CHAPTER 2

UNIT II

SQL and PL/SQL

Syllabus

- **SQL :** Characteristics and advantages
- SQL Data Types and Literals
- DDL, DML, DCL, TCL
- SQL Operators
- Tables : Creating, Modifying, Deleting
- Views : Creating, Dropping, Updating using Views
- Indexes
- SQL DML Queries: SELECT Query and clauses
- Set Operations
- Predicates and Joins
- Set membership
- Tuple Variables
- Set comparison
- Ordering of Tuples,
- Aggregate Functions
- Nested Queries
- Database Modification using SQL Insert, Update and Delete Queries.
- **PL/SQL:** concept of Stored Procedures & Functions
- Cursors, Triggers
- Assertions
- Roles and privileges
- Embedded SQL
- Dynamic SQL

Syllabus Topic : SQL

2.1 SQL - Characteristics and Advantages

- SQL stands for Structured Query Language. SQL is used to communicate with a database. It is the standard language for relational database management systems. SQL statements are used to perform different operations on database like retrieval, insertion, updation and deletion of data.
- SQL is used in various advanced Relational Database Management Systems (RDBMS). Some common RDBMS that use SQL are : MySQL, Oracle, Microsoft Access, Microsoft SQL Server, Sybase, Ingres, etc.
- SQL is developed by IBM as a part of System R project in 1970. Initially it was called as Sequel. SQL was one of the first commercial languages for Edgar F. Codd's relational model. SQL became a standard of the American National Standards Institute (ANSI) in 1986, SQL is a declarative language in which the desired result is given without the specific details about how to accomplish the task.
- The steps required to execute SQL statements are handled transparently by the SQL database. SQL can be characterized as non-procedural because in procedural languages the details of the operations to be specified, such as opening and closing tables, loading and searching indexes, or flushing buffers and writing data to file systems are required which is not necessary in SQL.

Syllabus Topic : Characteristics of SQL

2.1.1 Characteristics of SQL

- SQL is an ANSI and ISO standard computer language for creating and manipulating databases.
- SQL allows the user to create, update, delete, and retrieve data from a database.
- The tokens and syntax of SQL are oriented from English common speech to keep the access barrier as small as possible. Hence it is very simple and easy to learn.
- All the keywords of SQL can be expressed in any combination of upper and lower case characters. It makes no difference whether UPDATE, update, Update, UpDate i.e. the keywords are case insensitive.
- SQL is a declarative language, not a procedural one.
- SQL is very powerful language.
- SQL works with database programs like DB2, Oracle, MS Access, Sybase, MS SQL Sever etc.

Syllabus Topic : Advantages of SQL

2.1.2 Advantages of SQL

There are various advantages of SQL

- **High Speed :** SQL Queries can be used to retrieve large amounts of records quickly and efficiently from a database.
- **Portable :** SQL can be run on any platform. Also it can be executed on PCs, laptops, servers and even mobile phones. It runs in local systems, intranet and internet. Databases using SQL can be moved from a device to another without any problems.
- **Well Defined Standards Exist :** SQL databases use long-established standard, which is being adopted by ANSI & ISO. Whereas Non-SQL databases do not adhere to any clear standard.
- **Supports object based programming :** SQL supports various object oriented programming concepts which makes it powerful.
- **Used with all DBMS systems with any vendor :** SQL is used by all the vendors who develop DBMS.
- **No Coding Required :** Using standard SQL it is easier to manage database systems without writing large amount of code.
- **Used for relational databases :** SQL is widely used for number of relational databases.
- **Easy to learn and understand :** SQL mainly consists of English words and hence it is easy to learn and understand the SQL queries.
- **Complete language for a database :** SQL is used to create databases and manage the databases in all aspects.
- **Dynamic database language :** SQL can change the database dynamically at runtime even while the database is being used by users.
- **Can be used as programming and interactive language :** SQL can do both the jobs of being a programming as well as an interactive language at the same time.
- **Client/Server language :** SQL can be used in client server architecture as a mediator between client application and server database.
- **Multiple data views :** We can provide different views(presentations) of contents of a database to different users.
- **Used in internet :** SQL can be used in internet to access the web related data.

Syllabus Topic : SQL Data Types

2.2 SQL Data Types and Literals

2.2.1 SQL Data Types

In SQL, we store the data in tabular format where table (relation) is the combination of rows (tuples) and columns (fields). While creating table we have to assign data types to the columns. These data types are used to decide that which type of data the columns can store.

There are various types of data types in SQL to store different types of data items.

rows - tuples
columns - fields

1. CHAR fixed length
2. VARCHAR Variable length
3. BOOLEAN True OR false.
4. SMALLINT $2^{15} \rightarrow 2^{15}-1$ (After decimal truncated)
5. INTEGER or INT $2^{30} \rightarrow 2^{31}-1$)
6. DECIMAL [(p[,s])] or DEC [(p[,s])] float
7. NUMERIC [(p,[s])] float. (Same)
8. REAL
9. FLOAT(p) p-64 default
10. DATE 'yyyy-MM-DD'
11. TIME 'HH-MM-SS'
12. TIMESTAMP
13. CLOB [(length)] or CHARACTER LARGE OBJECT [(length)] or CHAR LARGE OBJECT [(length)]
14. BLOB [(length)] or BINARY LARGE OBJECT [(length)]

(1) CHAR (length)

- The CHAR data type accepts character OR string type of data including Unicode. It is known as fixed length data type. The length of the character string is specified while assigning the data type. For example, CHARACTER(n) where n represents the maximum size of the character string. If size is not specified then the default size will be 1.
- The minimum length of the CHARACTER data type is 1 and maximum length is up to the table page size. Character strings which are larger than the page size of the table can be stored as a Character Large Object (CLOB).
- If value having lower size than the size of CHAR data type is stored in it, then the remaining space is filled with blanks characters. That means it gets wasted.
- If value having greater size than the size of CHAR data type is tried to store, then the extra characters are truncated.

Examples :

CHAR(20) or CHARACTER(20)

'Phoenix', 'INFOTECH'

(2) VARCHAR (length)

- The VARCHAR data type accepts character OR string type of data including Unicode. It is known as variable length data type.
- The length of the character string is specified while assigning the data type which indicates the maximum number of characters it can accept.
- We can assign the length from 1 to the current table page size.
- If value having lower size than the size of VARCHAR data type is stored in it, then the remaining space will get reutilized. That means the memory does not get wasted.
- If you need to store character strings that are longer than the current table page size, the Character Large Object (CLOB) data type should be used.

Examples : VARCHAR(10)

'Phoenix', 'INFOTECH'

(3) BOOLEAN

- The BOOLEAN data type ~~can~~ accepts value either TRUE or FALSE. No need to declare size while declaring the BOOLEAN data type.
- TRUE or FALSE are case insensitive. If you attempt to assign any other value to a BOOLEAN data type, an error gets raised.

Examples : TRUE, true, True, False

(4) SMALLINT $2^{15} \rightarrow 2^{15}-1$

- The SMALLINT data type is used to accept numeric values with default scale as zero. It stores any integer value between the range 2^ -15 and 2^15 -1. Attempting to assign values outside this range causes an error.
- If you assign a numeric value with a precision and scale to a SMALLINT data type, the scale portion truncates, without rounding.

Examples : SMALLINT

-32768, 0, –13.7 (digits to the right of the decimal point are truncated), 32767

(5) INTEGER or INT

- The INTEGER data type is used to accept numeric values with a default scale as zero. It stores any integer value between the range 2^ -31 and 2^31 -1. Attempting to assign values outside this range causes an error.
- If you assign a numeric value with a precision and scale to an INTEGER data type, the scale portion truncates, without rounding.

Examples

– 345, 0, 4532.98 (digits to the right of the decimal point are truncated), 167

(6) DECIMAL [(p[,s])] or DEC [(p[,s])]

- The DECIMAL data type is used to accept floating point values for which you define a precision and a scale in the data type declaration. The precision is a positive integer that represents the total number of digits that the number will contain (precision + scale).
- The scale is a positive integer that represents the number of digits of decimal places which will occur to the right of the decimal point. The scale for a DECIMAL cannot be larger than the precision. [Scale ≤ precision] IMP.
- If you exceed the number of digits expected to the left of the decimal point, an error is thrown. If you exceed the number of expected digits to the right of the decimal point, the extra digits are truncated.

Examples : DECIMAL (10, 3)

98789, 765.123, 10.1234(Final digit is truncated), -987, -897.786, -1234567.1234 (Final digit is truncated)

(7) NUMERIC [(p[,s])]

It is same as of Decimal

(8) FLOAT (p)

The FLOAT data type accepts approximate numeric values, for which you may define a precision up to a maximum of 64. The default precision is 64 if not declared.

Examples : FLOAT(8)

12345678, 1.2, 123.45678, -12345678, -1.2, -123.45678

(9) DATE

- The DATE data type accepts date type of values. No need to assign size while declaring a DATE data type. Date values should be specified in the form: YYYY-MM-DD.
- The value of month must be between 1 and 12, value of day should be between 1 and 31 depending on the month and value of year should be between 0 and 9999. The values should be enclosed in single quotes, preceded by the keyword DATE.

Examples : DATE '1999-01-01'
DATE '2000-2-2'

(10) TIME

- The TIME data type accepts time values. No parameters are required when declaring a TIME data type. The format is : HH:MM:SS. The fractional value can be used to represent nanoseconds.
- The minutes and seconds values must be two digits. Hour values should be between zero 0 and 23, minute values should be between 00 and 59 and second values should be between 00 and 61.999999.
- Values assigned to the TIME data type should be enclosed in single quotes, preceded by keyword TIME.

Examples : TIME '1:10:20'

Syllabus Topic : SQL Literals

2.2.2 SQL Literals

The literal is the constant or fixed data value. For example 'Phoenix', 'Institute' are string literals while 100, 5 are numerical literals and so on. The String, date and time literals are always enclosed in single quotation marks while the numerical literals are without quotation marks. Literal are case sensitive.

SQL Supports following types of literals.

(1) Numeric Literals
(2) Character Literals
(3) String Literals

(4) Date Literals
(5) Time Literals

(1) Numeric Literals

Numeric literals are the sequence of digits proceeded by an optional sign (+ / –) and with an optional decimal point. The Numeric Literals are further classified into two categories :

Integer Literals : These are the whole numbers assigned as data values. An integer can store a maximum of 38 digits of precision.

For example : 100, + 7, – 8 etc.

Number or Floating Point Literals : These are the numbers with decimal point assigned as data values. For example : 10.2, +7.3, -8.2 etc.

(2) Character Literals

Character literal contains single character enclosed in single quotation marks.

For example : 'A', '%', '9', 'z', '('

(3) String Literals

- These are the sequence of characters enclosed in single quotes. In SQL versions up to 7.0, the maximum length of a string literal is 1024. From version 7.1, there is no specific maximum limit on number of characters.
- The character string literals cannot be enclosed in double quotes because it is reserved for delimiting identifiers such as field or table names.

For example : 'Phoenix', 'I' etc.

(4) Date Literals

- These literal are the date type of values in the ANSI date format 'YYYY-MM-DD' or the default date format specified in the application via the 'set date format' operation.
- The date literal is enclosed in single quotation marks.

For Example : '2017-03-18'

(5) Time Literals

- These literal are the Time type of values in the format 'HH:MM' or 'HH:MM:SS'. In addition 'am' or 'pm' can be included at the end for 12-hour time format.
- If the am/pm indicator is excluded, it is assumed that the time is in 24-hour format.
- The date literal is enclosed in single quotation marks.

'11:15', '8:30 am', '06:25:15 pm', '17:00'

Syllabus Topic : DDL, DML, DCL, TCL

2.3 DDL, DML, DCL, TCL

SPPU - May 13, May 14

University Questions

Q. Explain various database Languages. **(May 2013, 8 Marks)**

Q. Write short note on DDL, DML and DCL. **(May 2014, 4 Marks)**

The database languages are categorized as DDL, DML, DCL and TCL.

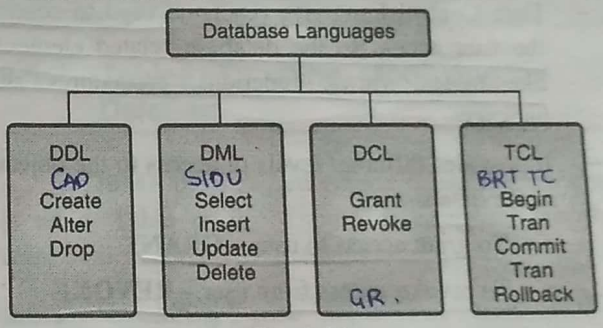

Fig. 2.3.1

2.3.1 Data Definition Language (DDL)

- This language allows the users to define data and their relationship to other types of data. It is used to create data tables, dictionaries, and files within databases.
- The DDL is also used to specify the structure of each table, set of associated values with each attribute, integrity constraints, security and authorization information for all the tables and physical storage structure of all the tables on the disk.
- Let's take SQL for instance to categorize the statements that comes under DDL.
  - To create the database instance – **CREATE**
  - To alter the structure of database – **ALTER**
  - To drop database instances – **DROP**
  - To rename database instances – **RENAME**

DDL | DML | DCL | TCL
Create Alter Rename Drop | Select Insert Delete Update | Grant Revoke

2.3.2 Data Manipulation Language (DML)

- The Data Manipulation Language (DML) is used for accessing and manipulating data in a database. DML provides a set of functionalities to support the basic data manipulation operations on the data stored in the database.
- It allows users to access, insert, update, and delete data from the database.
  - To access or read records from table – **SELECT**
  - To insert record into the table – **INSERT**
  - Update the records in table – **UPDATE**
  - Delete the records from the table – **DELETE**

2.3.3 Data Control Language (DCL)

- Data Control Language (DCL) is used to control the user access to the database related elements like tables, views, functions, procedures and packages.
- It provides different levels of access to the objects in the database.
  - To grant access to user – **GRANT**
  - To revoke access from user – **REVOKE**

Grant : GRANT is used to provide the privileges to the users on the database objects. The privileges could be select, delete, update and insert on the tables and views. On the procedures, functions and packages it gives select and execute privileges.

Revoke : REVOKE removes the privileges given on the database objects. All the privileges can be removed at a time or one or more privileges can also be removed from the objects as per requirement.

2.3.4 Transaction Control Language (TCL)

TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

- **BEGIN Transaction** – opens a transaction
- **COMMIT Transaction** – commits (Save permanently) transactions
- **ROLLBACK Transaction** – ROLLBACK (Cancels. undo) transactions in case of any issue

Syllabus Topic : SQL Operators

2.4 SQL Operators

- An operator is a character or reserved word used in SQL statements to perform different operations like arithmetic or comparison.
- Operators are used to specify conditions in an SQL statement and also used to integrate multiple conditions in SQL statement.
  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Operators used to negate conditions

2.4.1 Arithmetic Operators

Consider two operands x and y where value of x is 10 while y is 5.

Operators	Descriptions	Examples
+ (Add)	It adds the operands	x + y = 15
- (Subtract)	It subtracts right hand operand from left hand operand	x – y = 5
* (Multiply)	It multiply both operands	x * y = 50
/ (Divide)	It divides left hand operand by right hand operand	x / y = 2
% (Modulo)	It divides left hand operand by right hand operand and returns reminder	x % y = 0

2.4.2 Comparison Operator

Consider two operands x and y where value of x is 10 while y is 5.

Operators	Descriptions	Examples
=	Check whether both the operands have same values, if yes condition becomes true.	x=y is not true
>	Check whether left operand is greater than right, if yes condition becomes true	x>y is true
<	Check whether left operand is less than right, if yes condition becomes true	x<y is not true

Operators	Descriptions	Examples
>=	Check whether left operand is greater than or equal to the right operand or not, if yes condition become true	x>=y is true
<=	Check whether left operand is less than or equal to the right operand or not, if yes condition become true	x<=y is not true
!=	Check whether both the operands have same values or not, if not condition become true.	x != y is true
<>	Check whether both the operands have same values or not, if not condition become true.	x<>y is true
!>	Check whether left operand is not greater than the value of right operand	x!>=y is not true
!<	Check whether left operand is not less than the right operand value	x!<=y is true

2.4.3 Logical Operators

Operator	Description
ALL	TRUE if all of a set of comparisons are TRUE.
AND	TRUE if both Boolean expressions are TRUE.
ANY	TRUE if any one of a set of comparisons is TRUE.
BETWEEN	TRUE if the operand is within a range.
EXISTS	TRUE if a sub-query contains any rows.
IN	TRUE if the operand is equal to one of a list of expressions.
LIKE	TRUE if the operand matches a pattern.
NOT	Reverses the value of any other Boolean operator.
OR	TRUE if either Boolean expression is TRUE.
SOME	TRUE if some of a set of comparisons are TRUE.

2.4.4 Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise excusive OR

2.4.5 Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

Syllabus Topic : Tables - Creating, Modifying, Deleting

Note : All the queries are implemented considering MySQL.

2.5 Tables : Creating, Modifying, Deleting

In DBMS the standard format of storing the data is table. Table is also known as relation. It is the combination of rows and columns.

2.5.1 Creating Table

The **CREATE TABLE** statement is used to create table in database.

Syntax

```
CREATE TABLE table_name (
   column1 datatype[size],
   column2 datatype [size],
   column3 datatype[size],
   ....
);
```

In the CREATE TABLE statement the column parameters specify the names of the columns or fields of the table. The data type is the type of data which we want to store in the respective fields. The fields can hold data of different types like char, varchar, number, date etc.

The optional size value can also be mentioned after the data type. This size value indicated the maximum length of data for the field. If size is not given then default value depending upon the data type is assigned.

Example

Student Table

```
CREATE Table student
(roll_no integer(3), stud_name varchar(20),
bdate date , marks integer(3));
```

In the above example, a table student will be created with following attributes. The roll_no field will contain numerical value of maximum digit 3. The stud_name field will contain string value of maximum length 20. The date field will contain date type value of standard date length. For date data type no need to mention size. The marks field will contain numerical value of maximum digit 3.

The structure of the table will be as follows after record insertion.

Table 2.5.1 : Student

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

Employee Table

```
CREATE Table employee
(emp_id integer(3), emp_name varchar(20),
Salary integer(7), department varchar(20));
```

Product Table

```
CREATE Table product
(prod_code integer(3), prod_name varchar(20),
price integer(7), category varchar(20));
```

Customer Table

```
CREATE Table customer
(cust_id integer(3), cust_name varchar(20),
address varchar(20), email_id varchar(20));
```

2.5.1.1 Creating New Table from Existing Table

AS SELECT clause is used to create table from existing table. It can be considered as copy of existing table. While creating such table, we have option whether to take all records, fields from existing table or not. We can copy just structure of the existing table also. The different ways of coping table are as given below :

Consider the existing table student as shown in Table 2.5.1. Creating new table same as of existing table.

Syntax

```
Create table table_name
as select * from existing_table_name;
```

For Example

```
Create table newstudent1
as select * from student ;
```

Output : The newly created table will be

Table 2.5.2 : Newstudent1

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

Creating new table having specific fields but all the records from existing table.

Syntax

```
Create table table_name
as select field_1,field_2... from
existing_table_name;
```

For Example

```
Create table newstudent2
as select roll_no,stud_name from student;
```

Output : The newly created table will be

Table 2.5.3 : Newstudent2

roll_no	stud_name
101	Kunal
102	Jay
103	Radhika
104	Sagar
105	Supriya

Creating new table having specific records but all the fields from existing table.

Syntax

```
Create table table_name
as select * from existing_table_name
where condition
```

For Example

```
Create table newstudent3
as select * from student
where marks > 80;
```

The newly created table will be structure wise same as of existing table, but it will contain records of only those students who got marks above 80.

Output

Table 2.5.4 : Newstudent3

roll_no	stud_name	Bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

Creating new table having no records but all the fields from existing table. That means copying only structure of existing table.

Syntax

```
Create table table_name
 as select * from existing_table_name
where false condition
```

For Example

```
Create table newstudent4
as select * from student
where 1=2;
```

Here 1=2 is the false condition. The newly created table will be structure wise same as of existing table, but it will no records get copied.

Output

Table 2.5.5 : Newstudent4

roll_no	stud_name	bdate	marks

2.5.2 Modifying Table

ALTER TABLE query is used to modify structure of a table. We can add, delete or modify column.

Adding New Column in a Table

```
ALTER TABLE table_name
ADD column_name datatype;
```

For Example : Adding column grade in the table student.

```
ALTER TABLE student
ADD Grade varchar(2);
```

Dropping Column from Table

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

For Example : Deleting column grade from the table student.

```
ALTER TABLE student
DROP column Grade;
```

Modifying Column of a Table

```
ALTER TABLE table_name
MODIFY COLUMN column_name data_type;
```

For Example : Changing the data type and size of column roll_number of student table.

```
ALTER TABLE student
modify column roll_no varchar(4);
```

Here we are changing the.

Deleting all the records from Table

Syntax

```
TRUNCATE TABLE table_name;
```

For Example : Deleting all the records from newstudent1

```
TRUNCATE TABLE newstudent1;
```

2.5.3 Deleting Table

DROP TABLE query is used to delete table permanently from the database.

Syntax

```
drop table table_name;
```

For Example : Deleting the newstudent1 table from the database.

```
Drop table newstudent1;
```

Syllabus Topic : Views - Creating, Dropping, Updating View

2.6 View : Creating, Dropping, Updating View

SPPU - May 15

University Question

Q. Explain view objects in SQL with example. **(May 2015, 3 Marks)**

View : In SQL, a view is a virtual table containing the records of one or more tables based on SQL statement executed. Just like a real table, view contains rows and columns. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table. The changes made in a table get automatically reflected in original table and vice versa.

Purpose of View : View is very useful in maintaining the security of database. Consider a base table employee having following data.

Emp_id	emp_name	Salary	Address
E1	Kunal	8000	Camp
E2	Jay	7000	Tilak Road
E3	Radhika	9000	Somwar Peth
E4	Sagar	7800	Warje
E5	Supriya	6700	LS Road

1. Now just consider we want to give this table to any user but don't want to show him salaries of all the employees. In that we can create view from this table which will contain only the part of base table which we wish to show to the user.

   See the following View.

Emp_id	emp_name	Address
E1	Kunal	Camp
E2	Jay	Tilak Road
E3	Radhika	Somwar Peth
E4	Sagar	Warje
E5	Supriya	LS Road

2. Also in multiuser system, it may be possible that more than one user may want to update the data of same table. Consider two users A and B want to update the employee table. In such case we can give views to both these users. These users will make changes in their respective views, and the respective changes are done in the base table automatically.

2.6.1 Creating View

Consider existing table student

Table 2.6.1 : Student

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

1. Creating view having all records and fiel from existing table

Syntax

```
CREATE or replace VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

For Example : Creating a view of base table stud with same structure and all the records.

```
Create or replace view stud_view1
as select * from student;
```

Output

Table 2.6.2 : stud_view1

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

2. Creating view having specific fields but the records from existing table

Syntax

```
Create or replace view view_name
as select field_1,field_2...
from existing_table_name;
```

For Example

```
Create or replace view stud_view2
as select roll_no, name from student;
```

Output : The newly created view will be

Ta

r

3. Creating ne but all the fi

Syntax

```
Create or replace
as select * from
where condition;
```

For Example

```
Create or replace
as select * from
where marks > 8
```

Output

Ta

roll_no	stu
101	
103	R

2.6.2 Updatin

Update query view. Updation in Means the same c table also.

Syntax

```
UPDATE view_na
set field_name =
where condition
```

For Example : student having ro

```
UPDATE stud_vi
set marks = 73
where roll_no=1
```

In this case in both view viev

Output : View

roll_no	st
101	K
102	Ja

Table 2.6.3 : stud_view2

roll_no	stud_name
101	Kunal
102	Jay
103	Radhika
104	Sagar
105	Supriya

3. Creating new view having specific records but all the fields from existing table

Syntax

```
Create or replace view view_name
as select * from existing_table_name
where condition;
```

For Example

```
Create or replace view sud_view3
as select * from student
where marks > 80;
```

Output

Table 2.6.4 : stud_view3

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

2.6.2 Updating View

Update query is used to update the records of view. Updation in view reflects the original table also. Means the same changes will be made in the original table also.

Syntax

```
UPDATE view_name
set field_name = new_value;
where condition;
```

For Example : We are updating marks to 73 of student having roll_no 102.

```
UPDATE stud_view1
set marks = 73
where roll_no=102;
```

In this case marks of roll_no 102 will get updated in both view view1 as well as table student

Output : View - View1

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	**73**
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

Output

Table 2.6.5 : Student

roll_no	stud_name	bdate	Marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	**73**
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

There are some restrictions on the modification with respect to view.

- In case of view containing joins between multiple tables, only insertion and updation in the view is allowed, deletion is not allowed
- Data modification is not allowed in the view which is based on union queries.
- Data modification is not allowed in the view where GROUP BY or DISTINCT statements are used.
- In view the text and image columns can't be modified.

2.6.3 Dropping View

DROP query is used to delete a view.

Syntax

```
DROP view view_name;
```

For Example

```
DROP view stud_view2;
```

Syllabus Topic : Indexes

2.7 Indexes

SPPU - May 15

University Question

Q. Explain Index objects in SQL with example. **(May 2015, 3 Marks)**

- Sometimes the data in the database is very large. For example in the application of State Bank of India, the database related to customers and their

transactions is very large. In such case retrieval of data from such huge database becomes slower.

- Indexes are the special lookup tables which are available to only database search engine for accessing data. Indexes speed up data retrieval effectively.
- An index is a pointer to data in a table. It is similar to the alphabetical index of a book present at the end of book. An index is used to speed up SELECT queries and also WHERE clauses.
- But because of indexes the data input related to INSERT and UPDATE statements get slow down.
- Indexes can be created or dropped with no effect on the data.

Creating Index

CREATE INDEX statement is used to create an index. In this statement we have to mention name of the index, the table and column, and whether the index is in ascending or descending order.

There are different types of indexes.

2.7.1 Single Column Index

This is index is created on a single column of a table.

Syntax

```
CREATE INDEX index_name
ON table_name (column_name)
```

For Example

```
CREATE INDEX ind1
on student(stud_name);
```

2.7.2 Composite Index

Sometimes duplicate records may available in columns. In such case the composite indexing is better option to index the data. This index is created on a multiple columns of a table.

Syntax

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

For Example

```
CREATE INDEX ind2 on student(stud_name , marks);
```

2.7.3 Unique Index

A unique index does not allow any duplicate values to be inserted into the table.

Syntax

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

For Example

```
CREATE UNIQUE INDEX ind3
on student(stud_name);
```

2.7.4 Implicit Index

Implicit indexes are indexes that are automatically created by the database server when an object is created. Such indexes are created for primary key and unique constraints.

Displaying Index : To display index information regarding table following query is used.

Syntax

```
Show index from table_name;
```

For Example

```
Show index from student;
```

Syllabus Topic : SQL DML Queries - Select Query and Clauses

2.8 SQL DML Queries - Select Query and Clauses

2.8.1 SELECT Query

SELECT query is used to retrieve the data from database. SELECT query never make any change in the database. The data returned by the SELECT query is in the form of result sets.

Syntax

```
SELECT column_1, column_1... from table_name;
```

For Example : Consider the table student

Table 2.8.1 : Student

roll_no	stud_name	bdate	Marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68

roll_no	stud_name	bdate	Marks
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

```
SELECT roll_no, stud_name from student;
```

Output

roll_no	stud_name
101	Kunal
102	Jay
103	Radhika
104	Sagar
105	Supriya

The names of columns specify data from which columns we want to display. If we want data from all the columns then no need to mention column names. '*' symbol represent all the columns.

For Example

```
SELECT * from student;
```

Output

roll_no	stud_name	bdate	Marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

With SELECT statement different clauses can be used to display the data as per our requirements.

2.8.2 WHERE Clause

WHERE clause is used to specify condition in SELECT statement while fetching records from the database. The WHERE clause filters the data to be retrieved. The records satisfying the condition given by where clause are retrieved.

Syntax

```
SELECT column_1,columns_2... from table_name
where condition;
```

For Example

```
select * from student where marks > 80;
```

Output

roll_no	stud_name	Bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

```
Select * from student where ename = 'Kunal';
```

Output

roll_no	stud_name	Bdate	marks
101	Kunal	12-02-2000	90

2.8.3 DISTINCT Clause

This clause is used to avoid selection of duplicate rows. Consider there are duplicate values in JOB column of emp table

Table 2.8.2 : Emp

Eno	Ename	Job	Sal
101	Susheel	Clerk	12000
102	Ajay	Manager	18000
103	Dinesh	Clerk	10000
104	Bharati	Manager	17000
105	Prajakta	Salesman	13000

Syntax

```
SELECT distinct( column_name) from table_name;
```

For Example

```
select distinct(job) from emp;
```

Output

Job
Clerk
Manager
Salesman

2.8.4 GROUP BY Clause

The GROUP BY clause is used in collaboration with the SELECT statement. It helps to arrange similar data into groups. It is also used with SQL functions to group the result from one or more tables.

Table 2.8.3 : Emp1

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

1. Display sum of salaries department wise.

```
Select deptno,sum(sal) from emp group by deptno;
```

Output

DEPTNO	SUM(SAL)
30	9400
20	10875
10	8750

2. Display maximum salaries groping on the basis of job

```
Select job,max(sal) from emp group by job;
```

Output

JOB	MAX(SAL)
CLERK	1300
SALESMAN	1600
PRESIDENT	5000
MANAGER	2975
ANALYST	3000

In general we use WHERE clause to give some condition to filter the data. But WHERE clause is not allowed in collaboration with GROUP BY clause HAVING clause is used with GROUP by clause t specify condition.

2.8.5 HAVING Clause

1. Display sum of salaries of department 10

```
Select deptno,sum(sal) from emp group by deptno
having deptno = 10;
```

Output

DEPTNO	SUM(SAL)
10	8750

2. Display sum of salaries of department 10 and 20

```
Select deptno,sum(sal) from emp group by deptno
having deptno in (10,20);
```

Output

DEPTNO	SUM(SAL)
20	10875
10	8750

Syllabus Topic : Database Modification using SQL Insert, Update and Delete Queries

2.9 Database Modification using SQL Insert, Update and Delete Queries

2.9.1 Insert

This query comes under the category Data Definition Language. After creation of table the insert command is used to insert one or more records in the table.

Insert query has different forms
(Consider Table 2.8.2 Emp)

(1) Inserting values in all columns

Syntax

```
Insert into table_name
values (value1, value2....)
```

For example

```
Insert into emp
values(106,'Rajesh','Clerk',12000);
```

(2) Inserting values in specific columns

Sometimes we may not have all values to insert into table. In such case the syntax will be

```
Insert into table_name(column1,column2...)
values(val1,val2...);
```

For example, consider we do not have value for salary while inserting a new record in emp table. Then the query will be

```
insert into emp(Eno,Ename,Job)
values(107,'Ankur','Salesman');
```

(3) Inserting records from existing table into new table

We can also take records from existing table to add into new table using as select clause. Consider new table emp1 in which we will add records from Table 2.9.1. Here we can mention condition using where clause to take specific records.

```
Insert into emp1
Select eno,ename,job,sal from emp
Where sal > 15000;
```

Output

Table 2.9.1 : Emp1

Eno	Ename	Job	Sal
102	Dinesh	Manager	18000
104	Bharati	Manager	17000

2.9.2 Update

Sometimes changes to the database become necessary. To make changes in the database 'update' command is used. Updations can be done in single or multiple columns based on the given condition. The update command consists of 'set' clause and an optional 'where' clause'

Syntax

```
Update table_name set column_name = new_value
[where condition]
```

For Example

```
Update emp set sal = 15000;
```

This query will make salary of all the employees to 15000.

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	15000
102	Dinesh	Manager	15000
103	Dinesh	Clerk	15000
104	Bharati	Manager	15000
105	Prajakta	Salesman	15000

'WHERE' clause is used to make changes in specific records.

For Example

```
Update emp set sal = 20000 where job = 'Manager';
```

This query will change salary of mangers to 20000.

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	15000
102	Dinesh	Manager	20000
103	Dinesh	Clerk	15000
104	Bharati	Manager	20000
105	Prajakta	Salesman	15000

2.9.3 Delete

As per requirement, the records from existing table can be removed using delete command. Delete command can have 'WHERE' clause optionally.

Syntax

```
Delete from table_name;
```

For Example

```
Delete from emp;
```

This query will delete all the records from Table 2.9.1.

```
Delete from emp where Eno = 103;
```

This query will delete the record of employee with employee number 103 from emp table.

Syllabus Topic : Set Operations

2.10 Set Operations

- Set operations are supported by SQL to be performed on table data. For these operations special operators knows as Set Operators are used. Set operators are used to join the results of multiple SELECT statements.
- These operators help to get meaningful results from data, under different specific conditions. Queries which contain set operators are called compound queries.
- The different Set Operators are as follows

(i) Union (ii) Union All
(iii) Intersect (iv) Minus

Consider following two tables Emp and Dept

Table 2.10.1 : Emp

Empno	Ename	Job	DeptNo	Salary
101	Rahul	Manager	10	17000
102	Vinay	Clerk	20	12000
103	Kunal	Manager	30	18000
104	Rajesh	Salesman	20	13000
105	Kushal	Clerk	10	11000

Table 2.10.2 : Dept

DeptNo	DeptName	Loc
10	Sales	Mumbai
20	Production	Pune
30	Accounts	Nasik
40	Research	Bangalore

2.10.1 Union

The union operator returns all distinct rows selected by either query

Syntax

```
Select column_name from table_1
Union
Select column_name from table_2
```

For Example

```
Select DeptNo from emp
union
select Deptno from dept
```

Output

10
20
30
40

2.10.2 Union All

The Union All operator returns all rows selected by either query including duplicates.

Syntax

```
Select column_name from table_1
Union all
Select column_name from table_2
```

For Example

```
Select DeptNo from emp
Union all
select Deptno from dept
```

Output

10
20
30
20
10
10
20
30
40

2.10.3 Intersect

The intersect operator returns only those rows which are common to both the queries

Syntax

```
Select column_name from table_1
intersect
Select column_name from table_2
```

For Example

```
Select DeptNo from emp
intersect
select Deptno from dept
```

Output

10
20
30

2.10.4 Minus

Minus operator displays the rows which are present in the first query but absent in the second query, with no duplicates and data is arranged in ascending order by default.

Syntax

```
Select column_name from table_1
minus
Select column_name from table_2
```

For Example

```
Select DeptNo from dept
intersect
select Deptno from emp
```

Output

40

Syllabus Topic : Predicates and Joins

2.11 Predicates and Joins

2.11.1 Predicates

Predicate is an expression that evaluates to TRUE, FALSE or UNKNOWN. Predicates are used in the search condition of WHERE and HAVING clauses, the join conditions of FROM clauses and other constructs where value in Boolean format is expected.

Consider the table

Table 2.11.1 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

SQL provides various types of predicates.

2.11.1.1 Comparison Predicate

Comparison predicate is the combination of two expressions separated by a comparison operator. There are six types of comparison operators: =, >, <, >=, <=, <>. The data of NUMERIC type is compared with respect to their algebraic values. The data of CHARACTER STRING type is compared with respect to their alphabetic order.

= Equal to Predicate

```
Select * from emp
where ename = 'KING'
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10

> Greater Than Predicate

```
Select * from emp
where sal > 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10

< Less Than Predicate

```
Select * from emp
where sal < 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

>= Greater Than Equal To Predicate

```
Select * from emp
where sal >= 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20

<= Less Than Equal To Predicate

```
Select * from emp
 where sal <= 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

<> Not Equal To Predicate

```
Select * from emp
where sal <> 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Combination of Predicates can be used with AND operator

```
Select * from emp
where sal >=3000 and sal <=5000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20

2.11.1.2 Between Predicate

Between predicate is used to specify certain range of values. The AND keyword is used in this predicate.

Syntax

```
test_expression [ NOT ] BETWEEN begin_expression AND end_expression
```

Example

```
Select * from emp
where comm between 300 and 500;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30

In "Between" predicate the NOT keyword can also be used

```
Select * from emp
where comm not between 300 and 500;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30

2.11.1.3 In Predicate

IN predicate particularly determines whether the value of expression given to test matches any value in specified the list.

Just consider that we want display records of employees from depno 10 and 20.

Then the query will be

```
Select * from emp
where deptno in(10,20)
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7934	MILLER	CLERK	7782	01/23/1982	1300		10

2.11.1.4 Like Predicate

Like operator determines whether a specific character string matches the given pattern or not. In the pattern we can use regular characters and wildcard characters. In this pattern matching, it is necessary that regular characters must exactly match the characters specified in the character string. However, for the wildcard characters arbitrary fragment matching of the character string is done. The use of wildcard characters makes the LIKE operator more flexible.

Example : Display records of employee whose names starts with letter 'J'

Query

```
select * from emp
where ename like 'J%';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7900	JAMES	CLERK	7698	12/03/1981	950		30

Example : Display records of employee whose names ends with letter 'N'

Query

```
select * from emp
where ename like '%N';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30

Example : Display records of employee whose names contains 'L' as second character.

Query

```
select * from emp
 where ename like '_L%';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30

Example : Display records of employee whose names contains character 'A' anywhere;

Query

```
select * from emp
where ename like '%A%';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30

2.11.1.5 IS [NOT] NULL

When values for some attributes are not available then, NULL value is assigned. To display records having NULL value, IS NULL predicate is used.

Example : Display records of employees who never get any commission.

```
select * from emp
where comm IS NULL;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

The NOT keyword can be used to get values opposite to given condition

Example : Display records of employees who get commission.

```
select * from emp
where comm IS NOT NULL;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30

2.11.2 Joins

SPPU - Dec. 13, May 14

University Questions

Q. What are different types of joins in SQL? Explain with suitable example. (Dec. 2013, 6 Marks)

Q. Explain different types of joins with example. (May 2014, 8 Marks)

- A JOIN is a means for combining columns from one (self-table) or more tables by using values common to each.
- There are following types of Joins.

1. INNER
2. OUTER
3. LEFT OUTER
4. RIGHT OUTER
5. FULL OUTER
6. SELF

To understand joins consider following two tables.

Table 2.11.2 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Table 2.11.3 : Dept

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- In the database having large amount of data, it is not possible to store the entire data in a single table. The related data may be stored in different tables. In above tables emp and dept the data of employees is stored.
- The EMP table contains the basic information of employee like employee number, name, job(post), salary, department number etc. The DEPT table contains the information of same employees about their depart names and locations.
- If we want to display whole information means basic information with department names and locations then we have to combine these two tables in query using joins.
- For using joins, we required a common column between both the tables. In this example the EMP and DEPT tables contains a common column DEPTNO.

2.11.2.1 Inner Join (Equi Join)

The INNER JOIN is used to display records that have matching values in both tables.

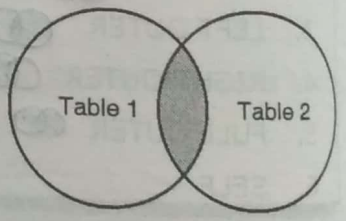

Fig. 2.11.1

Select ___ from ___
INNER JOIN ___
ON ___.___ = ___.___

Syntax

```
Select column_name_list from table_1
INNER JOIN table_2
ON table_1.column_name = table_2.column_name
```

Example

```
select ename,job,sal,emp.deptno,dept.dname from emp
INNER JOIN dept
on emp.deptno = dept.deptno;
```

Output

ENAME	JOB	SAL	DEPTNO	DNAME
CLARK	MANAGER	2450	10	ACCOUNTING
MILLER	CLERK	1300	10	ACCOUNTING
KING	PRESIDENT	5000	10	ACCOUNTING
FORD	ANALYST	3000	20	RESEARCH
SCOTT	ANALYST	3000	20	RESEARCH
JONES	MANAGER	2975	20	RESEARCH
SMITH	CLERK	800	20	RESEARCH
ADAMS	CLERK	1100	20	RESEARCH
WARD	SALESMAN	1250	30	SALES
MARTIN	SALESMAN	1250	30	SALES
TURNER	SALESMAN	1500	30	SALES
JAMES	CLERK	950	30	SALES
ALLEN	SALESMAN	1600	30	SALES
BLAKE	MANAGER	2850	30	SALES

The INNER JOIN keyword selects all rows from both tables Emp and Dept as long as there is a match between the columns. If there are records in the "Dept" table that do not have matches in "Emp", then such records will not get displayed. In Dept table department OPERATIONS of number 40 is present, but it is not displayed as no matching record is available in table emp.

2.11.2.2 Outer Join

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

(1) Left Outer Join (2) Right Outer Join (3) Full Outer Join

Consider following two tables Stud_data1 and Stud_data2

Table 2.11.4 : Stud_data1

Roll No	NAME
1	Rahul
2	Kunal
3	Jay
4	Vinay
5	Preeti

Table 2.11.5 : Stud_data2

Roll No	Address
1	Mumbai
2	Pune
3	Nasik
7	Bangalore
8	Goa

(1) Left Outer Join

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. Null values are shown at the place of right table values.

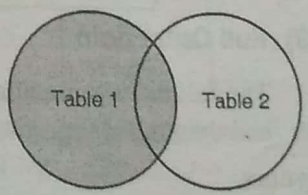

Fig. 2.11.2

Left Outer Join syntax is,

```
SELECT column-name-list
from table-name1
```

```
LEFT OUTER JOIN
table-name2
on table-1.column-name = table-2.column-name;
```

Example

```
SELECT * FROM Stud_data1
LEFT OUTER JOIN Stud_data2 ON
Stud_Data1.rollno = Stud_Data2.Rollno);
```

Output

ID	Name	ID	Address
1	Rahul	1	Mumbai
2	Kunal	2	Pune
3	Jay	3	Nasik
4	Vinay	NULL	NULL
5	Preeti	NULL	NULL

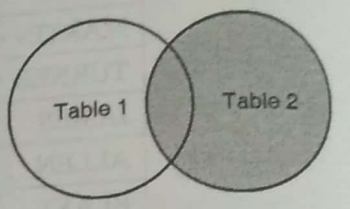

Fig. 2.11.3

(2) Right Outer Join

Returns all rows from the right table even if there are no matches in the left table. Null values are shown at the place of left table values.

Syntax

```
select column-name-list from table-name1
RIGHT OUTER JOIN table-name2
on table-1.column-name = table-2.column-name;
```

Example

```
SELECT * FROM Stud_Data1
RIGHT OUTER JOIN Stud_Data2
on (Stud_Data1.rollno= Stud_Data2. rollno);
```

Output

ID	Name	ID	Address
1	Rahul	1	Mumbai
2	Kunal	2	Pune
3	Jay	3	Nasik
NULL	NULL	7	Bangalore
NULL	NULL	8	Goa

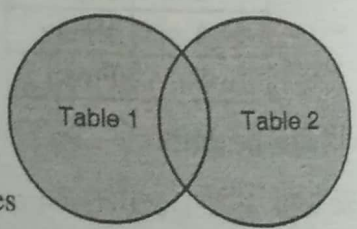

Fig. 2.11.4

(3) Full Outer Join

The full outer join returns a result with the **matching data** of both the tables and then remaining rows of both **left** table and then the **right** table.

Syntax

```
select column-name-list from table-name1
FULL OUTER JOIN table-name2
on table-1.column-name = table-2.column-name;
```

Example

```
SELECT * FROM Stud_Data1
FULL OUTER JOIN Stud_Data2
on (Stud_Data1.rollno= Stud_Data2. rollno);
```

Output

ID	Name	ID	Address
1	Rahul	1	Mumbai
2	Kunal	2	Pune
3	Jay	3	Nasik
4	Vinay	NULL	NULL
5	Preeti	NULL	NULL
NULL	NULL	7	Bangalore
NULL	NULL	8	Goa

2.11.2.3 SELF Join

- Self join is used to join a table to it-self as if the table were two tables. Virtual copies of the table are considered. Consider the table Emp

Table 2.11.6 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

- Here the MGR number indicates the name of MANAGER of particular employee. For Example MGR of MILLER is 7782 which is EMPNO of CLARK. That mean CLARK is manager of MILLER.
- Now we want to display list of employees with their manager names. In this case we have to join this emp table to itself. We will consider two copies of emp table, emp A and emp B. From emp A we will retrieve employee names while from emp B we will get manager names;

Query

```
select A.ename "Employee" ,B.ename "Manager"  from emp A, emp B where A.mgr = B.empno;
```

Output

Employee	Manager
FORD	JONES
SCOTT	JONES
ALLEN	BLAKE
JAMES	BLAKE
TURNER	BLAKE
MARTIN	BLAKE
WARD	BLAKE
MILLER	CLARK
ADAMS	SCOTT
BLAKE	KING
CLARK	KING
JONES	KING
SMITH	FORD

Ex. 2.11.1 **SPPU - Dec. 2014, 3 Marks**

Consider Following Relational Tables :
Instructor (ID, name, dept name)
Student (ID, dept_name,tot_cred)
Takes (ID, course-id, sec-id, semester, year)
Course (course –id, title, dept-name, credits)
Dept (Dept-id, dept -name)
Solve following queries using SQL
Design above relation using SQL DDL statement, primary key and foreign key.

Soln. :

```
Create table instructor(id varchar(5), name varchar(10), dept_name varchar(10), primary key(id));
Create table student(id varchar(5),dept_name varchar(10), tot_cred int(5),foreign key(id) references
instructor(id), foreign key(dept_name) references instructor(dept_name));

Create table takes(id varchar(5),course_id varchar(10), sec_id varchar(10), sem int(2), year int(2) , foreign
key(id) references instructor(id));

Create table course(course_id varchar(5),title varchar(10), dept_name varchar(10), credits int(4), foreign
key(course_id) references takes(course_id));
Create table dept(dept_id varchar(5),dept_name varchar(10),foreign key(dept_name) references
instructor(dept_name));
```

Ex. 2.11.2 **SPPU - Oct. 2016(In sem), 5 Marks**

Consider the relational database
Supplier (Sid, Sname, address)
Parts (Pid, Pname, color)
Catalog (sid, pid, cost)

Write SQL queries for the following requirements:

i) Find name of all parts whose color is green.

ii) Find names of suppliers who supply some red parts.

iii) Find names of all parts whose cost is more than Rs.25.

Select

Soln. : Consider the following 3 tables

```
mysql> select *from supplier;
+-----+-------+---------+
| sid | sname | address |
+-----+-------+---------+
| s1  | abc   | Pune    |
| s2  | pqr   | Mumbai  |
| s3  | xyz   | Nasik   |
+-----+-------+---------+
3 rows in set (0.08 sec)

mysql> select *from parts;
+-----+-------+-------+
| pid | pname | color |
+-----+-------+-------+
| p1  | prod1 | red   |
| p2  | prod2 | green |
| p3  | prod3 | blue  |
+-----+-------+-------+
3 rows in set (0.05 sec)
```

```
mysql> select *from catalog;
+------+------+------+
| sid  | pid  | cost |
+------+------+------+
| s1   | p1   |   30 |
| s2   | p3   |   10 |
| s3   | p2   |   40 |
+------+------+------+
3 rows in set (0.00 sec)

mysql>
```

Select parts.pname, p.pid, c.cost from c & p where cost > 25 & c.pid = p.pid.

i) Find name of all parts whose color is green.

```
Select * from parts where color = green;
```

Output

```
mysql> select* from  parts where color='green';
+-----+-------+-------+
| pid | pname | color |
+-----+-------+-------+
| p2  | prod2 | green |
+-----+-------+-------+
1 row in set (0.02 sec)

mysql>
```

ii) Find names of suppliers who supply some red parts.

```
Select s.sname, p.color from supplier s, parts p, catalog c where s.sid = c.sid and p.pid = c.pid and p.color ='red';
```

Output

```
mysql> select s.sname, p.color from supplier s,
 c.sid and p.pid = c.pid and p.color ='red';
+-------+-------+
| sname | color |
+-------+-------+
| abc   | red   |
+-------+-------+
1 row in set (0.02 sec)

mysql>
```

iii) Find names of all parts whose cost is more than Rs.25

```
select p.pname, c.cost from parts p, catalog c where p.pid = c.pid and c.cost > 25;
```

```
mysql> select p.pname, c.cost from parts p, catalog c
    -> where p.pid = c.pid and c.cost > 25;
+-------+------+
| pname | cost |
+-------+------+
| prod1 |   30 |
| prod2 |   40 |
+-------+------+
2 rows in set (0.05 sec)

mysql>
```

Ex. 2.11.3 **SPPU - Dec. 2015, 3 Marks**

Consider Following Relational Tables :

Person (pname, street, city)

works_for (pname, cname, salary)

Company (cname, city)

Manages (pname, mname)

Solve following quries using SQL

Find the street and city of all employees who work for the "Idea", live in Pune, and earn more than Rs. 3000.

Soln. : Consider the following tables.

```
mysql> select*from person;
+-------+-----------+--------+
| pname | street    | city   |
+-------+-----------+--------+
| abc   | Tilak Road| Pune   |
| pqr   | MG Road   | Mumbai |
| XY    | CR Road   | Nasik  |
+-------+-----------+--------+
3 rows in set (0.01 sec)

mysql>
```

```
mysql> select *from company;
+----------+--------+
| cname    | city   |
+----------+--------+
| Airtel   | Mumbai |
| Idea     | Pune   |
| Reliance | Nasik  |
+----------+--------+
3 rows in set (0.00 sec)

mysql>
```

```
mysql> select *from works_for;
+-------+----------+--------+
| pname | cname    | salary |
+-------+----------+--------+
| xy    | Idea     |   4000 |
| abc   | Airtel   |   5000 |
| pqr   | Reliance |   6000 |
+-------+----------+--------+
3 rows in set (0.00 sec)

mysql>
```

i) Find the street and city of all employees who work for the "Idea", live in Pune, and earn more than Rs. 3000.

```
Select p.street,p.city from person p,works_for w where w.cname='Idea' and p.city='Pune' and w.salary>3000;
```

Output

```
mysql> Select p.street,p.city from person p,works_for
    -> w where w.cname='Idea' and p.city='Pune' and
    -> w.salary>3000;
+------------+------+
| street     | city |
+------------+------+
| Tilak Road | Pune |
+------------+------+
1 row in set (0.00 sec)
```

Ex. 2.11.4 SPPU - Dec. 2013, 4 Marks

Consider Following Relational Tables:

Student (Roll-no, name, address)

Subject (Sub_code,Sub-name)

Marks (Roll-no, Sub-code, marks)

Solve following queries using SQL

(i) Find out average marks of each student, along with the name of student.

(ii) Find how many students have failed in the subject "DBMS".

Soln. :

```
Create table student(roll_no int(3) , name varchar(10),address varchar(20), primary key(roll_no));
Create table subject(sub_code varchar(10),sub_name varchar(10), primary key(sub_code));
```

```
Create table marks(roll_no int(3) , sub_code varchar(10),marks int(3), foreign key(roll_no) references
student(roll_no), foreign key(sub_code) references subject(sub_code));
```

Consider following tables

```
mysql> select *from manages;
+-------+-------+
| pname | mname |
+-------+-------+
| xy    | aaa   |
| pqr   | bbb   |
| abc   | ccc   |
+-------+-------+
3 rows in set (0.00 sec)

mysql>
```

```
mysql> select *from subject;
+----------+----------+
| sub_code | sub_name |
+----------+----------+
| c        | C Prog   |
| jv       | Java     |
| or       | Oracle   |
+----------+----------+
3 rows in set (0.00 sec)
```

```
mysql> select *from marks;
+---------+----------+-------+
| roll_no | sub_code | marks |
+---------+----------+-------+
|       1 | or       |    90 |
|       2 | c        |    80 |
|       3 | jv       |    78 |
+---------+----------+-------+
3 rows in set (0.00 sec)
```

(i) Find out average marks of each student, along with the name of student.

```
Select st.name,su.sub_code, m.marks from student st,subject su,marks m where st.roll_no = m.roll_no and
su.sub_code=m. sub_code;
```

Output

```
+------+----------+-------+
| name | sub_code | marks |
+------+----------+-------+
| abc  | or       |    90 |
| pqr  | c        |    80 |
| xyz  | jv       |    78 |
+------+----------+-------+
3 rows in set (0.02 sec)

mysql>
```

(ii) Find how many students have failed in the subject "Java".

```
Select st.name, su.sub_name, m.marks from student st,subject su,marks m where st.roll_no = m.roll_no and
su.sub_code=m. sub_code and  m.marks > 40 and su.sub_name='Java';
```

Output : Empty Set as no student is there who failed in Java.

Syllabus Topic : Tuple variables

2.12 Tuple Variables

A Relation R can be listed number of times as per the requirements. In this situation we need a way to refer to each occurrence of R. SQL allows us to define, for each occurrence of R in the FROM clause with the help of an "alias". This alias is known as **tuple variable.** When the R is used in the FROM clause, it is followed by the keyword AS which is optional and the name of the tuple variable.

We will see the example which we have already studied in SELF JOIN

Consider the table emp

Table 2.12.1 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Here the MGR number indicates the name of MANAGER of particular employee.

For Example MGR of MILLER is 7782 which is EMPNO of CLARK. That mean CLARK is manager of MILLER.

Now we want to display list of employees with their manager names. In this case we have to join this emp table to itself.

We will consider two copies of emp table, emp A and emp B. From emp A we will retrieve employee names while from emp B we will get manager names;

Query

```
select A.ename,B.ename from emp A, emp B where A.mgr = B.empno;
```

Output

Employee	Manager
FORD	JONES
SCOTT	JONES
ALLEN	BLAKE
JAMES	BLAKE
TURNER	BLAKE
MARTIN	BLAKE
WARD	BLAKE
MILLER	CLARK
ADAMS	SCOTT
BLAKE	KING
CLARK	KING
JONES	KING
SMITH	FORD

Here A and B are **tuple variables.**

Syllabus Topic : Ordering of Tuples

2.13 Ordering of Tuples

To arrange the displayed rows in ascending or descending order on given field(column) , Order By Clause is used.

Syntax

```
Select * from table_name order by col1,col2..[desc]
```

For Example : We need to display the employee information as per their names in ascending order, the query will be

```
Select * from emp order by Ename;
```

Output

Eno	Ename	Job	Sal
102	Ajay	Manager	18000
104	Bharati	Manager	17000
103	Dinesh	Clerk	10000
105	Prajakta	Salesman	13000
101	Susheel	Clerk	12000

Now to display same information in descending order on job the query will be

```
Select * from emp order by Ename desc;
```

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	12000
105	Prajakta	Salesman	13000
103	Dinesh	Clerk	10000
104	Bharati	Manager	17000
102	Ajay	Manager	18000

Sometimes same records may available in field given for sorting criteria. In such case we can give names of more than one columns for sorting purpose. If data in first column is same, in such case the data of second column can be taken into consideration for sorting. Consider following table

Eno	Ename	Job	Sal
101	Susheel	Clerk	12000
102	Dinesh	Manager	18000
103	Dinesh	Clerk	10000
104	Bharati	Manager	17000
105	Prajakta	Salesman	13000

Here names of two employees is same 'Dinesh'. Now sort the data we can mention sorting fields as ename and job.

```
Select * from emp order by ename,job;
```

Output

Eno	Ename	Job	Sal
104	Bharati	Manager	17000
103	Dinesh	Clerk	10000
102	Dinesh	Manager	18000
105	Prajakta	Salesman	13000
101	Susheel	Clerk	12000

Syllabus Topic : Aggregate Functions

2.14 Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single value. Usually these functions ignore NULL values(except for COUNT).

There are different types of aggregate functions :

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

Consider the table emp

(1) Min() : This function returns smallest value from specified column of the table.

Query

```
Select min(sal) from emp;
```

Output

```
800
```

(2) Max() : This function returns greatest value from specified column of the table.

Query

```
Select max(sal) from emp;
```

Output

```
5000
```

(3) Sum() : This function returns sum of all the values of specified column of the table.

Query

```
Select sum(sal) from emp;
```

Output

```
29025
```

(4) Avg() : This function returns average of all the values of specified column of the table.

Query

```
Select avg(sal) from emp;
```

Output

```
2073.21
```

(5) Count() : This function returns total number of values of specified column of the table.

Query

```
Select count(ename) from emp;
```

Output

```
14
```

Syllabus Topic : Nested Queries

2.15 Nested Queries

Writing a query inside another query is known as nested query or subquery. The inner query gets executed first, then the output on inner query is given as input to outer query. Consider the previous emp table

Example : To display records of employees working in SMITH's department

```
Select * from emp where deptno =
(select deptno from emp where ename = 'SMITH');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7876	ADAMS	CLERK	7788	01/12/1983	1100		20

Example : To display records of employees whose salary is more than the salary of FORD

```
Select * from emp where sal >
(select sal from emp where ename = 'FORD');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10

Example : To display records of employees who are senior to JONES

```
Select * from emp where hiredate <
(select hiredate from emp where ename = 'JONES');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30

Syllabus Topic : Set Membership

2.16 Set Membership

It is used to check if value of expression is matching a set of values produced by a subquery or not. There are two keywords used for SET membership – IN and NOT IN. IN is connective test for set of membership while the NOT IN is connective test for absence of SET membership.

Example – In keyword

Display records of employees working in SMITH's department

```
Select * from emp where deptno in
(select deptno from emp where ename = 'SMITH');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7566	JONES	MANAGER	7839	04/02/1981	2975		20

Example – Not In keyword

Display records of employees who are not working in SMITH's department

```
Select * from emp where deptno not in
(select deptno from emp where ename = 'SMITH');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7934	MILLER	CLERK	7782	01/23/1982	1300		10
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7839	KING	PRESIDENT		11/17/1981	5000		10
7900	JAMES	CLERK	7698	12/03/1981	950		30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30

Syllabus Topic : Set Comparison

2.17 Set Comparison

In nested queries, comparison operators are used with WHERE clause to specify the condition to filter the data to be display. Following are the various comparison operators

Comparison Operator	Description
=	Equal
<>	Not Equal
>	Greater Than
<	Less Than
>=	Greater Than or Equal
<=	Less Than or Equal

Example : Consider following table emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

< operator in nested query

Example : Display list of employees having salary less than ADAMS.

```
select * from emp where sal <
(select sal from emp where ename = 'ADAMS');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7900	JAMES	CLERK	7698	12/03/1981	950		30

Syllabus Topic : PL/SQL

PL/SQL stands for Procedure Language / Structure Query Language. It is the combination of SQL along with the procedural features of programming languages.

PL/SQL includes procedural language elements such as conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variables of those types, and triggers. It can handle exceptions (runtime errors). Arrays are supported involving the use of PL/SQL collections. It has included features associated with object-orientation. One can create PL/SQL units such as procedures, functions, packages, types, and triggers, which are stored in the database for reuse by applications.

Syllabus Topic : Concept of Stored Procedures and Functions

2.18 Concept of Stored Procedures and Functions

2.18.1 Stored Procedures

SPPU - May 13

University Question

Q. Explain Stored Procedures. **(May 2013, 4 Marks)**

Stored procedure is a group of SQL statements which can be executed repeatedly. It allow for variable declarations, flow control and other useful programming techniques. Parameters are the important part of procedures. The parameters make the stored procedure more flexible and useful.

Consider the table

```
mysql> select *from  student;
+------+-------+------------+-------+
| rno  | name  | dob        | class |
+------+-------+------------+-------+
|    2 | aa    | 1990-11-21 | NULL  |
|    3 | priya | 1989-10-30 | NULL  |
|    4 | asd   | 2000-02-02 | NULL  |
|    5 | ggg   | 0000-00-00 | NULL  |
+------+-------+------------+-------+
4 rows in set (0.00 sec)

mysql>
```

Syntax

```
DELIMITER //
CREATE procedure procedure _name
([parameter(s)])
STATEMENTS
DELIMITER //
```

IN Parameter : Accepts value when procedure get called.

Example : Create a procedure which should accept rollno as parameter and display the record.

```
DELIMITER //
CREATE PROCEDURE display(IN r integer (3))
BEGIN
SELECT * FROM students WHERE rno = r;
END //
DELIMITER ;
```

Now the procedure can be called as

```
CALL display(3);
```

Output

```
+------+-------+------------+-------+
| rno  | name  | dob        | class |
+------+-------+------------+-------+
|    3 | priya | 1989-10-30 | NULL  |
+------+-------+------------+-------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.06 sec)
```

Out Parameter : The value of an OUT parameter can be changed inside the stored procedure. The changed value is passed back to the calling program. The initial value of OUT parameter cannot be accessed by the procedure.

Example

```
DELIMITER $$
CREATE PROCEDURE display1(IN r INT,OUT nm
VARCHAR(25))
BEGIN
select name into nm from students where rno = r;
END $$
DELIMITER ;
```

Now the procedure can be called as

```
CALL display1(3 ,@n);
```

Here n is the OUT parameter which stores the name of student having rno 3;

Then execute the command

```
Select @n;
```

Output

```
+-------+
| @n    |
+-------+
| priya |
+-------+
1 row in set (0.00 sec)

mysql>
```

2.18.2 Stored Functions

SPPU - May 13

University Question

Q. Explain Stored functions. **(May 2013, 3 Marks)**

Stored Function is same as of stored procedure means it is a group of SQL statements which can be executed repeatedly. It allow for variable declarations flow control and other useful programming techniques.

Just difference is that function can return value.

Syntax :

```
CREATE FUNCTION function
  RETURNS data type
DETERMINISTIC
STATEMENTS
```

Create a function to having rollno 3.

```
DELIMITER |
CREATE FUNCTION anuals
 RETURNS int(7)
  DETERMINISTIC
   BEGIN
    DECLARE asal int(7);
Set asal = sal * 12;
    RETURN asal;
   END|
```

Then execute the cor

Select anualsal(5000

Output

```
mysql> Select anualsal(5
+-----------------+
| anualsal(5000)  |
+-----------------+
|           60000 |
+-----------------+
1 row in set (0.06 sec)
```

Syllabus Top

2.19 Cursor, Trig

2.19.1 Cursor

University Question

Q. What is curs Cursor?

Q. What is curs PL/SQL with

– Cursor is used to the records one calculate the av then we can re add in the tota concept used manipulate data

Functions → return a single value
Procedures → No return.

Syntax :

```
CREATE FUNCTION function_name ([parameter(s)])
   RETURNS data type
DETERMINISTIC
STATEMENTS
```

Create a function to return record of student having rollno 3.

```
DELIMITER |
CREATE FUNCTION anualsal (sal int)
 RETURNS int(7)
  DETERMINISTIC
   BEGIN
    DECLARE asal int(7);
Set asal = sal * 12;
    RETURN asal;
   END|
```

Then execute the command

Select anualsal(5000) from dual;

Output

```
mysql> Select anualsal(5000) from dual;
+-----------------+
| anualsal(5000) |
+-----------------+
|          60000 |
+-----------------+
1 row in set (0.06 sec)
```

Syllabus Topic : Cursor, Trigger

2.19 Cursor, Trigger, Assertions

2.19.1 Cursor

SPPU - May 13, Dec. 13

University Questions

Q. What is cursor? Explain various types of Cursor? **(May 2013, 8 Marks)**

Q. What is cursor? Explain explicit cursor in PL/SQL with suitable example? **(Dec. 2013, 6 Marks)**

- Cursor is used to traverse in the database to access the records one by one. For example if we want to calculate the average of marks of all the students, then we can retrieve marks of every student and add in the total_marks. Cursor is just like loop concept used to traverse to every row and manipulate data.
- An area of memory(Context) is allocated for the processing of SQL statements. The context area contains information necessary to complete the processing, including the number of rows processed by the statement, a pointer to the parsed representation of the statement.
- Cursor is a handle or pointer to the context area.
- There are two types of cursors
  1. Implicit Cursor
  2. Explicit cursor

2.19.1.1 Implicit Cursor

- When there is no explicit cursor, the implicit cursors are created automatically whenever an SQL statement is executed. Programmers cannot control the implicit cursors and the information in it.
- When Data Manipulation statements like insert, update or delete are executed, an implicit cursor is automatically associated with these statements. In case of insert statement the data is hold by cursor while for delete and update statements, cursor identifies the rows that would be affected.
- Consider following example in which increment of 10% is given to all the employees. Here an implicit cursor is created to identify the set of rows in the table which would be affected by the update.

```
UPDATE employee
  SET salary = salary * 0.1;
```

2.19.1.2 Explicit Cursor

An explicit cursor is the one in which the cursor name is explicitly assigned to the select statement. Processing an explicit cursor involves following three steps.

(1) **Open** : The Open statement executes the select statement. Positions the cursor at the first row.

(2) **Fetch** : The Fetch statement retrieves the current row and advances the cursor to the next row for processing.

(3) **Close** : After processing of last row the cursor is disabled with the help of Close statement.

Example : Consider the table city

```
mysql> select *from city;
+-----+-----------+----------+
| id  | city_name | state_id |
+-----+-----------+----------+
| 101 | Pune      |        1 |
| 102 | Mumbai    |        1 |
| 103 | Nasik     |        1 |
+-----+-----------+----------+
```

```
DELIMITER //
CREATE PROCEDURE cname(
  IN in_state_id INT
)
BEGIN
  DECLARE record_not_found INTEGER DEFAULT 0;
  DECLARE mycity_name VARCHAR(255)
DEFAULT "";
  DECLARE mysql_cursor CURSOR FOR
     SELECT city_name FROM city
     WHERE state_id = in_state_id;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET
record_not_found = 1;
  OPEN mysql_cursor;
   cities_loop: LOOP
  FETCH mysql_cursor INTO mycity_name;
    IF record_not_found THEN
      LEAVE cities_loop;
    END IF;
  SELECT mycity_name;
 END LOOP cities_loop;
 CLOSE mysql_cursor;
END //
DELIMITER ;
```

To call the procedure give query

```
CALL cname(1);
1 is the state id.
```

2.19.2 Trigger

SPPU - May 13

University Question

Q. Explain Triggers. **(May 2013, 4 Marks)**

A trigger is a set of actions which get executed automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a particular table.

Purpose of trigger

- To generate data automatically
- Validate input data
- Replicate data to different files to achieve data consistency
- Write to other files for audit trail purposes

Syntax

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

Consider the table employee

```
c:\wamp\bin\mysql\mysql5.5.24\bin\mysql.exe
+-----+---------+------+
| eno | ename   | sal  |
+-----+---------+------+
|   1 | Rajesh  | 9000 |
|   2 | Sudhir  | 8000 |
|   3 | Radhika | 7000 |
+-----+---------+------+
3 rows in set (0.00 sec)
```

Example

```
delimiter //
CREATE TRIGGER upd_check BEFORE UPDATE ON
employee
FOR EACH ROW
BEGIN
  IF NEW.sal < 5000 THEN
  SET NEW.sal = 5000;
  ELSEIF NEW.sal > 20000 THEN
  SET NEW.sal = 20000;
  END IF;
END;//
```

Now using update query if salary is updated to less than 5000 then it will be automatically set 5000 and if it is updated to more than 20000 then it will be automatically set 20000.

2.19.3 Assertion

Since SQL-92, asse SQL standard. An asser define the constraint on true.

Difference between As constraints is :

Unlike check cons table or column level i on schema level. Ass create table or alter constraint. To imple database level such as **table check constraint**

To declare Cross- used but it is error-pro on database to serve th is declared on some specified by assertion transaction which le tables.

Syntax for creating a

```
CREATE ASSERTION A
(Condition);
```

The condition is

Example : Bank ap database to store da given below :

```
Account(acc_no, b
Depositor(custome
```

Create an asse have maximum two

```
CREATE ASSERTI
(
(SELECT COUNT(*)
WHERE Account.a
);
```

Syllabus Topic : Assertion

2.19.3 Assertion

Since SQL-92, assertions have been part of the SQL standard. An assertion is an expression used to define the constraint on database that must be always true.

Difference between Assertions and check constraints is :

Unlike check constraints they are not defined on table or column level instead of that they are defined on schema level. Assertions are not defined within create table or alter table statements like check constraint. To implement some constraint at the database level such as **cross-row constraints, multi-table check constraints,** SQL assertions can be used.

To declare Cross-row constraint triggers can be used but it is error-prone, so better way use assertions on database to serve the same purpose. If an assertion is declared on some tables, then all the constraints specified by assertion are followed by the database transaction which leads to modification on those tables.

Syntax for creating assertion is as follows :

```
CREATE ASSERTION Assertion_Name CHECK
(Condition);
```

The condition is in the form of an SQL query.

Example : Bank application has many tables in their database to store data. Schema of two of them are given below :

```
Account(acc_no, branch_name, balance);
Depositor(customer_name, acc_no);
```

Create an assertion which allows customers to have maximum two accounts in a bank application.

```
CREATE ASSERSTION Num_of_Acconts CHECK
(
(SELECT COUNT(*) FROM Account, Depositor
WHERE Account.acc_no=Depositer.acc_no) =< 2
);
```

2.19.4 Assertion Vs Triggers

Sr. No.	Assertion	Triggers
1.	Assertion is used to specify restrictions on database, it doesn't used to modify the data.	Triggers are used to give restrictions as well as to modify the data.
2.	All assertions can be implemented as triggers.	All triggers can't be implemented as assertions.
3.	Assertions are not linked to specific tables in the database and also not linked to specific events.	Triggers are linked to specific tables and specific events.

Syllabus Topic : Roles and Privileges

2.20 Roles and Privileges

2.20.1 Roles

- This is a group of privileges that will be assigned to users : Creating a Role
- CREATE ROLE 'admin'; You can also create more than one role at once
- CREATE ROLE 'dba', 'developer', 'readonly';

2.20.2 Privileges

- **Privileges** defines the access rights to database users on database objects (like functions, procedures, or tables). They also define rights to run a SQL statement, or PL/SQL Package.

Creating New User

- There are different ways to create users with custom permissions.

  Creating new user within the MySQL shell:

  CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';

- In this situation the newly created user has no permissions to do anything with the databases. Even if the new user try to login with the given password, he will not be able to reach the MySQL shell.

- Hence the most important initial step is to provide the user with access to the information they will need.

  GRANT ALL PRIVILEGES ON *. * TO 'newuser'@'localhost';

- The * symbol indicates all the databases and tables. Means user get rights like read, edit, execute and perform all tasks on the database.

Grant Different User Permissions

The following list shows other common possible permissions that users can get.

- ALL PRIVILEGES - allows a MySQL user all access to a designated database.
- CREATE - allows user to create new tables or databases
- DROP - allows user to them to delete tables or databases.
- DELETE - allows user to delete rows from tables.
- INSERT - allows user to insert rows into tables.
- SELECT - allows user to use the Select command to read through databases.
- UPDATE - allow user to update table rows.
- GRANT OPTION - allows user to grant privileges.
- REVOKE - removes granted privileges.

Granting privileges

```
GRANT [type of permission] ON [database name].[table name] TO '[username]'@'localhost';
REVOKE privileges
REVOKE [type of permission] ON [database name].[table name] FROM '[username]'@'localhost';
Deleting user
DROP USER 'demo'@'localhost';
```

Syllabus Topic : Embedded SQL

2.21 Embedded SQL

- The method of combining of general purpose programming languages and database language like SQL is called as **embedded SQL.**
- The SQL is good for defining database structure and defining short queries but for some application it is required to mix SQL with programming language.
- In other words, SQL defines WHAT is required but it can't define HOW to meet this requirement.
- SQL is used to express queries but all the queries can't be expressed with SQL alone, so there is a need of combining database language with general purpose programming language to express such queries.

Some concepts in Embedded SQL

- **Host language :** These are programming languages (such as C, C++, COBOL, Java, etc) in which SQL statements are embedded.
- **SQL Pre-Compiler :** A pre-compiler is used to process embedded SQL statements. It is used to translate SQL statements into DBMS library calls which can be implemented into host language.
- The following code is a simple embedded SQL program, written in C

The program accepts order number from user, retrieves the customer number, salesperson, and status of the order, and displays the data on the screen.

```
int main()
{
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    int Order_ID;
    int Cust_ID;
    char Sales_Person[10]
    char Order_Status[6]
  EXEC SQL END DECLARE SECTION;

  /* Set up error processing */
  EXEC SQL WHENEVER SQLERROR GOTO qry_error;
  EXEC SQL WHENEVER NOT FOUND GOTO
invalid_number;

  /* Prompt the user for order number */
  printf ("Enter order number: ");
  scanf_s("%d", &Order_ID);

  /* Execute the SQL query */
  EXEC SQL SELECT Cust_ID, Sales_Person,
Order_status
    FROM Orders
    WHERE Order_ID = :Order_ID
    INTO :Cust_ID, :Sales_Person, :Order_Status;
```

```
    /* Display the results */
    printf ("Customer number:  %d\n", Cust_ID);
    printf ("Salesperson: %s\n", Sales_Person);
    printf ("Status: %s\n", Order_Status);
    exit();

qry_error:
    printf ("SQL error: %ld\n", sqlca->sqlcode);
    exit();

invalid_number:
    printf ("Invalid order number.\n");
    exit();
}
```

- **Host Variables :** These variables are declared in section starting with BEGIN DECLARE SECTION and ending with END DECLARE SECTION keywords. These variables are prefix by (:) in an embedded SQL statement. The precompiler distinguish between host variables and database objects(like column and tables) using (:).
- **Data Types :** Usually the data types of DBMS and a host language are different. This affects the host variable because they are used by both by host language statements and embedded SQL statements. When there is no similar type in host language that corresponds to a DBMS data type, the DBMS automatically converts the data.
- **Error Handling :** Run times errors are reported by DBMS to the application program using SQL Communications Area, or SQLCA. In the above program the statement INCLUDE SQLCA includes the SQLCA structure in program. It is used when errors are processed by the programs which are returned by the DBMS.
- **WHENEVER...GOTO statement :** It tells the pre-compiler to generate error-handling code when an error occurs.

2.21.1 Advantages of Embedded SQL

1) In embedded SQL, SQL is used to handle the database and the host language handles the variables and other input and output functions.
2) In the development cycle tasks like parsing and optimizing are done which taked high overheads. It increases efficiency of CPU resources.
3) Portability is achieved. The application be precompiled on one machine and can be moved another machine for further processing.
4) There is no need that programmer should understand the complex structure of DBMS. He has create only the embedded SQL source program code.

Syllabus Topic : Dynamic SQL

2.22 Dynamic SQL

SPPU - Dec. 13

University Question

Q. Write a short note on Dynamic SQL. **(Dec. 2013, 6 Marks)**

Although static SQL works well in many situations, in some applications the data access cannot be determined in advance.

Dynamic SQL refers to SQL code that is generated in an application or from the system tables and then executed against the database to manipulate the data at run time. The SQL code is not stored in the application but rather it is collected depending upon the input given by user.

The dynamic SQL helps to develop powerful applications which allows us to create database objects and manipulate them based on the input provided by the user.

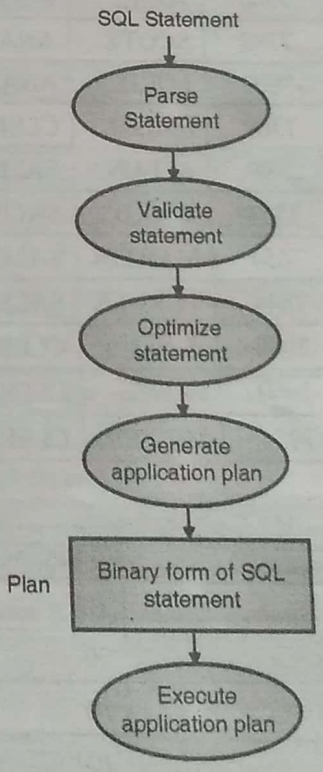

Fig. 2.22.1

In dynamic SQL the column_name or Table_name are not known at compile time so the DBMS can't able to prepare the SQL statement for execution in advance.

In this method, when the program is executed, the SQL statement get the table_name or column_name from user and this statement is given to the DBMS for further execution.

In the Fig. 2.22.1 we can observe the general execution process of dynamic sql.

An Example of Dynamic SQL statement

```
mysql> PREPARE stmt FROM
    -> 'select count(*)
    -> from information_schema.schemata
    -> where schema_name = ? or schema_name = ?'
;
Query OK, 0 rows affected (0.00 sec)
Statement prepared
mysql> EXECUTE stmt
    -> USING @schema1,@schema2
```

Output

```
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
```

SQL *PLUS Assignments & Solved Queries

Table 2.22.1 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Table 2.22.2 : Dept

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

1. Display the name and jobs of the employees who are clerk.

```
Select Ename, Job From Emp Where Job='Clerk';
```

2. Display the name and dept numbers of the employees whose job is an 'Analyst' or a 'Clerk'.

```
Select Deptno, Ename, Job From Emp Where Job In('ANALYST','CLERK');
```

3. Give the List of Employees sorted On Name.

```
Select * From Emp Order By Ename;
```

4. Display different kinds of jobs available.

```
Select Distinct Job From Emp;
```

5. Find the locations at which various Employees are situated.

```
Select Emp.Ename, Dept.Location From Emp, Dept Where Emp.Deptno= Dept.Deptno ;
```

6. Display the name, sal of the Employees whose dept locations is 'Chicago'.

```
Select Emp.Deptno, Ename, Sal From Emp Where Emp.Deptno= (Select Deptno From Dept Where
Location='Chicago') ;
```

7. Display the names of employees whose dept is 30.

```
Select Deptno, Ename From Emp Where Deptno= 30 ;
Select all Employees whose name is 5 letters long.
Select * From Emp, Where Length(Ename)=5 Order by Deptno;
Select all Employees whose name ends with R.
Select * From Emp, Where Ename like '%R';
```

8. Display the names of employees working in sales or research dept.

```
Select Ename, Deptno From Emp Where Deptno In (Select Deptno From Dept where Dname In ('SALES',
'RESEARCH'));
```

9. Display the dept no and dept name whose dept no > 20.

```
Select Deptno, Dname From Dept Where Deptno>20;
```

10. Find out dept in which maximum employees works.

```
Select Deptno, count(*) From Emp Group by Deptno Having Count(*)=(Select MAX(COUNT(*) From Emp Group
by Deptno);
```

11. Display the job, dept no, name of the employees whose name start with 'B' or 'M'.

```
Select Deptno, Ename, Job From Emp Where Ename Like 'B%' Or Ename like 'M%';
```

12. Find out the difference between maximum sal in dept 10 and minimum sal earn by a person in dept 30.

```
Select A.Sal-B.Sal From Emp A, Emp B Where A.Sal=(Select max(Sal) From Emp  where deptno=10) And
B.Sal=(Select min(Sal) From Emp  where deptno =30);
```

13. Find out the difference in dollers between the average earning of dept 20 and 30.

```
Select avg(A.Sal)-avg(B.Sal) From Emp A, Emp B group by A.Deptno,B.Deptno having avg(A.Sal)=(Select
avg(Sal) From Emp  where Deptno=20) And avg(B.Sal)=(Select avg(Sal) From Emp  where Deptno=30);
```

14. Display names, Sal and commission for Employees whose commission is more than 5% of sal.

```
Select Ename, Sal, Comm From Emp where Comm > Sal*0.05;
```

15. Find out the employees whose sal is < the average sal of dept 20.

```
Select * From Emp where Sal <(select avg(Sal) From Emp where Deptno=20);
```

16. Display information about people who is having the maximum no of people reporting to them.

```
Select * From Emp where Empno= (select MGR from Emp Group by MGR Having Count(MGR)=(select
Max(Count(MGR)) From Emp Group By MGR));
```

17. Give the query to concatenate Empno, name and manager from Emp table.

```
Select Empno||Ename||MGR From Emp;
```

18. List the employees who are hired earliest and latest.

```
Select * From Emp where HireDate=(Select Max(HireDate) From Emp) Or HireDate=(select min(HireDate) From
Emp);
```

19. List the names who are reporting to 'Blake'.

```
Select * From Emp where MGR=(Select Empno From Emp Where Ename ='Blake');
```

20. List all the employees hired in the month of December.

```
Select * From Emp where To_char(HireDate,'Month')='DEC';
```

21. List all the employees hired after 1980.

```
Select * From Emp where To_char(HireDate,'YYYY')>1980;
Pramote 'JAMES; to 'MANAGER' with increment of 1000
Update emp set job = manager and sal = sal + 1000 where ename = 'JONES';
```

22. Write sql command to find average annual salary per job in each department

```
Select deptno,avg(sal*12) from emp group by deptno;
```

23. Display department numbers of the departments who has more than one clerks.

```
Select deptno,count(*) from emp where job = 'CLERK' group by deptno having count(*)>1;
```

❑❑❑

CHAPTER 3

Syllabus

- Relatio
- Relatio
- Databa
- Norma
  Deper
- Modeli

Syllabus

3.1 Relatio

3.1.1 Introdu

- The **Relation** tables.) Thi Dr. E.F. Cod model is th processing ap
- Relational m simplest stru models like
- The relation primary mod
- The relation having all th to process da
- Most of t Systems (DI
- The relatio components
  1. The set defines structur
  2. Integrit protect
  3. The op (data m