can create, modify, delete and view the database objects by using these commands. The DDL commands are AutoCommit commands, since they process and save objects automatically in the database. All commands starting with CREATE, ALTER, DROP are treated as DDL commands.

Eg: CREATE TABLE, CREATE VIEW, ALTER TABLE, ALTER USER, DROP TABLE, DROP INDEX etc.

DML commands:

These commands are used to manipulate data within the ORACLE database. They are used to insert, delete, update, and query the data in the database objects. These commands do not implicitly commit the current transaction. SQL provides three data manipulation statements namely INSERT, UPDATE and DELETE.

TCL statements:

Transaction Control statements manage changes made by Data Manipulation Language commands. These commands are used to save or cancel the process of DML commands. COMMIT, ROLLBACK and SAVEPOINT commands are treated as TCL commands.

DCL commands:

These commands are used to control user access to the database objects. They used to grant or remove various privileges and roles to users. GRANT and REVOKE are treated as DCL commands.

Write about data definition language commands (DDL)

The data definition language commands are used to create objects (tables). Alter the table structure of an object, and also to drop the object, the DDL contains the following commands.

1. Create
2. Alter
3. Drop
4. Truncate

1. CREATE: The create command is used to create database objects such as table, view, index etc. The general syntax for create statement is.

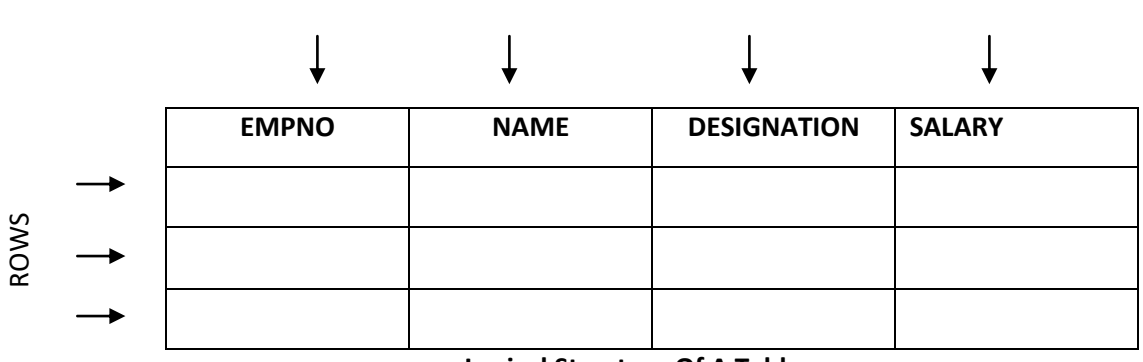
Syntax:

Sql> create table <table name> (<column name1> data type (size), <column name2> data type (size),…… <column name n> data type(size));

Example:

Sql> create table emp(eno number(4),ename varchar2(15), zip number(6), hdate date);

A table is one of the objects of Oracle database. It is the basic unit of data storage in a relational database management system. Every table has a table name and a set of columns (fields) and rows (records) in which the data is stored. Each column is identified by a column name and it is given a data type and width.



Logical Structure Of A Table

2. *Alter*: Sometimes there may be need to change the structure of a table. It is used for to modify structure of existing database table. This modification may be changing the datatype, constraints of the properties or adding new properties to the existing objects. The general syntax for alter command is

Syntax:

Alter table < table name> **keyword** (<column name1> data type (width));

Keywords

a. Modify b. Add c. Rename d. Drop

***a) Modify*:** It is used for

1. To increase or decrease size of the column.
2. To change data type of the column.
3. To change not null to null and null to not null constraint.

Example:

1. Increase the size of ename column name to 30 bytes.

Sql> alter table employee modify ename varchar2(30);

2. Change the data type of salry column to varchar2

Sql>alter table emp modify sal varchar2(20);

3. Change null constraint of job column to not null

Sql>alter table emp modify job not null;

b) Add: It is used for

1. To add new column to the existing table.
2. To add new constraint to the existing columns except not null constraint.

Example

1. Add remarks column to emp table.

Sql> alter table emp add remarks varchar2(40);

2.Add check constraint to the sal column .

Sql> alter table add check (sal between 15000 and 25000);

c) *Rename*: It is used for to rename a column.

Example: Rename sal column to salary column.

Sql> alter table emp rename column sal to salry;

d) *Drop*: It is used for to drop a column.

1. To remove a column remarks

Sql> alter table emp drop column remarks;

2. Remove pho and mail -id columns from emp table

Sql>alter table student drop(pho, mail-id);

3. Drop: It is used for to drop database object (table). Whenever we can use this command we can loss the data permanently we cannot recollect.

Syntax:

Sql> drop table <table name>

Examples

Remove student table from the database.

Sql> drop table student;

4. Truncate: The truncate command is used to delete the records (data) only not table structure. The general syntax for truncate command is.

Syntax: truncate table <table name>

Sql> truncate table student;

Write about data manipulation language commands (DML).

Data manipulation commands are most frequently used commands. These commands are used to query and manipulated data in an existing objects like table. Data manipulation commands are:

1. Insert
2. Select
3. Update
4. Delete

***1. Insert*:** It is used for to insert data into data base table. While giving data values each value must be separated by a comma symbol. The data values like char, varchar, varchar2, date, long etc must be enclosed within a pair of single quote symbols. This command adds the rows only at the end of the table. It doesn't insert any row in between existing rows of a table. The general syntax for insert command is

Syntax of adding table rows

Method 1

Sql>insert into <Table name> values (List of data values in the order of structure defined);

Method-2: We can also insert values into the table in the specified columns as follows:

Sql>INSERT INTO tablename (column-list) VALUES (values-list);

Here, the 'column-list' indicates the column names of the table. We can specify column names in any order. The number and order (data type) of values in the 'values-list' must match with the number and order of columns in 'column-list'.

Eg: INSERT INTO employee (Empname, Joindate) VALUES ('XYZ', '12-JUN-89');

Using address method: This will prompt you for the values but for every insert you have to use forward slash.

Syntax:

Sql>insert into <*table_name*) values *(&col1, &col2, &col3 …. &coln*);

This will prompt you for the values but for every insert you have to use forward slash.

***2. Select*:** It is used for to retrieve data from one or more data base tables. The general syntax for select command is

Syntax

SELECT [DISTINCT] */columnlist

FROM tablename1 [, tablename2, ...]

[WHERE condition]

[GROUP BY columnlist]

[HAVING condition]

[ORDER BY column [DESC],……] ;

Selecting rows with condition:

WHERE clause can be used to add conditional restrictions to the SELECT statement.

Syntax:

SELECT <Column List> FROM <Table List> WHERE <Condition>;

Example:

Select EName from Employee Where Dept='Marketing';

Select Sid, SName, Average from Student where Average>=75;

Selecting Unique Values:

While reading values, we can also read only unique values of a column. For this we use DISTINCT clause.

Syntax:

SELECT DISTINCT <Column Name> FROM <Table Name>;

Example:

Select Distinct Pname from Product;

Selecting Ordered Data:

When we read data values, they can also be ordered in either ascending or descending order.

Syntax:

SELECT <Column List> FROM <Table List> ORDER BY <Column Name> ASC/DESC;

Examples:

Select sname from Student Order By sname Asc;

Select sname from Student Order By sname Desc;

Select Eid, Ename From Employee Order By Salary Desc;

Selecting Computed Columns and Column Aliases:

While selecting multiple columns of a table we can also perform computations on those columns and we can name that result as a **column alias**.

Example:

Select Pid, PName, Price*QOH from Product;

Select sid, sname, maths+physics+cs As Total from Student_Results;

The **GROUP BY** clause is used to combine a group of rows based on particular column(s).

Eg:

1. SELECT JOB, COUNT(*) FROM employee GROUP BY job;
2. SELECT SUM(salary) FROM employee GROUP BY deptno, job;

The **HAVING** clause is used to specify a condition using group functions.

Eg:

SELECT JOB, COUNT (*) FROM employee GROUP BY job HAVING COUNT(*) > 3;

3. Update: The update command is used to modify an existing data in table.

i. The update command updates a single column or more than one column at a single time.

ii. Specific rows would be updated based on specific condition in where clause. The general syntax for update command is

Syntax

Update <Table name> set <column name> = <value> [where <condition>];

Increase salary 10000 for eno=3

Sql>update emp set sal =10000 where eno=3;

Condition base update example

Delete employee no=3 record

Sql> delete from emp where eno=3;

4. Delete: The delete command is used to delete data from table. The general syntax for delete command is

Syntax

Sql> delete from <Table name> [where <condition>];

Examples

Delete all rows from emp table

Sql> delete from emp;

Write about transaction control language commands (TCL)

All changes made to the database can be referred to as a transaction. A transaction begins with an executable statement and explicitly with both ROLLBACK and COMMIT statements and implicitly where a DDL statement is used. Transaction control languages are

1. Commit
2. Rollback
3. Save point

1) Commit: This command is used to end the transaction. Commit command is used to make changes permanent to the database. This command also erases all save points in the transaction. The general syntax for commit command is:

Syntax: Sql commit work; or

Sql: commit

2) Save point: Savepoints are like markers to divide a very large transaction into smaller ones. They are used to identify a point in a transaction to which user can rollback later. The general syntax for save point is:

Syntax*:* Sql> savepoint savepoint_name;

Sql>savepoint s1;

3) Rollback: This command is used to undo the work done in the current transaction user can either rollback the entire transaction or rollback a portion of transaction with savepoint.

Syntax to the roll back entire transaction:

Sql>rollback work; or

Sql rollback;

Syntax to the rollback to a particular stage in a transaction is

Sql>rollback to savepoint _name;

Sql rollback to savepoint s2;

Write about data control language commands. (DCL)

Data control language provides users with privilege commands. The owner of database objects has the sole authority over them. Granting privileges (insert, select, update, delete) to others allow then perform operations.

1. Grant
2. Revoke

***1. Grant*:** It is used for to give permissions on data base objects. A user can grant any object privilege on any other user.

Syntax:

grant <privilege>, <privilege> , … on <object name> to <user1>, <user2> , …

***Example*:** Grant select privilege to Harini user on emp tables

Sql> grant on emp to Madhu;

2. Revoke: This command is used to stop the permission other users.

Syntax:

Revoke <privilege> , <privilege> .. on <object name> from <user1>, <user2>,…;

Example

Deny select privilege of Harini user on emp table.

Sql> revoke select on emp from Madhu;

Group functions: (aggregate functions) (multiple row functions)

These functions process on multiple rows and return a single value. Each of the function accepts an argument. The group functions are:

1. Max ( )
2. Min ( )
3. Count ( )
4. Sum ( )
5. Avg ( )

1. MAX ( ) : This command is used to find the maximum value from the given numerical column.

Syntax: Max (numerical column)

Example: Write a query to find maximum sal earning by an employee in the emp table.

Sql>select max(sal) from emp;

2. MIN ( ) : This command is used to find the minimum value from the given numerical column.

Syntax: Min (numerical column)

Example:

Write a query to find minimum sal earning by an employee in the emp table

Sql>select min(sal) from emp;

3. COUNT ( ): The count function is used to count the number of rows. The general syntax for count function is

Syntax: count (* | distinct <column name>)

Write a query to count total number of records in the given table

- select count(*) from emp;

write a query to count, how many types of jobs available in the emp table

- select count(job) from emp;
  select count(*) from emp WHERE JOB='MANAGER';
  select count(*) from emp WHERE DEPTNO=30;
  Select deptno,count(*) from emp group by deptno;

4. SUM ( ): The function sum is used to obtain the sum of values in a column. The general syntax for sum column is

Syntax: sum (distinct <numerical column>)

SQL>Select sum (sal) from emp;

5. AVG ( ): This command is used to find the average value from the given numerical column.

***Syntax*:** Avg(distinct <numerical column>)

SQL>Select AVG (sal) from emp;

Write a query to find average salary of all employee in the emp table.

Sql>select avg(sal) from emp;

Write about group by and having clauses

ENO	ENAME	JOB	MGR	HDATE	SAL	COMM	DEPTNO
101	smith	clerk	110	17-DEC-10	20000	3000	20
102	allen	salesman	110	20-FEB-10	16000	5000	30
103	ward	salesman	111	09-JAN-11	12500	5000	10
110	scott	manager	105	09-MAR-09	18750	2000	10
111	king	manager	111	08-AUG-09	43750		10
105	Madhu	manager	110	09-AUG-10	25000		20

Group by clause:

It is used to divide the rows in a table into multiple groups based on one or more columns. We can then use the group functions to return summary information for each group.

Syntax:

Select [distinct] [column] group_function (column) …. From table [where condition] [group_ by _expression] [order by column];

Display maximum salary department wise.

Sql> select deptno, max(sal) from emp group by deptno;

SQL> select deptno,sum(sal) from emp group by deptno;

DEPTNO	SUM(SAL)
30	16000
20	45000
10	75000

Having clause:

It is used to restrict rows based on the result of a group function, then you must have a group by clause as well as the having clause.

Syntax:

Select [distinct] [column,] group_function (column) …. From table [where condition] [group _ by _ expression] [having having _ expression] [order by column];

SQL> select deptno,sum(sal) from emp group by deptno;

DEPTNO	SUM(SAL)
30	16000
20	45000
10	75000

Discuss about sub queries and correlated sub queries.

Sub Queries Or Nested Queries

A query within another query is called a sub query. We can define any number of sub queries with in a query. But the system executes the inner most query first, based on the inner query output, outer query will be executed.

The Following Points should be kept in Mind While using SubQueries

1. The inner Query must be enclosed in parentheses
2. The Inner query must be on the right hand side of the condition

DIFFERENT TYPES OF SUB QUERIES

- Single row subqueries

- Multi row subqueries
- Multiple subqueries
- Correlated subqueries

A sub query only that returns only one row is called as "Single row sub query". A sub query that returns more than one row is called as multiple row sub query.
The inner query always must be enclosed is braces. Sub query will be evaluated first followed by main query. All the inner and outer queries will be nested using comparison operators.

***Single row Sub queries*:**

In single row sub query, it will return only one value. The following operators are used with single row sub queries.< , >, <=, >=,==, !=

Example:

1.Display all the employees whose job same as allen job.

SQL> select * from emp where job=(select job from emp where ename='allen')

Display all the employees who belongs to smith employee department

Sql>select * from emp where deptno=(select deptno from emp where ename='smith');

Multi row sub queries:

In multi row sub query, it will return more than one value. In such cases we should include operators like any, all, in or not in between the comparison operators are used.

Example:

Display details of employee whose salary greate than the range of 20000 and 30000.

Sql> Select *from emp where sal > any (select sal from emp where sal between 20000 and 30000);

2. Retrieve that department names in which department does not have any employee.

SQL> select *from dept where deptno not in (select distinct(deptno) from emp);

Multiple sub queries:

There is no limit on the number of sub queries included in a where clause. It allows nesting of a query within a sub query.
Example: Display details of employee who are getting maximum salry in the employee table.

SQL> select * from emp where sal = (select max(sal) from emp where sal < (selectmax(sal) from emp));

Correlated sub queries:

In correlated sub query first outer query will be executed and based on the output of outer query, inner query will be executed.

Outer→inner→outer

Correlate sub queries are used for row-by-row processing .Each sub query is executed once for every row of the outer query.

A column from outer query will be substituted in the inner query condition using a table alias name is known as correlated column.

Example:

Display details of employee whose salary is greater than the average salary of their departments.

SQL> select eno, ename from emp e where sal > (select avg(sal)) from emp where (dept=e.dept);

UNIT – V
PL/SQL

INTRODUCTION TO PL/SQL

- It is a procedural language.
- It is an extension of non procedural language (SQL).
- PL/SQL program is collection of both procedural and non procedural statements.
- SQL is common for any data base but PL/SQL is a specific to oracle data base.
- SQL is a powerful data manipulation language. But PL/SQL is powerful data processing language.
- One block can contain any number of statements, it reduce network traffic.
- Block is a logical grouping of both procedural and non procedural statements.
- PL/SQL engine can find in the following two places
  - Oracle data base server.
  - Oracle tools. (like forms and reports i.e D2K)
- PL/SQL supports both client and server side programming.
- PL/SQL is especially design for to do activities at data base side known as data base programming or server side programming or stored procedures.
- PL/SQL is an application programming by using SQL statements. So it is 3rd generation language. The **advantages of PL/SQL** are as follows.
  - Re – usability
  - Compatibility
  - Portability
  - comprehensive

1. Re – usability: - It is used by many users and in many situations.

2. Compatibility: - Anyone can work with PL/SQL by remember programs to maintain data base.

3. Portability: - Share data with another data base or Re – use the data base.

4. Comprehensive: - Cover entire data base in detail means data base point of view.

Write about structure of PL/SQL Program

Pl/SQL stands Procedural Language/Structured Query Language. The structure of PL/SQL language is a block structure. Block is independent programs created and stores outside the data base. PL/SQL extends SQL by adding extra features like variable, conditional statements, looping structures, cursor management; exception handling statements etc.PL/SQL combines the flexibility of SQL with the power of programming language.

```
DECLARE
        Which stores variable names
        Which stores object names
BEGIN
        <SQL and PL/SQL executable statements>
        <Procedural constraints>
EXCEPTION
        < Error handling statements>
END ;
```

Declare section: Declare section is optional section. It is used to place all the declarative statements that declare variables, cursors, exceptions etc. All sql data types are supported. Boolean is also supported. Executable Statements DML, TCL are supported. DDL, DCL are not supported.

Begin Section: Begin section is Mandatory. It is used to place all the executable statements that process the data of the program. It contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.

Exception section: It is an optional section. It is used to place all the exception handling statements of the program. Exception block is used for to handle run time errors.

End Section: It is mandatory to place PL/SQL Program. Each procedure block must be terminated by the keyword END.

- Declare, Begin, Exception and End are PL/SQL keywords
- PL/SQL is not a case sensitive
- Each and every PL/SQL statement must end with semicolon.

Write about PL/SQL Data types:

Mainly PL/SQL data types are 2 types

1. Scalar data types
2. Composite data types

All PL/SQL data types defined in a package called 'standard".

1. Scalar data types:

a. Number

- It is used for both integer and float numbers.
- Number (P) for integer
- Number (p, s) for float
- The maximum size is 38 digits
- The default size is 38 digits.

b. varchar2:- It is also used for to store alphanumeric values.

- It is used for to store variable length strings.
- No default size.
- Maximum size is 4000 bytes.

c. Varchar

- It is also used for alphanumeric values.
- It is used for to store variable length strings.
- No default size.
- Maximum size is 2000 bytes.

d. Char

- It is used for to store alphanumeric values.
- It is used for to store fixed length characters.
- Default size is one byte.

- Maximum size is 256 bytes.

e. Date

- It is used for to store both date and time
- It allocate 7 fixed bytes of memory
- The default format of date is DD-MM-YY
- The default time zero hour's i.e. 12:00:00 AM.

f. long :- This data type is similar to varcjhr2 data type. The maximum size of this data type is 32767 bytes.

Syntax: - long (l);

g. Boolean: - The Boolean data type can hold only true, false and null. The default value for this data type is 'False'

h. rowid:- The rowid PL/SQL type is same as in the data base rowid pseudo column type. it is 18 characters length string.

COMPOSITE DATA TYPES (PL/SQL attributes)

It is used to define the PL/SQL variables dynamically according to table structure. The following types of attributes supported by PL/SQL.

1. %Type
2. %Rowtype

%Type: - It is column type declaration.

It is used top define the variables according to the specific column structure.

Syntax: - variable <table name>. <Column name>;

Veno empp.empno%type;

%Rowtype: - it is record type declaration.

It is used to define the variable according to the complete table structure.

Syntax: variable <table name> %row type;

I emp%rowtype;

Write about PL/SQL Operators.

Concatenation Operator: The concatenation operator is represented as double vertical bars (||). It is designed to attach one string at the end of another string. The string may be CHAR, VARCHAR2, CLOB, or the equivalent Unicode enabled type.

Example:

'quantity' || 'delivered'

The above expression returns the following value.

'quantitydelivered'

Comparison Operator: The comparison operators are designed to compare one expression to another. And the generated result can be among true, false or NULL. This operator is generally use in conditional control statements, WHERE clause of SQL data manipulation statements.

Example:

```
IF qty_delivered > THEN

UPDATE storage SET qty =qty-1

WHERE number  =item_number;

ELSE

.

.

.

END IF;
```

Write about Control structures in PL/SQL.

The control structures are used to control the flow of program execution. In PL/SQL there are there are three types of flow statements are exist.

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

1. CONDITIONAL CONTROL:

A conditional control structure tests a condition to find out whether it is true or false. Based on the result of the condition it executes one block among several mutually exclusive blocks. In PL/SQL, IF statement is called as conditional control statement. This statement can be used in the following forms.

- ☞ IF-THEN
- ☞ IF-THEN-ELSE
- ☞ IF-THEN-ELSEIF
- ☞

IF-THEN:

It is the simplest form of IF conditional statement. It is used to execute one set of statements based on the result of a given test condition.

Syntax

```
IF<CONDITION>THEN
Sequence of statements;
END IF;
```

The statement block is allowed to execute if the condition evaluates to TRUE, otherwise it will be skipped.

IF-THEN-ELSE:

This statement is used to execute one set of statements among two mutually exclusive set of statements. Therefore, at any time, only one block of statements will be executed.

Syntax

```
 IF<CONDITION>THEN
Sequence of statements;
ELSE
```

Sequence of statements;
END IF;

In this case, if we use number of else if statements but at end we use only one end if.

If the test condition evaluates to TRUE, statement block 1 will be executed, if it evaluates to FALSE, statement block 2 will be executed.

IF-THEN-ELSIF:

This statement is used when we compare more than one condition to execute one set of statements among several mutually exclusive statements.

Syntax

IF<CONDITION>THEN
Sequence of statements;
ELSE IF < CONDITION>THEN
Sequence of statements;
END IF;

In this case, if we use number of else if statements but at end we use only one end if.

If the test condition evaluates to TRUE, statement block 1 will be executed, if it evaluates to FALSE, statement block 2 will be executed.

Ex:

```
DECLARE
    dno number(2);
BEGIN
    select deptno into dno from dept where dname = 'ACCOUNTING';
    if dno = 10 then
      dbms_output.put_line('Location is NEW YORK');
    elsif dno = 20 then
          dbms_output.put_line('Location is DALLAS');
    elsif dno = 30 then
          dbms_output.put_line('Location is CHICAGO');
    else
          dbms_output.put_line('Location is BOSTON');
    end if;
END;
```

2. ITERATIVE CONTROLS:

The sequence of statements can be executed any number of times using loops. Loops in PL/SQL can be broadly classified as

a. Simple loop
b. While loop
c. For loop

a. Simple loop: The keyword loop should be placed before the first statements of sequence of statements and the keyword end loop should be placed after the last statements in the sequence. This is infinite loop. The sequence of statements can be executed infinitely. If we want to exit the loop we use exit keyword.

Syntax

loop
Sequence of statements;
End loop;

Ex:

```
DECLARE
    i number := 1;
BEGIN
    loop
       dbms_output.put_line('i = ' || i);
       i := i + 1;
       exit when i > 5;
    end loop;
END;
```

(b) While loop: The while loop statements include a condition associated with sequence of statements. if the condition evaluated true then the sequence of statements will be executed. This sequence of statements will be executed .this sequence of statements executed up to if while condition evaluate false.

Syntax

While<condition>
Loop
Sequence of statements;
End loop;

Ex:

```
DECLARE
    i number := 1;
BEGIN
    While i <= 5 loop
```

```
            dbms_output.put_line('i = ' || i);
            i := i + 1;
        end loop;
    END;
```

For loop: The number of iterations for a while loop is known until the loop terminator where as the number of iterations in for loop is known before the loop gets executed. By default iteration proceeds from lower bound to upper bound. If we use the keyword reverse the iteration proceeds from upper bound to lower bound.

Syntax

for variable name in (reverse) lower bound upper bounds (step number)
Loop
Sequence of statements;
End loop;

Ex1:

```
BEGIN
    For i in 1..5 loop
        dbms_output.put_line('i = ' || i);
    end loop;
END;
```

3. SEQUENTIAL CONTROL STRUCTURES:

The sequential control passes the control to specified label unconditionally. It can pass in forward direction or backward direction.

Goto: PL/SQL includes goto statements.

Syntax : goto < label name>

In PL/SQL labels are enclosed with in double angle (<< >>) brackets when a goto statement evaluated control immediately passes to the statement identified by the label..

Ex
Declare
Count number;
Veno number;
Begin
Veno:= &veno;
Update emp set eno= veno;

```
If (veno :=100) then
Goto insert
Endif
<<insert>>
Insert into emp (eno ) values (veno);
End;
```

CURSORS

In PL/SQL, it is not possible for an SQL statement to return more than one row. In such cases we can use cursors. A cursor is a mechanism that can be used to process the multiple row result sets one at a time.

- ☞ A cursor is nothing but a pointer b/w the contest area and physical data storage.
- ☞ It is a temporary buffer used to hold the transactional data for manipulation purpose.
- ☞ It is not stored in database.
- ☞ It is not re-usable.
- ☞ It is valid in PL/SQL block only.
- ☞ It is created in the logical memory.
- ☞ Cursors can be divided into **two** types.
    a. Implicit cursor.
    b. Explicit cursor

A) **IMPLICIT CURSORS**: Automatically created by oracle whenever "DML" operations are performed by user.

Implicit Cursors:

PL/SQL implicitly declares a cursor for every SQL DML statement such as INSERT, DELETE, UPDATE and SELECT statement that is not a part of an explicitly declared cursor even if the statement processes a single row. PL/SQL allows referencing the most recent cursor or the cursor as 'SQL' cursor.

Cursor attributes are used to access information about the most recently executed SQL statement using SQL cursor.

Attributes:

In PL/SQL, both implicit and explicit cursors have four attributes

1. %FOUND
2. %NOTFOUND
3. %ROWCOUNT
4. %ISOPEN

The general syntax for cursor attributes is

Syntax: SQL%<Attribute Name>

<Cursor name >%<attribute>

1. **%IS OPEN:** Return true or false

Return true if cursor is opens successfully els false.

2. % FOUND: Returns True/False.

Returns true if fetch statement successfully retrieves the data into PL/SQL variables.

3. %**NOT FOUND**: Returns True/False.

Returns true if fetch statement fails to retrieve the data into PL/SQL variables.

4. %**ROW COUNT**: returns number.

Returns the no of rows successfully retrieved from cursor so for initially it holds 0. After every successful "Fetch" it is incremented by one.

Example:

```
Select * From Student;

  SID SNAME                GENDER
----- -------------------- ------
  100 Anil Kumar           Male
  101 Lavanya              Female
  102 Hema Gopika          Female
  103 Pavan Kalyan         Male
  104 Lokesh Krishna       Male
```

```
Declare
    rows Number;
Begin
  UPDATE Student Set Gender='M' Where Gender='Male';
  IF SQL%ISOPEN THEN
       dbms_output.put_line('Cursor is in OPEN state');
  ELSE
       dbms_output.put_line('Cursor is in CLOSED state');
  END IF;
  IF SQL%NOTFOUND THEN
    dbms_output.put_line('No Student Updated');
  ELSIF SQL%FOUND THEN
    rows := SQL%ROWCOUNT;
    dbms_output.put_line( rows || ' Students Updated ');
  End If;
End;
```

Output:

```
Cursor is in CLOSED state
3 Students Updated
```

Result:

```
Select * From Student;

  SID SNAME                GENDER
----- -------------------- ------
  100 Anil Kumar           M
  101 Lavanya              Female
  102 Hema Gopika          Female
  103 Pavan Kalyan         M
  104 Lokesh Krishna       M
```

```
Declare
     Name Student.SNAME%Type;
Begin
    Select SNAME into Name From Student Where SID=102;
    IF SQL%FOUND THEN
        dbms_output.put_line('Student Name is ' ||Name);
    ELSIF SQL%NOTFOUND THEN
        dbms_output.put_line( 'No student exists with that SID');
    END IF;
End;
```

Output:

```
Student Name is Hema Gopika
```

B) **EXPLICIT CURSORS**: Created by user in PL/SQL block. It is used to retrieve multiple rows from multiple tables into PL/SQL block for manipulation purpose .It is based on select statement. Explicit cursor can be classified into **3types**.They are:

1. Declaring cursor
2. Cursor operations
3. Cursor attributes

1. **DECLARING CURSOR**:

Declaring cursor means using the select statement. Declaring Cursor functionally in the declarative part of the PL/SQL block. In declaring cursor any select statement is legal including joins and set operations.

Syntax Cursor<Cursor name>is<Select statement>

Ex: Cursor C1 is select *from emp;

Here C1 is cursor name. It declares as selecting all data from emp table.

2. **CURSOR OPERATIONS**:

i) **Opening cursors**: It is used to open the cursor memory will be allotted to cursor after opening it.

Syntax Open<Cursor name>;

ii) **Fetch the data**: used to retrieve data from cursor to PL/SQL variables. At a time it can retrieve only one row into variables. Generally Fetch will be place in loop.

Syntax Fetch<Cursor name>into<variables>;

iii**) Close the cursor**: It is used to close the cursor memory allotted will be de allocated.

Syntax Close<cursor name >

Steps in using Explicit Cursors:

Declaring the Cursor:

Like other variables, a cursor must be declared in the DECLARE section of a PL/SQL block. Each cursor is associated with a SELECT query.

Syntax: CURSOR <Name of the Cursor> IS SELECT Query;

Open the Cursor:

After declaration, the cursor is opened with an OPEN statement for processing rows in the cursor. The SELECT statement associated with the cursor is executed when the cursor is opened, and the active set associated with the cursor is created. The active set contains the rows that satisfy the SELECT query.

Syntax: OPEN <Name of the Cursor>;

Fetching data from the Cursor:

After a cursor has been opened, the SELECT statement associated with the cursor is executed and the active set is created. To retrieve the rows in the active set one row at a time, the rows must be fetched individually from the cursor. After each FETCH statement, the cursor will move to the next row in the active set and retrieves.

Syntax: FETCH <Name of the Cursor> INTO <Variable Name>, <Variable Name>, . . .

Closing the Cursor:

After processing the rows in the cursor, it is released with the CLOSE statement. Once the cursor is CLOSED, it can be opened again.

Syntax: CLOSE <Name of the Cursor>;

Note: The attributes explained in the above section, can also be used with Explicit Cursors also.

Example:

```
Select * From Student;

   SID SNAME                GENDER
------ -------------------- ------
   100 Anil Kumar           M
   101 Lavanya              Female
   102 Hema Gopika          Female
   103 Pavan Kalyan         M
   104 Lokesh Krishna       M
```

```
Declare
    std_row  Student%ROWTYPE;
    CURSOR selectcur IS Select SID, SNAME, GENDER From Student;
Begin
    OPEN selectcur;
    LOOP
       FETCH selectcur INTO std_row.sid, std_row.sname, std_row.gender;
       EXIT WHEN selectcur%NOTFOUND;
       dbms_output.put_line(std_row.sid ||'  '|| std_row.sname ||'  '|| std_row.gender);
     END LOOP;
    CLOSE selectcur;
End;
```

Output:

```
100     Anil Kumar    M
101     Lavanya    Female
102     Hema Gopika    Female
103     Pavan Kalyan    M
104     Lokesh Krishna    M
```

Declaring Parameterized Cursors:

PL/SQL allows declaration of cursors that can accept input parameters which can be used in SELECT statement with WHERE clause.

Syntax: CURSOR <Name of the Cursor> [(parameter list)] IS SELECT ..... WHERE <col>=parameter;

Passing Parameters to Cursor:

Parameters to a parameterized cursor can be passed when the cursor is opened.

Syntax: OPEN <Name of the Cursor>(Parameters);

Example:

```
Select * From Student;

   SID SNAME                GENDER
------ -------------------- ------
   100 Anil Kumar           M
   101 Lavanya              Female
   102 Hema Gopika          Female
   103 Pavan Kalyan         M
   104 Lokesh Krishna       M
```

```
Declare
      std_row  Student%ROWTYPE;
      CURSOR selectcur(id Number) IS Select SID, SNAME, GENDER From Student Where SID=id;
Begin
      OPEN selectcur(100);
      LOOP
           FETCH selectcur INTO std_row.sid, std_row.sname, std_row.gender;
           EXIT WHEN selectcur%NOTFOUND;
           dbms_output.put_line(std_row.sid ||'  '|| std_row.sname ||'  '|| std_row.gender);
      END LOOP;
      CLOSE selectcur;
End;
```

Output:

```
100    Anil Kumar    M
```

Cursor FOR LOOP:

PL/SQL provides FOR loop to manage cursors effectively in situations where the rows in the active set are to be repeatedly processed in a looping manner. A Cursor FOR loop simplifies of processing of a cursor. Cursor FOR loop can be used instead of the OPEN, FETCH and CLOSE statements.

A cursor FOR loop implicitly opens the cursor, fetches the data and closes the cursor. It also declares a record of type %ROWTYPE.

Syntax: **FOR <Record Name> IN <Name of the Cursor> LOOP**

Loop Body

END LOOP;

Example:

```
Select * From Student;

   SID SNAME                GENDER
------ -------------------- ------
   100 Anil Kumar           M
   101 Lavanya              Female
   102 Hema Gopika          Female
   103 Pavan Kalyan         M
   104 Lokesh Krishna       M
```

```
Declare
    CURSOR display IS Select * From Student;
Begin
    FOR StudRec IN display LOOP
      dbms_output.put_line(StudRec.sid||StudRec.sname||StudRec.gender);
    END LOOP;
End;
```

Output:

```
100 Anil Kumar           M
101 Lavanya              Female
102 Hema Gopika          Female
103 Pavan Kalyan         M
104 Lokesh Krishna       M
```

1. Define Data dictionary?

- Data dictionary is a collection of system defined tables.
- This includes elementary-level data items (fields), group and record-level data structures and relational tables.
- It keeps track of relationships that exist between various data structures.
- Data dictionary is the main source of information for any RDBMS.
- All data dictionary tables belong to SYS user account.
- The system defined tables are created automatically whenever the data base is created.

2. What is a data model? Name any four data models.

- A conceptual method of structuring data is called Data Model.
- The development of systems based on three principal data models.
- These four data models are

  Hierarchical
  Network
  Relational
  Object-oriented

3. What is an entity?

- Anything about which data are to be collected and stored.
- It can be anything in the real world, which has got certain properties, which is identified.

4. Define View

- A view is a virtual table in the database defined by a query. A view does not exist in the database as a stored set of data values.
- The row and columns of data are visible through the view are produced by the query that defines the view.

5. What is SQL join?

- Join is a set operation.
- Join operations take two tables and return another table as a result.
- It is used to define relationship between columns of 2 different tables.
- It is required to retrieve data from more than one database table.

JOIN: It allows information to be combined from two or more tables. JOIN is the real power behind the relational database, allowing the use of independent tables linked by common attributes.

6. What is star schema?

The star schema is a data modeling technique used to map multi – dimensional decision support data into a relational database. Star schema creates a multi –dimensional database schema from the existing relational database. Star schemas yield an easily implemented model for multidimensional data analysis. The basic star schema has four components.

1. Facts
2. Dimensions
3. Attributes
4. Attribute hierarchies

7. Keys

In the relational model, keys are important because they are used to ensure that each row in a table is uniquely identified. They are also used to establish relationships among tables and to ensure the integrity of the data. Therefore, a proper understanding of the concepts and use of keys in the relational model is very important. A key consists of one or more attributes that determines other attributes.

Key: - A key is an attribute or combination of attributes that identifies (determine) other attributes uniquely.

Simple key: The key which consist only one attribute is called as "simple key".

Super key: An attribute (or combination of attributes) that uniquely identifies each row in a table. **Or**

Super key of an entity set is a set of one or more attributes whose values uniquely determine each entity.

Composite key: The key which consist multiple attributes is called as "composite" key. (OR)

If a primary key contains two or more attributes then that type of primary key is called composite key.

Candidate Key: A minimal (irreducible) super key. A super key doesn't contain a subset of attributes that is itself is a super key.

Primary Key: A candidate key selected to uniquely identify all other attribute values in any given row. Cannot contain null entries.

Secondary Key: An attribute (or combination of attributes) used strictly for data retrieval purpose.

Foreign Key (Referential integrity constraint): - It is used to establish relation between two tables.

- Used to define relationship between 2 Tables.
- It allows Null and duplicates values.

It can be related to either Primary key or unique constraint column of other Table.

PK / UNQ <-----> FK

8. Define System catalog.

The system catalog is defined as detailed system data dictionary that describes all objects within the data base. System catalog contains data about table name, its owner, created date, the number of columns in each table, the data type of each column, index file names, index creators, authorized users and all the access privileges. Sometimes both the term data dictionary and system catalog are used interchangeably.

9. Define normalization.

It is a step by step procedure used to design the final database objects from raw data by avoiding "Data Redundancy" and improving “Data Integrity and Data Concurrency ".

10. Define entity clustering.

Some wide information’s systems have over 1000 entity types and relationships. In this case, we cannot define the entire organizational data model into the ER diagram. To overcome this problem approach is to create multiple ER diagrams. But in this situation the problem is the total picture could not exist. To solve this problem, we use the method of entity clustering.

Entity clustering is a useful way to represent a data model for a large and complex organization, here we group the related entity types and its associated relationships called entity clusters.

11. What is the role of DBA?

A database administrator is a person or a group responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring s/w and h/w resources as needed.

The functions of DBA include

- Database Design
- User Training
- Database Security and Integrity
- Database System Performance

12. What is the multi-valued Attribute?

An attribute that can have multiple values is called Multi-Valued Attribute. For example, a person can have several college degrees. A person can have multiple phone numbers.

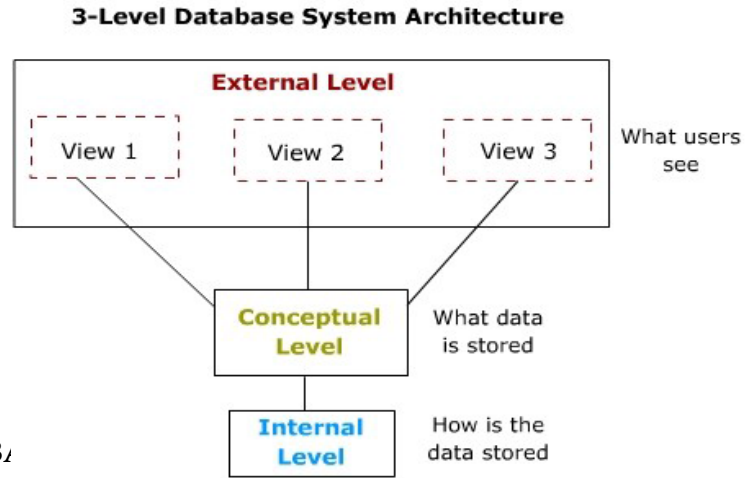
13. What are the different levels of data abstraction?

There are three levels of abstraction/Architecture:

Internal/ Physical level: The lowest level of abstraction describes how data are stored.

Conceptual/Logical level: The next higher level of abstraction, describes what data are stored in database and what relationship among those data.

External/View level: The highest level of abstraction, enable to view or access only a part of the database.



14. Define database schema.

It is a collection of related data base objects placed under one user account known as schema.

Internal schema.

Conceptual schema.

External schemas or user views.

15. Write any two differences between DROP and TRUNCATE commands.

Drop: - It is used for to drop database object (table). Whenever we can use this command we can loss the data permanently we cannot recollect.

Syntax:

Sql> drop table <table name>

Truncate: The truncate command is used to delete the records (data) only not table structure. The general syntax for truncate command is.

Syntax: Truncate table <table name>

Sql> truncate table student;

16. Define OLAP.

OLAP create an advanced data analysis environment that supports decision making, business modeling and operations research. OLAP system shares four main characteristics.

1. They main Multi- dimensional data analysis techniques.
2. They provide advanced database support
3. They provide easy-to-use end user interfaces.
4. They support client / server architecture.

17. What is a business rule?

A business rule is a brief precise description of policy, or procedure or principle within the organization. Business rules are not universal, different organizations may have different business rules.

31. Write syntax of a table creation of SQL.

Create: - The create command is used to create a table. The general syntax for create statement is.

Syntax:

Sql> create table <table name> (<column name1> data type (size), <column name2> data type (size), …….);

Example :

Sql> create table emp(eno number(4),ename varchar2(15), zip number(6), hdate date);

18. Give an example query with 'where' clause.

Where clause: - it is used for to filter the data based on given condition known as restriction.

Syntax: select [distinct] <col 1>, <col 2> … from <tab 1> , <tab 2> where <condition>;

Display details of employee whose salary more than 20000

Sql> select *from emp where sal >20000;

19. Write syntax of UPDATE command.

Update: - The update command is used to modify an existing data in table.

iii. The update command updates a single column or more than one column at a single time.

iv. Specific rows would be updated based on specific condition in where clause. The general syntax for update command is

Syntax

Update <Table name> set <column name> = <value> [where <condition>];

Increase salary 10000 for eno=103

Sql>Update emp set sal =10000 where eno=103;

20. Name five commercial RDBMS Software.

Oracle
Sybase
Informix
MySQL
DB2
Ingres II
SQL Server

21. Define information system.

Information system facilitates the transformation of data into information and allows the user for managing both data and information. Database occupies major portion of an information system that provides facilities for data storage, manipulation and retrieving.

22. What constitutes a DBMS?

The database system is composed of the five major pars.
Hardware
Software
People

Procedures
Data

23. What is normal form?

A normal form is a state of relation that results from applying simple rules regarding functional dependencies or relationships between attributes to the relation.

24. Why E-R modeling is used?

The Entity relationship model is a detail graphical representation of data from an organization. An ER Model is normally expressed in terms of ER diagrams, which is graphical representation of an ER Model.

Benefits of ER Modeling

- Documents information for the organization in clear, precise format.
- Provides a clear picture of scope of the information requirement.
- Provides an easily understood pictorial map for the database design.
- Offers an effective framework for integrating multiple applications.

25. What is meant by de-normalization?

De-normalization is a strategy that database managers use to increase the performance of database infrastructure. It involves adding redundant data to a normalized database to reduce certain types of problems with database queries that combine data from various tables into a single table.

26. Why database is important.

It is a collection of programs working together to control Data Manipulations (add , change ,remove), Retrievals (Read) and sharing on Database.

DBMS is to provide an environment that is both convenient and efficient for people to use in retrieving and storing information.
The main objective of database is record the history for the future use.

27. Write any two advantages of DBMS.

1. Controlling redundancy.
2. Restricting unauthorized access.
3. Providing persistent storage for program objects and data structures.
4. Providing multi-user interfaces.
5. Representing complex relationships among data.
6. Enforcing integrity constraints.
7. Providing backups and recovery.

28. What is functional dependency?

A functional dependency is a constraint between two attributes or two sets of attributes. If there are two attributes A and B, then attribute B is said to be functionally dependent on attribute A if each value in A determines

29. What is Generalization and Specialization?

Generalization: An object set that is a superset of another object set.
Specialization: An object set that is a subset of another object set.

Statements

A statement is a syntactic construction that performs some action when the program is executed.

1. Simple statements/ Expression statements
2. Compound statements / Block of statements
3. Control statements.

Level 1

Simple statement: A simple statement is a single statement that ended with a semicolon.

1. Write a PLSQL program to print your name (Single statement).

```
begin
dbms_output.put_line('Enrich');
end;/
```

Level 2

Compound statements: Any sequence of simple statements can be grouped together and enclosed with in a pair of braces termed as compound or block of statements.

2. Write a PLSQL program to print your name and address (Multiple statements).

```
begin
dbms_output.put_line('Hai');
dbms_output.put_line('Welcome to ');
dbms_output.put_line('Enrich Computer Education');
end;/
```

3. Write a PLSQL program to read and print a name.

```
declare
name varchar2(20);
begin
name:='&name';
dbms_output.put_line('your entered name is: '||name);
end;/
```

4. Write a PLSQL Program to read and print someone name and blood group.

```
declare
name varchar2(20);
blood_group varchar2(20);
begin
name:='&name';
blood_group:='&blood_group';
dbms_output.put_line('you are entered name is:   '||name);
dbms_output.put_line('you are entered blood group is: '||blood_group);
end;/
```

5. Write a PLSQL program to read and print any two numbers.

```
declare
N1  number;
N2  number;
begin
N1:=&N1;
n2:=&N2;
dbms_output.put_line('The first number is  '||N1);
dbms_output.put_line('The second number is '||N2);
end;/
```

Level 3

6. Write a PLSQL program addition of two numbers.

```
declare
a number(3):=10;
b number(3):=30;
c number(3);
begin
c:=a+b;
dbms_output.put_line(' first number '|| a);
dbms_output.put_line('second number '|| b);
dbms_output.put_line('addition of two numbers: '||c);
end;/
```

7. Write a program to accept two different numbers and find addition, subtraction, multiplication and division of given number.

```
declare
a number(2):=&a;
b number(2):=&b;
addition number(3);
subtraction number(3);
multiplication number(3);
division number(3);
begin
addition:=a+b;
subtraction:=a-b;
multiplication:=a*b;
division:=a/b;
dbms_output.put_line('ADDITION '||addition);
dbms_output.put_line('SUBSTRACTION '||subtraction);
dbms_output.put_line('MULTIPLICATION '||multiplication);
dbms_output.put_line('DIVITION '||division);
end;/
```

8. Write a PLSQL program to find out area of the rectangle.

```
declare
length number(8);
breadth number(7);
area number(20);
begin
length:=&length;
breadth:=&breadth;
area:=length*breadth;
dbms_output.put_line('area of the rectangle  '||area);
end;/
```

Level 4

Control statements: Generally programs are executed in sequential order from top to bottom. But sometimes it is necessary to change the order of executing statements based on certain condition. And repeat a group of statements until certain conditions false. Such statements are called control statements.

IF-THEN Statement

The simplest form of IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF (not ENDIF), as follows:

```
IF condition THEN
   sequence_of_statements
END IF;
```

The sequence of statements is executed only if the condition is true. If the condition is false, the IF statement does nothing.

Write a PLSQL program to demonstrate IF-THEN Statement:

```
declare
a number(2);
begin
a:=&a;
if a>=18 then
dbms_output.put_line(' he is eligible t vote');
end if;
end;/
```

IF-THEN-ELSE Statement

The second form of IF statement adds the keyword ELSE followed by an alternative sequence of statements, as follows:

```
IF condition THEN
  sequence_of_statements1
ELSE
  sequence_of_statements2
END IF;
```

Write a PLSQL program to demonstrate IF-THEN ELSE Statement:

```
declare
a number(2);
b number(2);
begin
a:=&a;
b:=&b;
if a>b then
dbms_output.put_line(a||' is biggest');
else
dbms_output.put_line(b||' is biggest');
end if;
end;/
```

The sequence of statements in the ELSE clause is executed only if the condition is false. Thus, the ELSE clause ensures that a sequence of statements is executed.

IF-THEN-ELSIF Statement

Sometimes you want to select an action from several mutually exclusive alternatives. The third form of IF statement uses the keyword ELSIF (not ELSEIF) to introduce additional conditions, as follows:

```
IF condition1 THEN
   sequence_of_statements1
ELSIF condition2 THEN
   sequence_of_statements2
ELSE
   sequence_of_statements3
END IF;
```

Write a PLSQL program to demonstrate IF-THEN ELSIF Statement:

```
declare
```

```
a number(2);
b number(2);
begin
a:=&a;
b:=&b;
if a=b then
dbms_output.put_line(a||'and'||b|| 'are equal');
elsif a>b then
dbms_output.put_line(a||' is biggest');
else
dbms_output.put_line(b||' is biggest');
end if;
end;/
```

If the first condition is false or null, the ELSIF clause tests another condition. An IF statement can have any number of ELSIF clauses; the final ELSE clause is optional. Conditions are evaluated one by one from top to bottom. If any condition is true, its associated sequence of statements is executed and control passes to the next statement. If all conditions are false or null, the sequence in the ELSE clause is executed. Consider the following example:

When possible, use the ELSIF clause instead of nested IF statements. That way, your code will be easier to read and understand. Compare the following IF statements:

```
IF condition1 THEN                    |    IF condition1 THEN
  statement1;                         |      statement1;
ELSE                                  |    ELSIF condition2 THEN
  IF condition2 THEN                  |      statement2;
    statement2;                       |    ELSIF condition3 THEN
  ELSE                                |      statement3;
    IF condition3 THEN                |    END IF;
      statement3;                     |
    END IF;                           |
  END IF;                             |
END IF;                               |
```

SUM OF TWO NUMBERS

Aim: To write a pl/sql program to find sum of two numbers.

Procedure:

```
Declare
     i number(3);
     j number(3);
     k number(3);
Begin
     i:=&i;
     j:=&j;
     k:=i+j;
    Dbms_output.put_line('The sum of '||i||','||j||'is:'||k);
end; /
```

Execution:

```
SQL> @e:\sqlpl\add.sql
Enter value for i: 4
old   6:  i:=&i;
new   6:  i:=4;
Enter value for j: 4
old   7:  j:=&j;
new   7:  j:=4;
the sum of 4,4 is:8

PL/SQL procedure successfully completed.
```

Conclusion:

A pl/sql program is successfully executed for finding sum of two numbers.

EVEN OR ODD

Aim: To write a pl/sql program to find given number is even or odd.

Procedure:

```
Declare
    n number(4):=&n;
 Begin
    if mod(n,2)=0
   then
     dbms_output.put_line(n||' even number');
   else
     dbms_output.put_line(n||' odd number');
  end if;
end;/
```

Execution:

```
SQL> @e:\plsql\even.sql
Enter value for n: 5
old   2:  n number(4):=&n;
new   2:  n number(4):=5;
5 odd number
PL/SQL procedure successfully completed.
```

Conclusion:

a pl/sql program is successfully executed for finding even or odd.

MULTIPLES OF 5

Aim: To write a pl/sql program for finding Multiples of 5.

Procedure:

```
declare
    i number(3):=5;
    n number(3):=&n;
Begin
    Dbms_output.put_line('The multiples of 5 are:');
    while i<=n
     loop
       dbms_output.put_line(i);
       i:=i+5;
    end loop;
 end;/
```

Output:

```
SQL> @e:\sqlpl\2.sql
Enter value for n: 20
old   3:  n number(3):=&n;
new   3:  n number(3):=20;
the multiplus of 5 are:
5          10         15       20
PL/SQL procedure successfully completed.
```

Conclusion:

A pl/sql program is successfully executed for finding Multiples of 5.

MULTIPLICATION TABLES

Aim: To write a pl/sql program for display the Multiplication Tables up to given number.

Procedure:

```
 Declare
     i number(4);
     j number(3);
    n number(3):=&n;
  Begin
   for i in 1..n
    loop
        dbms_output.put_line(i||' Table');
        dbms_output.put_line('----------');
        for j in 1..10
         loop
           dbms_output.put_line(i||'*'||j||'='||n*j);
          end loop;
    end loop;
end;/
```

Execution:

```
SQL> @e:\plsql\3.sql
Enter value for n: 2
old   4:   n number(3):=&n;
new   4:   n number(3):=2;
1 Table   and 2 Table displayed
----------
PL/SQL procedure successfully completed.
```

Conclusion: A pl/sql program is successfully executed for Multiplication Tables.

PRIME NUMBER

Aim: To write a pl/sql program to check weather given number is Prime or not.

Procedure:

```
declare
    num number;
    i number:=1;
    c number:=0;
begin
     num:=&num;
    for i in 1..num
    loop
      if((mod(num,i))=0)
       then
          c:=c+1;
      end if;
   end loop;
  if(c>2)
  then
     dbms_output.put_line(num||' not a prime');
  else
    dbms_output.put_line(num||' is prime');
  end if;
end;/
```

Execution:

```
SQL> @e:\plsql\prime.sql
Enter value for num: 5
old   6:  num:=&num;
new   6:  num:=5;
5 is prime
```

Conclusion:

A pl/sql program is successfully executed to check the given number is prime or not.

FACTORIAL OF A NUMBER.

Aim: To write a pl/sql program to finding factorial of given number.

Procedure:

```
declare
    i number(4):=1;
    n number(4):=&n;
    f number(4):=1;
begin
  for i in 1..n
  loop
    f:=f*i;
   end loop;
  Dbms_output.put_line('the factorial of '||n||' is:'||f);
  End; /
```

Output:

```
SQL> @e:\plsql\fact.sql
Enter value for n: 5
old   3:   n number(4):=&n;
new   3:   n number(4):=5;
the factorial of 5 is:120
PL/SQL procedure successfully completed.
```

Conclusion:

a pl/sql program is successfully executed for finding factorial of a given number.

Reverse of a Number

Aim: To write a pl/sql program to generate reverse for given number.

Procedure:

```
declare
   n number(4):=&n;
   s number(4):=0;
```

```
    r number(4);
 begin
    while n>0
    loop
     r:=mod(n,10);
     s:=(s*10)+r;
     n:=trunc(n/10);
    end loop;
  dbms_output.put_line('the reverse number is:');
  dbms_output.put_line(s);
end;/
```

Execution:

```
SQL> @e:\plsql\rev.sql
Enter value for n: 457
old   2:   n number(4):=&n;
new   2:   n number(4):=457;

the reverse number is:
754
PL/SQL procedure successfully completed.
```

Conclusion:a pl/sql program is successfully executed to generate reverse number for given number.

FIBINOCCI SERIES

Aim: To write a pl/sql program to generate fibinocci series.

Procedure:

```
  declare
     a number(3):=1;
     b number(3):=1;
     c number(3);
     n number(3):=&n;
begin
 Dbms_output.put_line('the fibinocci series is:');
 while a<=n
  loop
      dbms_output.put_line(a);
      c:=a+b;
      a:=b;
      b:=c;
   end loop;
 end;/
```

Execution:

```
SQL> @e:\plsql\fibi.sql
Enter value for n: 13
old   5:   n number(3):=&n;
new   5:   n number(3):=13;
the fibinocci series is:
1
1
2
3
5
8
13
PL/SQL procedure successfully completed.
```

Conclusion: a pl/sql program is successfully executed for to generate fibinocci series.

INSERTING A ROW USING PL/SQL

Aim: To write a pl/sql program for inserting a row into vender table.

Procedure:

1.Creating vender table.

```
SQL> create table vendor(
  2  v_code number(5) primary key,
  3  v_name varchar(10),
  4  v_contact varchar(10),
  5  v_area varchar(10),
  6  v_phone number(10));
Table created.
```

2. Inserting row using pl/sql

```
SQL> declare
     begin
        insert into vendor values(1012,'bajaj','delhi',121,9949481807);
     end;/
         PL/SQL procedure successfully completed.
```

3. Displaying the row

```
SQL> select *From vendor;
   V_CODE    V_NAME     V_CONTACT     V_AREA      V_PHONE
--------- ----   -------------   -------------- ---   ------------    --------------
       1012          bajaj             delhi             121     9985001199
```

Conclusion:

a pl/sql program is successfully executed for inserting a row into vender table.

USING %TYPE

Aim: To write a pl/sql program to display the employee details using %type data type.

Procedure:

1.creating emp table:

```
SQL> create table emp(
  2  eno number(4) primary key,
  3  ename varchar(8),
  4  job varchar(8));
           Table created.
```

2.Inserting values into emp table:

```
SQL> insert into emp values(101,'kalam','clerk');
1 row created.
SQL> insert into emp values(100,'john','sales');
1 row created.
SQL> insert into emp values(102,'priya','clerk');
1 row created.
```

3. Execution:

```
           SQL> declare
     2  Eno emp.eno%type;
     3  Ename emp.ename%type;
     4  begin
     5  select eno,ename into Eno,Ename
     6  from emp
     7  where eno=100;
     8  dbms_output.put_line('eno:'||Eno);
     9  dbms_output.put_line('ename:'||Ename);
    10  end;
    11  /
       eno:100
    ename:john              PL/SQL procedure successfully completed.
```

Conclusion: a pl/sql program is successfully executed to display the employee details using %type data type.

USING %ROWTYPE

Aim: To write a pl/sql program to display the employee details using %rowtype data type.

Procedure:

Execution:

```
SQL> declare
```

```
  2  E emp%rowtype;
  3  begin
  4  select * into E
  5  from emp
  6  where eno=100;
  7  dbms_output.put_line('eno:'||E.eno);
  8  dbms_output.put_line('ename:'||E.ename);
  9  end;
 10  /
eno:100
ename:john
PL/SQL procedure successfully completed.
```

Conclusion: a pl/sql program is successfully executed to display the employee details using %type data type.

CALCULATE STUDENT GRADE

Aim: To Write a pl/sql program to calculate the student grade using case statement.

Procedure:

Display ing student details:

```
SQL>select *From student where sno=101;
  Sno   sname      group  m1  m2  m3   grade
 -----  ----------  ---------  ----  ----  ----   -------
 100   wasim    mpc       75    76   88       B
```

Source code:

```
declare
   Grd student.grade%type;
begin
  select grade into Grd from student
  where sno=100;
  dbms_output.put_line('Grade:');
 case Grd
    when 'A' then Dbms_output.Put_line('Very Good');
    when 'B' then Dbms_output.Put_line('Good');
    when 'C' then Dbms_output.Put_line('Avrage');
   else
    Dbms_output.Put_line('Fail');
  end case;
 end;
/
```

Output:

```
SQL> @e:\plsql\case.sql
Grade:
Good
```

Conclusion: a pl/sql program is successfully executed to calculate the student grade using case statement.

DISPLAYING EMPLOYEE DETAILS USING CURSORS.

Aim: To write a pl/sql program to displaying employee details using cursors.

Source code:

```
declare
```

```
  no emp.eno%type;
  name emp.ename%type;
  cursor emp_cur is select eno,ename from emp;
begin
  open emp_cur;
  loop
    fetch emp_cur into no,name;
    exit when emp_cur%notfound;
    DBMS_output.put_line('Employee No:'||no||'Employee Name:'||name);
  end loop;
  close emp_cur;
end;
/
```

Execution:

```
SQL> @e:\plsql\c1r.sql;
Employee No:101 Employee Name:kalam
Employee No:100 Employee Name:john
Employee No:102 Employee Name:priya
Employee No:103 Employee Name:raja
Employee No:104 Employee Name:giri
Employee No:105 Employee Name:kohili
Employee No:106 Employee Name:latha
Employee No:107 Employee Name:hari
PL/SQL procedure successfully completed.
```

Conclusion:

a pl/sql program is successfully executed for displaying employee details using cursors.

TOP 10 EMPLOYEES

Aim: To Write a pl/sql program to displaying top 10 employee details based on salary using cursors

Source code:

```
declare
    i number(2);
    E emp%rowtype;
    cursor ec is select * from emp  order by salary desc;
 begin
     Dbms_output.put_line('Eno Ename   Job  Salary');
     Dbms_output.put_line('----------------------');
    i:=1;
   open ec;
   loop
     fetch ec into E;
       i:=i+1;
      Dbms_output.put_line(E.eno||' '||rpad(E.ename,8,' ')||' '|| rpad(E.job,5,'
')||'  '||E.salary);
   exit when i>10 or ec%notfound;
  end loop;
close ec;
```

```
  end;
   /
```

Execution:

```
SQL> @e:\plsql\c2.sql;
Eno     Ename    Job     Salary
-------------------------------------
104     giri          gm      14000
106     latha        sales    12800
111     kiran         agm     12800
112     swathi       clerk    12600
113     wasim        agm      12400
107     hari         sales     6000
105     kohili       clerk     6000
108     lakshmi      clerk     5600
100     john          sales    4880
101     kalam        clerk     4800
PL/SQL procedure successfully completed.
```

Conclusion : a pl/sql program is successfully executed for program to displaying top 10 employees.

STUDENT MARK LIST

Aim: To write a pl/sql program to print mark list using cursors.

Procedure:

Creating table:

```
SQL>create table Student(
            sno  number(4),
             sname varchar(10),
             m1  numbar(3),
             m2 numbar(3),
             m3 numbar(3));
            table created.
SQL>insert into student values (500,'wasim',70,75,89);
1 row inserted.
SQL>insert into student values (500,'siva',75,69,88);
1 row inserted.
SQL>insert into student values (500,'vani',84,75,75);
1 row inserted.
SQL>insert into student values (500,'naga',67,50,33);
1 row inserted.
```

Source code:

```
declare
    stu student%rowtype;
    total number(4);
    result varchar(4);
    cursor c is select * From student;
begin
  for stu in c
    loop
        Dbms_output.put_line('STUDENT MARKLIST');
        Dbms_output.put_line('----------------------------');
        Dbms_output.put_line('sno:'||stu.sno||' sname:'||stu.sname);
        total:=stu.m1+stu.m2+stu.m3;
        if stu.M1>35 and stu.M2>35 and stu.M3>35
        then
```

Execution:

```
SQL> @e:\plsql\c3.sql
STUDENT MARKLIST
----------------------------
sno:500  sname:wasim
m1:70 m2:75 m3:89
total:234 result:pass

STUDENT MARKLIST
----------------------------
sno:501 sname:siva
m1:75 m2:69 m3:88
total:232 result:pass

STUDENT MARKLIST
----------------------------
sno:502 sname:vani
m1:84 m2:75 m3:75
total:234 result:pass

STUDENT MARKLIST
----------------------------
sno:504 sname:naga
m1:67 m2:50 m3:33
total:150 result:fail

      PL/SQL procedure successfully
completed.
```

```
          result:='pass';
        else
          result:='fail';
        end if;
         Dbms_output.put_line('m1:'||stu.M1||' m2:'||stu.M2||' m3:'||stu.M3);
         Dbms_output.put_line('total:'||total||' result:'||result);
      end loop;
end;
/
```

Conclusion:

a pl/sql program is successfully executed for displaying the student mark list using cursors.

PARAMETERIZED CURSOR

Aim: To write a pl/sql program to display employees using parameterized cursor.

Source code:

```
            declare
     n number;
      er emp%rowtype;
     cursor c(d number) is
     select * from emp where emp.eno=d;
begin
    n:=&n;
    open c(n);
    if c%isopen then
      loop
          fetch c into er;
          exit when c%notfound;
             Dbms_output.put_line('Employee No:'||er.eno
              ||'Employee Name:'||er.ename);
          end loop;
      else
       DBMS_OUTPUT.PUT_LINE('not found');
      end if;
    close c;
end;
  /
```

Execution:

```
SQL> @e:\plsql\c4.sql
Enter value for n: 100
old   7:  n:=&n;
new   7:  n:=100;
Employee No:100  Employee
Name:mahesh

       PL/SQL procedure successfully
completed.
```

Conclusion:

a pl/sql program is successfully executed for displaying employees using parameterized cursor

INCREMENTING SALARY OF EMPLOYEES.

Aim: To Write a pl/sql program to update the commission values for all employees with salary less than 2000 by adding Rs.1000 to existing employees.

Source code:

```
 declare
```

```
      cursor c is select *From emp for update;
       emp_rec emp%rowtype;
begin
    for emp_rec in c
    loop
        if emp_rec.salary<2000 then
          update emp set comm=comm+1000
          where current of c;
         end if;
       end loop;
  end;
/
```

Displaying rows before execution:

```
   SQL>Select *From employee;
        Eno  Ename  Job   Salary comm
-------------------------------------------------
106     latha       sales     12800     5000
111     kiran        agm      12800     6000
100     john         sales     1880      500
101     kalam       clerk      1800      600
105     kohili      clerk      1000      200
```

Execution:

```
       SQL>@E:\plsql\incr.sql ;
         PL/SQL procedure successfully completed.
```

Displaying rows after execution:

```
   SQL>Select *From employee;
        Eno  Ename  Job   Salary comm
-------------------------------------------------
106     latha       sales     12800     5000
111     kiran        agm      12800     6000
100     john         sales     1880     1500
101     kalam       clerk      1800     1600
105     kohili      clerk      1000     1200
```

Conclusion:

a pl/sql program is successfully executed for incrementing salary of employees.

****

All the best