Chapter-4

Relational Database Query Language

Introduction to SQL (Structured Query Language)

SQL is the standard query language to communicate with a relational database. SQL works with database programs like MS-Access, MS-SQL server, Oracle, Sybase etc. SQL is not a case sensitive language. IBM developed the original version of SQL, originally called Sequel, whose name has now been changed to SQL (Structured Query Language).

Components of SQL

1. **DDL (Data Definition Language)**

   DDL is the part of SQL that allows database users to create and restructure database objects. The main DDL statements are CREATE, ALTER and DROP.

2. **DML (Data Manipulation Language)**

   DML is the part of SQL that allows manipulation of data within object of a relational database. DML statements are INSERT INTO, UPDATE, DELETE etc.

3. **DQL (Data Query Language)**

   DQL is the part of SQL that is used to compose queries against a relational database.

   DQL statements are SELECT.

4. **DCL (Data Control Language)**

   DCL is the part of SQL that provides control access to data and to the database.

   DCL statements are COMMIT, ROLLBACK, GRANT, REVOKE etc.

Domain Types in SQL

Domain type or Data type represents the type of data that an object can contain, such as character data or integer data. Some of the mostly used SQL basic data types are

1. int:

   Its full form is Integer. It can store whole numbers in the range (2^{31} to $2^{31}-1$)

2. smalllint:

   It is subset of integer. It can store whole numbers in the range (2^{15} to $2^{15}-1$)

3. tinyint:

   whole numbers (0-255)

4. bigint:

   whole numbers (2^{63} to $2^{63}-1$)

5. numeric(p,d):

fixed point number, with user specified precision of p digits, with d digits to the right of decimal point.

6. datetime:

date and time data

7. char(n):

the char data type stores fixed length character string with user specified length n. The maximum number of character the data type can hold is 255 character. it uses static memory allocation.

8. varchar(n):

variable length character string with user specified length n. it is slower than char but is highly memory efficient as it uses dynamic memory allocation.

9. nvarchar(n):

nvarchar type is used to store multilingual data using the Unicode representation.

10. float(n):

A floating-point number, with precision of at least n digits.

Data Definition language

- DDL is the part of SQL that allows database users to create and restructure database objects.
- A database schema is specified by a set of definitions expressed by a special language called data definition language.
- So data definition language is a set of SQL commands which is used to create, modify and delete database structure but not data.
- The main tasks of DDL are:
  1. Creating database object like tables.
  2. Modifying database objects.
  3. Destroying database objects.

DDL also updates a special set of tables called the data dictionary. The result of compilation of DDL statement is a set of tables that is stored in special files called data dictionary. A data dictionary contains **metadata** i.e. data of data. Data about structure of a database is called metadata.

The main DDL statements are CREATE, ALTER and DROP.

1. CREATE DATABASE command

It is used to create a new database.

Syntax:

CREATE DATABASE <database_name>

Eg.

CREATE DATABASE db_Morgan

2. CREATE TABLE command

It is used to create a table.

Syntax:

```
CREATE TABLE <table_name>
(
Field1 datatype1,
Field2 datatype2,
.
.
.
FieldN datatypeN
)
```

E.g.

CREATE TABLE student (id int, name char(20), roll int)

3. DROP DATABASE command

It is used to destroy the complete database structure.

Syntax:

DROP DATABASE <database_name>

E.g.

DROP DATABASE db_EEMC

4. DROP TABLE command

It is used to destroy the complete structure of table.

Syntax:

DROP TABLE <table_name>

E.g.

DROP TABLE student

5. ALTER TABLE command

it is used to change the structure of table.

i. add new column

Syntax:

ALTER TABLE <table_name> ADD <Newfield1 datatype1>,<Newfield2 datatype2>..<NewfieldN datatypeN>

E.g.

ALTER TABLE student ADD maths int, section char(20)

ii. delete existing column

Syntax:

ALTER TABLE <table_name> DROP COLUMN <column_name1>,<column_name2>……<column_nameN>

E.g.

ALTER TABLE student DROP COLUMN maths,section

iii. Changing the datatype of existing column

Syntax:

ALTER TABLE <table_name> ALTER COLUMN <column_name new datatype>

E.g.

ALTER TABLE tbl_student ALTER COLUMN ram nvarchar(50)

DML (Data Manipulation Language)

The data manipulation languge is use for modifying the data stored in the database.

1. INSERT INTO command

It is used to insert new record into a table.

Syntax:

a. INSERT INTO <table_name> (field1,field2-----fieldN) VALUES (value1, value2----------value
b. INSERT INTO <talb_name> values (value1, value2……..valueN)

- INSERT query of type 'a' can be used to insert values for selected field. The non selected field will have NULL value insertd upon executaion of the above query.
- INSERT query of type 'b' is used when we have to insert values for all the fields in the table. However if the numbers and datatype of supplied field doesn't match the numbers and datatype defined in table defination then it will create error on execution.

E.g.

INSERT INTO tbl_student values(1,'Bheeshma',10)

INSERT INTO tbl_student values(1,'Bipin') // wrong

INSERT INTO tbl_student (id,name) values (2,'Bhesh') // NULL value will be inserted for roll field

2. UPDATE command

It is used to modify selected or all records from a table.

Syntax:

a. UPDATE <table_name> SET field1=newvalue1, field2=newvalue2……fieldN=newvalueN

b. UPDATE <table_name> SET field1=newvalue2, field2=newvalue2……fielnN=newvaluN WHERE <Expression>

E.g.

UPDATE tbl_student SET roll=5

UPDATE tbl_student SET roll=5 WHERE name='Bhesh'

OR

UPDATE A SET roll=7 FROM tbl_student A WHERE name="Bhesh'

3. DELETE command

It is used to delete all or selected records from a table.

Syntax:

a. DELETE FROM <table_name> // deletes all records from table

b. DELETE FROM <table_name> WHERE <Expression>

E.g.

DELETE FROM tbl_student // delete entire records from tbl_student

DELETE from tbl_student where id>10 // delete all records whose id value is greater than 10

DQL (Data Query Language)

1. SELECT command

It is used to display all or selected records from a table.

Syntax:

a. SELECT <field1>,<field2>…..<fieldN> FROM <table_name>

b. SELECT * FROM <table_name> // * represents all the field name defined for the particular table

c. SELECT * FROM <table_name> WHERE <Experssion>

d. SELECT <field1>,<field2>….<fieldN> FROM <table_name> WHERE <Expression>

e. SELECT DISTINCT <field_name> FROM <table_name>

f. SELECT TOP <N> <field1>,<field2>…<fieldN> from <table_name>

E.g.

SELECT id,name FROM tbl_student // display field id and name with all records

SELECT * FROM tbl_student // display all fields with all records

SELECT * FROM tbl_student where id<10 // display all the fields with only those records whose id is less than 10

SELECT DISTINCT name from tbl_student // display all the names from tbl_student without duplication of values

SELECT TOP 10 * FROM tbl_student // display first 10 records

DCL (Data Control Language)

1. COMMIT and ROLLBACK commands

DCL is a set of SQL commands for controlling access to data and to the database.

Syntax:

COMMIT TRANSACTION <transaction_name>

ROLLBACK TRANSACTION <transaction_name>

E.g. **BEGIN TRANSACTION Tr1**

BEGIN TRY

INSERT INTO tbl_student VALUES (1,'Bhesh',10)

DELETE FROM tbl_student WHERE name='Bheeshma'

UPDATE tbl-student SET name="Bipin" WHERE id=1

COMMIT TRANSACTION Tr1

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION Tr1

END CATCH

2. GRANT and REVOKE commands

GRANT is a command used to provide access right or privilege on the database objects to the users. These commands are generally used by DBA.

REVOKE is a command which remove user access right or privilege to the database object

Syntax:

GRANT <Privilage_Name> ON <Object_Name> TO <User_Name>

REMOVE <Privilage_Name> ON <Object_Name> FROM <User_Name>

- Privilage_Name means access right. Some of the access right are ALL, SELECT, INSERT, UPDATE, DELETE, EXECUTE.
- Object_Name is the name of database object such as TABLE, VIEWS, STORED PROC etc.

- User_Name is the name of user to whom an access right is being granted.

E.g.

GRANT SELECT ON tbl_student TO user1 // grants select permission on tbl_student to user1

REVOKE SELECT ON tbl_student FROM user1 // restrict user1 from selecting data from tbl_student

Data Constraints

Constraints define rules that must be followed to maintain consistency and correctness of data. Generally constraints are created at the time of creation of table; however they can be added after table creation also. A constraint can be defined on a column while creating a table. Constraint can be defined into the following types.

1. Primary Key Constraint:

A primary key constraint is defined on a column or a set of columns whose values uniquely identify all the rows in a table. A primary key column can't contain NULL values.

i. Creating primary key during table creation

Syntax:

a.

```
CREATE TABLE <table_name>
(
        Field1 datatype1 CONSTRAINT <constraint_name>  PRIMAY KEY,
        Field2 datatype2,
        .
        .
        .
        FieldN datatypeN

)
```

E.g.

CREATE TABLE tbl_student (id int CONSTRAINT idcode PRIMARY KEY,roll int, class int, name varchar(50))

b.

```
CREATE TABLE <table_name>
(
Field1 datatype1 Primary key,
Field2 datatype2,
```

.

.

.

FieldN datatypeN

)

E.g.

CREATE TABLE tbl_student (id int PRIMARY KEY,roll int,class int, name varchar(50))

c.

CREATE TABLE <table_name>

(

Field1 datatype1 ,

Field2 datatype2,

.

.

.

FieldN datatypeN,

CONSTRAINT <constraint_name> PRIMARY KEY(<field1>,<field2>……<fieldN>)

)

d.

CREATE TABLE <table_name>

(

Field1 datatype1 ,

Field2 datatype2,

.

.

.

FieldN datatypeN,

PRIMARY KEY(<field1>,<field2>……<fieldN>)

)

E.g.

CREATE TABLE tbl_student (id int ,roll int,class int, name varchar(50), PRIMARY KEY(roll,class))

ii) Adding primary key after table creation

Syntax:

a. ALTER TABLE <table name> ADD PRIMARY KEY(<field name>)
b. ALTER TABLE <table name> ADD CONSTRAINT <constraint name> PRIMARY KEY(<field name>)
c. ALTER TABLE <table name> ADD CONSTRAINT <constraint name> PRIMARY KEY(field1,field2)
d. ALTER TABLE <table name> ADD primary key(field1,field2)

E.g.

ALTER TABLE tbl_student ADD PRIMARY KEY (id)

ALTER TABLE tbl_student ADD CONSTRAINT idcode PRIMARY KEY (roll,class)

iii) Removing primary key Constraint

Syntax:

ALTER TABLE <table_name> DROP <constraint_name>

E.g.

ALTER TABLE tbl_student DROP idcode

2. Foreign Key constraint:

A foreign key is imposed on a column of one table which refers the primary key column of another table. A foreign key constraint removes the inconsistency in two tables when the data in one table depends on data in another table. Simply it is the linking pin between two tables.

Syntax:

```
CREATE TABLE <foreigntable_name>
        (
        Field1 datatype1  REFERENCES <primarytable_name>(<Primary_Key_column_name/ primary key
        constraint  name>),
        Field2 datatype2,
        .
        .
        FieldN datatypeN

        )
```

E.g.

CREATE TABLE tbl_marks (id int REFERENCES tbl_student(id), maths int, science int)

CREATE TABLE <foreigntable_name>

(

Field1 datatype1 CONSTRAINT <constraint_name> REFERENCES <primarytable_name>(<Primary_Key_column_name/ primary key constraint name>),

Field2 datatype2,

.

.

FieldN datatypeN

)

E.g.

CREATE TABLE tbl_marks (id int CONSTRAINT fcode REFERENCES tbl_student(id), maths int, science int)

3. Unique Constraint:

The unique constraint is used to enforce uniqueness on non-primary key columns. The unique constraint is similar to primary key constraint except that it allows one NULL row. Also multiple unique constraints can be created on a table.

Syntax:

a. CREATE TABLE <table_name> (field1 datatype1 UNIQUE, field2 datatype2…..fieldN datatypeN)
b. CREATE TABLE <table_name> (field1 datatype1 CONSTRAINT <constraint_name> UNIQUE, field2 datatype2,…..fieldN datatypeN)

E.g.

CREATE TABLE tbl_student(id int CONSTRAINT unkey UNIQUE,NAME VARCHAR(20))

CREATE TABLE tbl_student(id int UNIQUE,NAME VARCHAR(20))

4. Check Constraint:

A check constraint enforces domain integrity by restricting the value to be inserted in a column.

Syntax:

a. CREATE TABLE <table_name>(field1 datatype1 CHECK (expression),field2 datatype2,---fieldN datatypeN)

E.g.

CREATE TABLE tbl_student(id int,roll int,maths float CHECK(maths<99.99 and maths>0),name nvarchar(50) CHECK(len(name)>5)) // note: len() is a function which return the number of characters.

5. Default Constraint:

A default constraint can be used to assign a constant value to a column and the user need not insert values for such column. However if the user provides value then the particular value will be stored.

Syntax

a. CREATE TABLE<table_name> (field1 datatype1 DEFAULT <default value>,field2 datatype2…fieldN datatypeN)

E.g.

CREATE TABLE tbl_student(id int, name varchar(50), address varchar(50) DEFAULT 'Kathmandu')

6. Not Null Constraint:

If a column is declared as not null, a value has to be necessary to be inserted for such column.

Syntax

a. CREATE TABLE <table_name> (field1 datatype1 NOT NULL,field2 datatype2,…….fieldN datatypeN)\

E.g.

CREATE TABLE tbl_student(id int, name varchar(50) NOT NULL,roll int)

Note:

CREATE TABLE tbl_student(id int identity(1,1), name varchar(50),roll int)

In the above table tbl_student has a field id with identity() function, this type of field are called auto increment field. It act as a default value for a column that increments for each record.

Syntax: identity(seed,increment)

Seed: start number

Increment: incremental value for seed

Operators

To conditionally select, update and delete data from a table, a WHERE clause followed by an expression involving several operators is used. Operators can be classified as below.

1. **Arithmetic Operators:**

Arithmetic operators are used to perform mathematical operations such as addition, subtraction, multiplication and division on numeric columns.

Arithmetic operators are + ,-, *, / and %.

Consider the following relation tbl_Marks

tbl_Marks(id, name, address, maths, science, english)

E.g.

a. Increment the mark of math's of all student by 10%.

```
UPDATE tbl_Marks SET maths=maths*1.1
```

b. Display all the records if every student is given a 10 percent raise in science.

```
SELECT id,name,maths,science*1.1,English FROM tbl_Marks
```

2. **Comparison Operators:**

Comparison operators are used to select, update and delete the records in the relation based on condition specified in WHERE clause .

Comparison operators are =, <>, >, <, !=, ! >, !<, >= and <=.

E.g.

a. Display all the records from tbl_Marks for which marks in maths is greater than 60.

```
SELECT * from tbl_Marks WHERE maths > 60
```

b. Decrement the marks of English by 10 for student whose id is1001.

```
UPDATE tbl_Marks SET english=english-10 WHERE id=1001
```

c. Delete all the records from tbl_Marks whose marks in maths is less than 40.

```
DELETE FROM tbl_Marks WHERE maths<40
```

3. **Conjunctive Operators / Logical Operators:**

Logical operators are used to select, update and delete records based on one or more conditions. Logical Operators are AND, OR and NOT.

E.g.

a. Display all the records from tbl_Marks for which marks in maths is greater than 60 or marks in English is greater than 60.

```
SELECT * FROM tbl_Marks WHERE maths>60 OR english>50
```

b. Delete all the records of failed student from tbl_Marks. The pass mark for each subject is 40.

```
DELETE FROM tbl_Marks WHERE maths<40 OR english<40 or science<40
```

c. Delete all the records of students from tbl_Marks who are failed in all three subjects.

```
DELETE FROM tbl_Marks WHERE maths<40 AND English<40 AND science<40
```

4. **Range Operators:**

The range operators can be used to select, update and delete the records based on the condition specified by certain range. Range Operators are BETWEEN and NOT BETWEEN.

Syntax:

SELECT <.field1>, <field2>…. <fieldN>
FROM <table_name>
WHERE <Field_name>
BETWEEN <Expression1> AND <Expression2>

a. Display all the records from tbl_Marks whose id is between 200 to 300.

```
SELECT * FROM tbl_Marks WHERE id BETWEEN 200 AND 300  // 200 and 300 are inclusive
```

Or

```
SELECT * FROM tbl_Marks WHERE id>=200 AND  id<=300
```

b. Display all the records from tbl_Marks whose id in not in the range 200 to 300

```
SELECT * FROM tbl_Marks WHERE id NOT BETWEEN 200 AND 300  // 200 and 300 are inclusive
```

c. Delete all the records from tbl_Marks whose id is between 200 to 300

```
DELETE FROM tbl_Marks WHERE id BETWEEN 200 AND 300
```

5. **List Operators:**

The list operators are used to select, update and delete the records based on the condition specified by the list of values. List Operators are IN and NOT IN. The IN operator selects values that match any one values given in the list.

Syntax:

SELECT <field1>, <field2>…<fieldN> from <table_name> WHERE <Field_name> IN <item1,item2….itemN>

E.g.

a. Display all the records of students from tbl_Marks who are from Pokhara, Kathmandu or Butwal.

```
SELECT * FROM tbl_Marks WHERE address IN('Pokhara','Kathmandu','Butwal')
```

Or

```
SELECT * from tbl_Marks WHERE address='Pokhara' OR address='Kathmandu' OR address='Butwal'
```

b. **Display all the records from tbl_Marks whose id is not 20, 30, 45, 78 and 92**

```
SELECT * FROM tbl_Marks WHERE id NOT IN(20,30,45,78,92)
```

or

SELECT * FROM tbl_Marks WHERE id<>20 AND id<>30 AND id<>45 AND id<>78 AND id<92>

Or

SELECT * FROM tbl_Marks WHERE NOT(id=20 OR id=30 OR id=45 OR id=78 OR id=92)

6. **String Operator:**

SQL includes a string-matching operator for comparison on character strings. The Operator LIKE matches the given string with the specified pattern.

The pattern includes combination of wildcard characters and regular characters.

Some of the wild card characters are:

Wildcard	Description
%(Percent)	Represent any string of zero or more characters
_(Underscore)	Represent any single character
[]	Represent any single character within specified range
[^]	Represent any single character not within the specified range

E.g.

a. Display all the records from tbl_Marks for which name start with 'M'.

SELECT * from tbl_Marks WHERE name LIKE 'M%'

b. Display all the records from tbl_Marks for which name ends with 'ma'

SELECT * FROM tbl_Marks WHERE name LIKE '%ma'

c. Display all the records from tlb_Marks for which the address is three letters and ends with 'rt'.

SELECT * FROM tbl_Marks WHERE address LIKE '_rt'

d. Display all the records from tbl_Marks for which the name begins with 'a', 'x','y' or 'z'.

SELECT * FROM tbl_Marks WHERE name LIKE '[axyz]%'

e. Display all the records from tbl_Marks for which the name begins with any letter from 'p' to 'w' and ends with 'eta'

SELECT * FROM tbl_Marks WHERE name LIKE '[p-w]%eta'

f. Display all the records from tbl_Marks for which the name begins with 'd' and not having 'c' as the second letter and ends with 'a'.

SELECT * FROM tbl_Marks where name LIKE'd[^c]%a'

Sorting

Data can be sorted either in ascending or descending by using the ORDER BY clause

Syntax:

SELECT <field1>, <field2>……..<fieldN> FROM <table_name> ORDER BY <field_name> [ASC/DESC]

E.g.

a. Display all the records from tbl_Marks by sorting the names in descending order.

SELECT * FROM tbl_Marks ORDER BY name DESC

The Rename Operation

The SQL allows renaming the attributes as well as relation using the AS clause.

Syntax:

Select <field1>..<FieldN> From <table_name> as <new_table_Name> Where <NewFiled1> and <new_table_Name> are the aliase i.e. alternative name for <field1> and <table_name> respectively.E.g:

SELECT id AS student id, name, maths, science, English, (maths+science+english) AS total from tbl_Marks

Set Operations

The set operators combines results from two or more queries into a single result set. SQL support few set operations to be performed on table data. Different set Operators are

1. **UNION**:

It is used to combine the results of two or more SELECT statement. This operation eliminates the duplicate rows from its result set. The necessary condition to use this operation is the number of columns and data types must be same in both the table.

tbl_first

Id	Name
1	ram
2	hari

tbl_second

Id	Name
2	hari
3	ravi

The UNION query is

SELECT * FROM tbl_first UNION SELECT * FROM tbl_second

Result:

Id	Name
1	ram
2	hari
3	ravi

2. **UNION ALL:**

This operation is similar to UNION. But it also shows the duplicate rows

The UNION ALL query is

```
SELECT * FROM tbl_first UNION ALL SELECT * FROM tbl_second
```

Result:

Id	Name
1	ram
2	hari
2	hari
3	ravi

3. **INTERSECT:**

Intersect operation is used return the records which are common to both SELECT statement. This operation eliminates the duplicate rows from its result set.

INTERSECT ALL operation can be used if we want to keep duplicate rows in the result table.

The INTERSECT query is

```
SELECT * FROM tbl_first INTERSECT SELECT * FROM tbl_second
```

Id	Name
2	hari

4. **EXCEPT:**

Except operation returns all records from first table that are not in second table. This operation eliminate the redundant rows from its result set.

EXCEPT ALL operation can be used if we want to keep duplicate rows in the result table.

The EXCEPT query are

```
SELECT * FROM tbl_first EXCEPT SELECT * FROM tbl_second
```

Id	Name
1	ram

```
SELECT * FROM tbl_second EXCEPT SELECT * FROM tbl_first
```

Id	Name
3	ravi

Grouping and Summarizing Data

Summary of the data contains aggregated value that helps in data analysis at broader level.

We can summarize the data using the following **aggregate functions.**

1. AVG():

Returns the average values in a numeric expression.

E.g. Display the average math mark for the whole school.

```
SELECT AVG(maths) AS Averagemathsmarks FROM tbl_Marks
```

2. COUNT():

Return the numbers of values in an expression.

E.g. Display the total number of records in table tbl_Marks.

```
SELECT COUNT(*) AS total FROM tbl_Marks
```

or

```
SELECT COUNT(ID) AS total FROM tbl_Marks
```

Count the total number of unique name in tbl_mark

```
SELECT COUNT(DISTINCT name) from tbl_Marks
```

3. MIN():

Returns the lowest value in the numeric expression.

E.g. Display the lowest mark obtained in maths.

```
SELECT MIN(maths) AS lowestmathmark FROM tbl_Marks
```

4. MAX():

Returns the highest value in the numeric expression.

E.g. Display the highest mark obtained in maths.

```
SELECT MAX(maths) AS highestmathmark FROM tbl_Marks
```

5. SUM():

Returns the sum of values in a numeric expression.

E.g. Display the sum of marks in maths for all students.

```
SELECT SUM(maths) AS totalmathsmarksofallstudent FROM tbl_Marks
```

Grouping Data:

The **GROUP BY** clause summarizes the result set into groups as defined in the SELECT statement by using `aggregate function.

The GROUP BY clause is used to group a set of tuples having same value on given attribute. The attribute or attributes given in the GROUP BY clause are placed in one group.

The **HAVING** clause further restricts the result set to produce the data based on a condition. The HAVING clause is used to specify a selection condition on groups rather than on individual tuples.

Syntax:

SELECT <column_names>, aggregate_functinions(Column_name) FROM <table_name> WHERE <ExpressiGROUP BY <column_name> HAVING <Exprssion with/without aggregate function>

Example 1:

Consider the below Relation tbl_emp

Name	Address	Salary
Ram	Pkr	1000
Hari	Btl	8000
Shyam	Pkr	5000
Ram	Btl	4000
Sita	Pkr	12000
Rita	Btl	3000

a. Display the minimum and maximum salary paid for a employee of different addresses.

SELECT Address, MIN(Salary) AS minsalary,MAX(Salary) AS maxsalary FROM tbl_emp GROUP BY address

Address	Minsalary	maxsalary
Btl	3000	8000
Pkr	1000	12000

b. Display the average salary paid for all address except Btl.

SELECT Address, AVG(Salary) AS AverageSalary FROM tbl_emp WHERE Address<>'Btl' GROUP BY Address

Or

SELECT Address, AVG(Salary) AS AverageSalary FROM tbl_emp GROUP BY Address HAVING Address<>'Btl'

Address	AverageSalary
Pkr	6000

c. Display the maximum salary for each address where the maximum salary is greater than 10000

SELECT Address, MAX(Salary) AS maxsalary FROM tbl_emp GROUP BY address HAVING MAX(Salary)>10000

Address	maxsalary

Pkr	12000

Subquery / Inner query / Nested query

Subquery is a query that is used within another query i.e. query in query.

Subqueries are nested inside the WHERE clause of SELECT, INSERT, UPDATE and DELETE statement.

The query that represents the parent query is called outer query and the query that represents the subquery is called inner query.

The Database engine first executes the inner query and then outer query to calculate the result set.

Example 1:

Consider the tbl_emp relation

Name	Address	Salary
Ram	Pkr	1000
Hari	Btl	8000
Shyam	Pkr	5000
Ram	Btl	4000
Sita	Pkr	12000
Rita	Btl	3000

1. Display all the record of employee who have the salary greater than that of Rita

```
SELECT * FROM tbl_emp WHERE salary > (SELECT salary FROM tbl_emp WHERE name = 'rita').
```

Name	Address	Salary
Hari	Btl	8000
Shyam	Pkr	5000
Ram	Btl	4000
Sita	Pkr	12000

2. Display all the records of employee whose salary is greater than average salary of all employees

```
SELECT * FROM tbl_emp WHERE salary > (SELECT AVG(salary) FROM tbl_emp)
```

Name	Address	Salary

Hair	Btl	8000
Sita	Pkr	12000

3. Display all the records of employee who have same address as Rita

```
SELECT * FROM tbl_emp WHERE address = (SELECT address FROM tbl_emp WHERE name='rita'
```

Name	Address	Salary
Hari	Btl	8000
Ram	Btl	4000
Rita	Btl	3000

4. Display address for employes whose address is that of Rita or Sita.

```
SELECT address FROM tbl_emp WHERE address IN (SELECT address FROM tbl_emp WHERE name='Rita' OR name='Sita'
```

Address
Pkr
Btl
Pkr
Btl
Pkr
Btl

Consider the following relation.

tbl_mark(id,name,roll,address,math,science,english)

Write SQL for the following

1. Create above table structure

```
create table tbl_mark
(
	id int primary key,
	name nvarchar(50),
	roll int,
	address nvarchar(50),
	math float,
	science float,
```

```
        english float
)
```

2. Insert any five new records.

```
insert into tbl_mark values (1001,'Ram',1,'Pkr',44.5,68.9,54)
insert into tbl_mark values (1002,'Rina',1,'Pkr',84.5,67.7,99)
insert into tbl_mark values (1004,'Siva',1,'Pkr',34.7,43.7,69)
insert into tbl_mark values (1005,'Prabin',1,'ktm',67.9,77,67)
insert into tbl_mark values (1006,'Ram',1,'Pkr',84.5,67.7,99)
```

3. Display the minimum mark in math.

```
select MIN(math) as MathMin from tbl_mark
```

	MathMin
1	34.7

4. Display the name of students who have obtained highest mark in math.

```
select name from tbl_mark where math in (select MAX(math) from tbl_mark )
```

	name
1	Rina
2	Ram

5. Display the name and address of student who have obtained lowest mark in science.

```
select name, address from tbl_mark where science in (select MIN(science) from tbl_mark)
```

	name	address
1	Siva	Pkr

6. Display the numbers of student who are pass in all three subject. The pass mark for each subject is 40.

```
select COUNT(*) as totalpassstudent from tbl_mark where math>=40 and science >=40  and
                                                    english>=40
```

	totalpassstudent
1	5

7. Display the numbers of student who have obtained greater than average of all students mark in math.

```
select COUNT(*)as NoofStudents from tbl_mark where math> (select AVG(math) from tbl_mark)
```

	NoofStudents
1	3

8. Display the average mark in math for each address having average mark greater than 50.

```
select address,AVG(math) as average from tbl_mark group by address having AVG(math)>50
```

	address	average
1	ktm	61.95
2	Pkr	62.05

9. Display id, name, roll and address of the records of student whose mark in math is greater than average mark of math of their address.

```
select id,name,roll,address from tbl_mark t where math>(select AVG(math) from tbl_mark
                                    where address=t.address)
```

	id	name	roll	address
1	1005	Prabin	1	ktm
2	1006	Ram	1	Pkr
3	1002	Rina	1	Pkr

Null Values

With insertions, we saw how null values might be needed if values were unknown. Queries involving nulls pose problems.

If a value is not known, it cannot be compared or be used as part of an aggregate function. However, we can use the keyword **null** to test for null values:

Example:

```
select * from tbl_student where middlename is null
```

i.e. the above queries displays only those records for which the middle name is null.

	id	name
1	1001	Ram
2	1006	Ram

Join Operation

Join operation is done to retrieve records from multiple tables.

We can join more than one table based on a common attributes. Join operations take two or more relations as input and return a single relation as result. There are four types of join which are explained below.

1. **Inner join**:

When an inner join is applied, only rows with values satisfying the join condition in the common column are displayed. Records in both the tables that do not satisfy the join condition are not displayed.

Syntax:

SELECT <column_names>

FROM <table1>

INNER JOIN <table2> ON <table1>.<common_field> <join operator> <table2>.<common_field>

Notes: we can simply use JOIN keyword instead of INNER JOIN keyword in above statement because the inner join is the default join.

2. **Outer Join:**

The outer join displays the result set containing all the rows from one table and matching rows from another table. An outer join displays NULL for the columns of the related table where it doesn't find matching records.

An outer join is of following types.

a. **Left Outer join**:

A left outer join returns all the rows from the table specified on left side of LEFT OUTER JOIN keyword and the matching rows from the table specified on the right side. It displays NULL for the columns of the table specified on the right side where it doesn't find matching records.

Syntax:

SELECT <column_names>

FROM <table1>

LEFT OUTER JOIN <table2> ON <table1>.<common_field> <join operator>

<table2>.<common_field>

b. Right Outer join:

A right outer join returns all the rows from the table specified on the right hand side of the RIGHT OUTER JOIN keyword and the matching rows from the table specified on the left side. It displays NULL for the columns of the table specified on the left side where it doesn't find matching records.

Syntax:

SELECT <column_names>

FROM <table1>

RIGHT OUTER JOIN <table2> ON <table1>.<common_field> <join operator> <table2>.<common_field>

c. **Full Outer join:**

A full outer join is the combination of left outer join and right outer join which returns all the matching and non matcing rows from both the tables. In case of non matching rows, a NULL values is displayed for the columns for which the data is not available.

Syntax:

SELECT <column_names>

FROM <table1>

FULL OUTER JOIN <table2> ON <table1>.<common_field> <join operator> <table2>.<common_field>

E.g. Let us consider the following two relations.

tbl_filmname

filmId	Filmname	Yearmade
1	Chennai Express	2013
2	3-idiots	2012
4	Titanic	2008

tbl_actor

filmId	Firstname	lastname
1	Sharukh	Khan
1	Deepika	Padukon
2	Aamir	Khan
2	Karina	Kapoor
3	Salman	Khan
3	Katrina	Kaif

a. Join the two table tbl_filmname and tbl_actor using inner join.

SELECT * FROM tbl_filmname JOIN tbl_actor ON tbl_filmname.filmid=tbl_actor.filmid

Or

SELECT * FROM tbl_Filmname tf, tbl_actor ta where tf.filmid=ta.filmid

Filmid	Filmname	Yearmade	Filmid	Firstname	Lastname
1	Chennai Express	2013	1	Sharukh	Khan
1	Chennai Express	2013	1	Deepika	Padukon
2	3-idiots	2012	2	Aamir	Khan
2	3-idiots	2012	2	Karina	Kapoor

b. Join the two table tbl_filmname and tbl_actor using left outer join.

SELECT * FROM tbl_filmname LEFT OUTER JOIN tbl_actor ON tbl_filmname.filmid=tbl_actor.filmid

Filmid	Filmname	Yearmade	Filmid	Firstname	lastname
1	Chennai Express	2013	1	Sharukh	Khan
1	Chennai Express	2013	1	Deepika	Padukon
2	3-idiots	2012	2	Aamir	Khan

2	3-idiots	2012	2	Karina	Kapoor
4	Titanic	2008	NULL	NULL	NULL

c. Join the two table tbl_filmname and tbl_actor using Right outer join.

SELECT * FROM tbl_filmname RIGHT OUTER JOIN tbl_actor on tbl_filmname.filmid=tbl_actor.filmid

Filmid	Filmname	Yearmade	Filmid	Firstname	lastname
1	Chennai Express	2013	1	Sharukh	Khan
1	Chennai Express	2013	1	Deepika	Padukon
2	3-idiots	2012	2	Aamir	Khan
2	3-idiots	2012	2	Karina	Kapoor
NULL	NULL	NULL	3	Salman	Khan
NULL	NULL	NULL	3	Katrina	Kaif

d. Join the two table tbl_filmname and tbl_actor using full outer join.

SELECT * FROM tbl_filmname FULL OUTER JOIN tbl_actor on tbl_filmname.filmid=tbl_actor.filmid

Filmid	Filmname	Yearmade	Filmid	Firstname	lastname
1	Chennai Express	2013	1	Sharukh	Khan
1	Chennai Express	2013	1	Deepika	Padukon
2	3-idiots	2012	2	Aamir	Khan
2	3-idiots	2012	2	Karina	Kapoor
4	Titanic	2008	NULL	NULL	NULL
NULL	NULL	NULL	3	Salman	Khana
NULL	NULL	NULL	3	Katrina	Kaif

3. **Cross join:**

A cross join also known as Cartesian product joins each row from one table with each row on the other table.

If P is the number of rows in first table and Q be in second table, then the total number of rows in the result set is P *Q.

Syntax:

SELECT <column_names> FROM <table1> CROSS JOIN <table2>

4. **Equi join:**

A equi join is similar to the inner join with joining condition which is based on the equality between values in the common columns.

Q. How eui-join is different from inner join?

The difference is, equi join is used to retrieve all the columns from both the tables but inner join is used to retrieve selected columns from tables. So using this join operation results a common column to appear redundantly in the resultant table.

Stored Procedure

When a batch of SQL statements needs to be executed more than once, we need to recreate SQL statement and submit them to the server. This leads to an increase in the overhead, as the server needs to compile and create the execution plan for these statements again. Therefore, if we need to execute a batch multiples times, we can save it within a stored procedure. A stored procedure is a **precompiled** object stored in a database data dictionary.

A stored procedure is also called as proc, sproc or sp and is similar to the user defined functions. Stored procedure can invoke both DDL and DML statements and can return values.

Advantages:

1. Faster
2. Better Security because calls are parameterized.

Disadvantages:

1. Less flexible because we need DBA to make changes.

Creating Stored Procedure

We can create stored procedure using CREATE PROCEDURE statement.

Syntax:

```
CREATE PROCEDURE <procedure_name> AS
BEGIN
        Sql_statement1
        Sql_statement2
        .
        Sql_statementN
END
```

E.g.

```
CREATE PROCEDURE sp_mobile AS
BEGIN
    SELECT * FROM tbl_mobile
END
```

When the CREATE PROCEDURE command is executed, the server compiles the procedure and save it as a database object. The procedure is then available for various applications to execute.

Executing a Stored Procedure

A procedure can be executed using EXECUTE or EXEC statement.

Syntax: EXECUTE | EXEC <Procedure_name>

E.g. EXEC sp_mobile

Altering Stored Procedure

A stored Procedure can be modified by using ALTER PROCEDURE statement.

Syntax:

```
ALTER PROCEDURE <Procedure_name> AS
BEGIN
    Sql_statement1
    Sql_statement2
    .
    Sql_statementN
END
```

Dropping a stored Procedure

We can destroy a stored procedure form database by using DROP PROCEDURE statement.

Syntax: DROP PROCEDURE <Procedure_name>

Creating a Parameterized Stored Procedure

A parameterized stored procedure is necessary when we need to execute a procedure for different values of a variable that are provided at runtime. Parameters are used to pass values to stored procedure during run time. Each parameter has a name and its data type.

Syntax:

```
CREATE PROCEDURE <Procedure_name>
    @parameter1 data type1,
    @parameter2 data type2,
    .
```

.

@parameterN data typeN

As

BEGIN

Sql_statement1

Sql_statement2

.

Sql_statementN

END

E.g.

```
CREATE PROC sp_mob
    @mobile_type varchar(50)
AS
BEGIN
    SELECT * FROM tbl_mobile WHERE mobile_type=@mobile_type
END
```

Query By Example (QBE)

QBE is a graphical query language which is based on the domain relational calculus.

A user needs minimal information to get Started and the whole language contains relatively few concepts. QBE is especially suited for queries that are not too complex and can be expressed in terms of a few tables.

QBE provide a tabular interface that has expressive power of relational calculus in a user friendly form i.e queries look like table.

Here queries are expressed by 'example'.

E.g. Let us consider the following three relations.

Sailors (sid, sname, age)

Boats(bid, bname,color)

Reserve(sid,bid,day)

1. Create example table for printing name and age of all sailors.

Sailors	Sid	Sname	age
		P._N	P._A

In the above table, _N and _A are variables. The use of such variable is optional i.e. we can just write P. (P. mean print).

Duplicate values are removed by default. To retain duplicate use P.All.

2. Display all the records of sailor.

Sailors	Sid	Sname	age
	P.	P.	P.

Or

Sailors	Sid	Sname	age
P.			

3. Display all the records of sailor whose age is equal to 50.

Sailors	Sid	Sname	age
P.			60

Placing a constant 60 means, placing a condition =60. We can use other comparison operations (<, >,<=,>=, ¬) as well.

4. Display all the records of Sailors in ascending order of age.

Sailors	Sid	Sname	age
P.			AO

P.AO means ascending order and P.DO means Descending order.

Sailors	Sid	Sname	age
	P.	P.	P.AO

Solved Examples:

Write the Relation algebra, SQL comma for the following task.

Loan(loannumber, branchname, amount)

Customer(customername,street,city)

Borrower(customername,loannumber)

Depositer(customername,accountnumber)

1. Find the loan number of each loan of amount greater than Rs 1200.

Relational Algebra: $\prod_{loannumber}(\sigma_{amount > 1200}(Loan))$

SQL; SELECT loannumber FROM Loan WHERE amount > 1200

2. Find all the name of customer who have loan, account or both.

Relational Algebra: $\prod_{customername}(Borrower) \cup \prod_{customername}(Depositer))$

SQL: SELECT customername FROM Borrower UNION SELECT customername FROM Depositer

3. Find all the name of customer who have loan and account at bank.

Relational Algebra: $\prod_{customername}(Borrower) \cap \prod_{customername}(Depositer))$

SQL: SELECT customername FROM Borrower INTERSECT SELECT customername FROM Depositer

4. Find customers having an account, but not a loan.

(SELECT cname from depositor) except (SELECT cname from borrower)

5. Find all customer having an account

SELECT customername FROM Depositer

2) Given relation schema as below

employee(e_name(pk), street city)

work(e-name(pk), companyname(pk), salary)

company(companyname(pk),city)

manages(empname(pk),manager-name)

Write the SQL for the following.

1. Modify the database so that Ram now lives in Kathmandu.
   Update employee set city='Kathmandu' where name='Ram'
2. Give all employee of first bank corporation a 10 percent raise
   Update work set salary=salary * 1.1 where companyname = 'first bank corporation'
3. Delete all tuple in work relation for employee of small bank corporation.
   Delete from work where companyname='small bank corporation'
4. Find all employee who earn more than average salary of all employee of their company.
   select e.name from work e where salary >(select avg(salary) from work e1 where e1. company_name= e.company_name)

   or

   select e.name from work e where salary >(select avg(salary) from work where company_name= e.company_name)

3) Given relation schema as below

Employe(emp-id, name, address, telephone, salary, age)

Works- on (emp-id, project-id, join-date)

Project(project-id, project-name, city,duration, budget)

Write the sql commands for the following.

1. Insert new record in project relation.
   INSERT into Project VALUES(201,'DBMS Projects','Kathmandu')

2. Find the name of employees with the name of project they work on.

3. Find the name and city of that project on which salary of employee is greater than or equal to 20000.
   SELECT project-name, city FROM Employee e, Works-on, Project P WHERE e.emp-id=w.emp-id and w.project-id=p.project-id and salary>=20000

   OR

   SELECT project-name, city from Employee e JOIN Work-on w ON e.emp-id=w.emp-id JOIN Project p ON w.-project-id=p.project-id WHERE salary>=20000

4. List the name of employees whose name starts with "m" and ends with "a"
   SELECT name FROM Employee WHERE name LIKE 'm%a'

5. Find the employee name and project name of those employees who living in address Pokhara.

SELECT name, project-name FROM Employee e, Work-on w, Project p WHERE e.emp-id=w.emp-id and w.project-id=p.project-id and address='Pokhara
'

6. List name of employee whose age is greater than average age of all employees.

SELECT name FROM Employee WHERE age>(SELECT AVG(age) FROM EMPLOYEE)

7. List employee id of all employees whose age is greater than minimum budget of all projects.

SELECT emp-id FROM Employee age > (SELECT MIN(budeget) FROM PROJECT)

Or

SELECT emp-id FROM Employee age> any(SELECT budget FROM PROJECT)

8. List employee id of all employees who joint project on "05/01/2015"

SELECT emp-id FROM Works-on where join-date='05/01/2015'

9. List the name of employees whose name starts with N or with K

SELECT name FROM Employee WHERE name LIKE 'N%' OR name LIKE 'K%'

10. Display the project id with maximum budget.

SELECT project-id, budget FROM Project WHERE budget = (SELECT MAX(budget) FROM Project)

4) Given relation schema as below

Doctor(name, age , address)

Works(name, depart_no)

Department(depart no, floor, room)

Write the SQL for the following.

1. Display the records of doctor with their department information.

SELECT * FROM Doctor d, Works w, Department d WHERE d.name=w.name and w.depart _no=d.depart_no

2. Display the number of room in each floor

SELECT floor, COUNT(room) FROM Department GROUP BY floor.

3. Display the name of doctor who works in at least one department.

SELECT name FROM Department d, Works w where d.name=w.name

OR

```
SELECT name FROM Doctor d where exists (SELECT * FROM Works w WHERE d.name= w.name)
```

4. Display the name of doctor who do not work in any department.

```
(select name from Doctor) except (SELECT d.name from Doctor d, Works w WHERE d.name
    =w.name)

OR

SELECT name FROM Doctor d where not exists (SELECT * FROM Works w WHERE d.name= w.name)

Or

SELECT name FROM DOCTOR where name NOT IN (SELECT name FROM Works)

OR

SELECT Doctor.name FROM Doctor LEFT OUTER JOIN Works on Doctor.name = Works.name where
works.name IS NULL

OR

SELECT d.name FROM Doctor d LEFT OUTER JOIN Works w on d.name = w.name where w.name IS
NULL
```

5. Display the depart no of department where no one works.

SELECT depart_no FROM Department EXCEPT SELECT depart_no FROM WORKS

OR

SELECT depart_no FROM department WHERE depart_no NOT IN (SELECT depart_no FROM Works)

OR

SELECT Department.depart_no FROM Works RIGHT OUTER JOIN Department on Works.depart_no=Department.depart_no where name is null

OR

Select depart_no FROM Department d where not exists (select * from Works w where w.depart_no=d.depart_no)

6. Display the name of doctor who works in at least two department

```
SELECT d.name, COUNT(*) FROM Doctor d, Works w where d.name=w.name group by d.name having
COUNT(*)>=2
```

7. Display the name of doctor with maximum age.

SELECT name,age FROM DOCTOR WHERE age=(SELECT MAX(age) FROM Doctor)

8. Delete the records of those doctors whose name start with “m” and works in 10 th floor

Delete FROM Doctor where name in (Select d.name from doctor d,works w, department de where d.name=w.name and w.depart_no=de.depart_no AND d.name LIKE 'm%' AND FLOOR='10th')

OR

Delete d From doctor d,works w, department de where d.name=w.name and w.depart_no=de.depart_no AND d.name LIKE ‘m%' AND FLOOR='1oth'

Note: we must specify the table from which record to be deleted using alias

9. Update database such that Ram now lives in Kathmandu.

UPDATE doctor SET address='Kathmandu' WHERE name='Anju'

OR

UPDATE d SET address='Pokhara' FROM Doctor d WHERE name='Anju'

10. Update database such that Ram now work in room no 11.

UPDATE de SET room=11 FROM Doctor d, Works w, Department de WHERE d.name=w.name AND w.depart_no=de.depart_no AND d.name='Ram'

Q. Consider a simple relational database of Hospital Management System. (*Underlined attributes represent Primary key attributes*)

Doctors (DoctorID, DoctorName, Department, Address, Salary)

Patients (PatientID, Patent Name, Address, Age, Gender)

Hospitals (PatientID, Doctor ID, HostpitalName, Location)

Write Down the SQL statement for the following.

i. Display ID of Patient admitted to hospital at Pokhara and whose name ends with ‘a’.

SELECT p.PatientID FROM Patients p, Hospitals h WHERE p.PatientID=h.PatientID AND location=’Pokhara’ AND PatientName LIKE ‘%a’

ii. Delete the record of Doctors whose salary is greater than average salary of doctors.

DELETE FROM Doctors WHERE Salary >(SELECT AVG(Salary) FROM Doctors)

iii. Increase the salary of doctors by 18.5% who works in OPD department.

UPDATE Doctors SET Salary=Salary*1.185 WHERE Department=’OPD’

iv. Find the average salary of Doctors for each address who have average salary more than 55K.

SELECT Address, AVG(Salary) FROM Doctors GROUP BY Address HAVING AVG(Salary)>55000

Q. Given relation schema as below

employee(e_name(pk), street city)

work(e-name(pk), companyname, salary)

company(companyname(pk),city)

manages(empname(pk),manager-name)

Write the SQL for the following.

5. Modify the database so that Ram now lives in Kathmandu.
   Update employee set city='Kathmandu' where name='Ram'
6. Give all employee of first bank corporation a 10 percent raise
   Update work set slary=salary * 1.1 where companyname = 'first bank corporation'
7. Delete all tuple in work relation for employee of small bank corporation.
   Delete from work where companyname='small bank corporation'
8. Find all employee who earn more than average salary of all employee of their company.
   select e.name from work e where salary >(select avg(salary) from work e1 where e.company_name= e1.company_name)

Q. Given Relational Schema as below.

Sailors (sid, sname, age,rating)

Boats(bid, bname,color)

Reserve(sid,bid,day)

Write the sql for the following.

1. **Find the sid of the sailor who have reserved red or green boat.**
   SELECT S.sid FROMSailors S, Boats B, Reserves R WHERE S.sid=R.sidANDR.bid=B.bid AND(B.color='red'ORB.color='green')

2. **Find sid's of sailors who've reserved a red and a green boat.**

SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERES.sid=R.sidANDR.bid=B.bid ANDB.color='red' INTERSECT SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERES.sid=R.sidANDR.bid=B.bid ANDB.color='green'

3. **Find names of sailors who have reserved boat #103:**
   SELECT S.sname FROMSailors S WHERE EXISTS (SELECT* FROM Reserves R WHERE R.bid=103 AND S.sid=R.sid)
4. **Find sailors who've reserved all boats**

SELECTS.sname FROM Sailors S WHERE NOT EXISTS (
(SELECT B.bid FROM Boats B) EXCEPT (SELECT R.bid FROM Reserves R WHERE R.sid=S.sid)
)

5. Find name and age of the oldest sailor(s) with rating >
   SELECTS.sname, S.age FROMSailors S WHERES.Rating> 7 AND S.age= (SELECT MAX (S2.age) FROMSailors S2
   WHERES2.Rating > 7)
6. Find the age of the youngest sailor for each rating level
   SELECT S.rating, MIN(S.Age) FROM Sailors S GROUP BY S.rating
7. Find the age of the youngest sailor with age >= 18 for each rating level with at least 2 suchsailors
   SELECT S.rating,MIN(S.Age) FROMSailors S WHERES.age>= 18 GROUP BY S.rating HAVING
   COUNT(*)>= 2
8. Find the average age for each rating, and order results in ascending order on avg. ag
   SELECTS.rating, AVG(S.age) ASavgage FROMSailors S GROUP BY S.rating ORDER BYavgage asc

5) Consider the following Relational Schema

$$Branch\text{-}scheme = (bname,\ bcity,\ assets)$$
$$Customer\text{-}scheme = (cname,\ street,\ ccity)$$
$$Depositor\text{-}scheme = (cname,\ account\#)$$
$$Account\text{-}scheme = (bname,\ account\#,\ balance)$$
$$Loan\text{-}scheme = (bname,\ loan\#,\ amount)$$
$$Borrower\text{-}scheme = (cname,\ loan\#)$$

1. Find the name of all branch in account relation.
   SELECT bname FROM Account
2. Finds the names of all branches that have assets greater than at least one branch located in Burnaby
   SELECT bname FROM Branch WHERE assets> Some(SELECT assets FROM Branch WHERE bcity='Burnaby')
   OR
   select distinct T.bname from branch S, branch T where S.bcity='Burnaby' and T.assets > S.assets
3. Find all customers whose street includes the substring "Main".
   SELECT cname FROM Customer WHERE street like '%Main%'
4. Finds all customers who have a loan and an account at the SFU branch.

SELECT distinct cname FROM Burrower WHERE cname in (SELECT cname FROM Account WHERE bname='SFU')

5. Find the name and loan number of all customers who have a loan at SFU branch.
   SELECT DISTINCT cname , b.loan# FROM Burrower b, Loan l WHERE b.loan#=l.loan# AND bname='SFU'
6. Finding all customers who have a loan but not an account at the SFU branch.
   SELECT distinct cname FROM Burrower WHERE cname not in (SELECT cname FROM Account WHERE bname='SFU')
7. Find the branches whose assets are greater than some branch in Pokhara.
   SELECT bname FROM Branch WHERE assets > SOME(SELECT assets FROM branch where bcity='Pokhara')
8. Find all customers who have a loan and an account at the bank.
   SELECT cname FROM Borrower INTERSECT SELECT cname FROM Depositer

OR

SELECT cname FROM BURROWER b WHERE exists (SELECT * FROM Depositer d WHERE b.cname=d.cname)

OR

SELECT distinct cname FROM Burrower WHERE cname in (SELECT cname FROM Account)

6) Given relation schema as below

Student(sid, sname, level, age, sex)

Instructor(iid, iname, age, sex)

Course(cid, cname, credit_hour)

Enrolledby(sid, cid)

Taughtby(iid, cid)

Write the SQL statements for the following

1. Find the id and name of student who enrolled in course "cmp-101"
   SELECT id, name FROM Student s, Enrolledby e WHERE s.sid=e.sid AND cid='cmp-101'

OR

SELECT id,name FROM Student s JOIN Enrolledby e ON s.sid=e.sid WHERE cid='cmp-101'

2. Find the id, name and level of all students who enrolled in course "DSA"
   SELECT id, name, level FROM Student s, Enrolledby e, Course s WHERE s.sid=e.sid AND e.cid=c.cid AND cname='DSA'

3. Find id, name and age of student by increasing age by 5.
   SELECT sid, sname, age+5 FROM Student.
4. Find id and name of instructor who do not teach any subject.
   SELECT i. iid, i. iname FROM Instructor i LEFT OUTER JOIN taughtby t ON i.iid=t.iid WHERE cid is NULL

5. Find all courses record that is not enrolled by any student.

   SELECT cid, cname, credit_hour FROM Enrolledby e RIGHT OUTER JOIN Course c ON e.cid=c.cid WHERE sid IS NULL

6. Find id and name of all instructors who teaches at least one subject taught by instructor 331.

   SELECT i.iid, i. iname FROM Instructor i, Taughtby t WHERE i.iid=t.iid AND cid IN(SELECT cid FROM Taughtby WHERE iid=331)

7. Find id and name of all instructors who do not teaches any subject taught by instructor 331.

   SELECT i. iid, i. iname FROM Instructor i, Taughtby t WHERE i.iid=t.iid AND cid NOT IN(SELECT cid FROM Taughtby WHERE iid=331)

8. Find name of all students whose level is same as the level of 'Bipin'

   SELECT name FROM Student WHERE level =(SELECT level FROM Student WHERE sname='Bipin')

9. Find id, name and age of all instructors whose age is greater than age of all students.

   SELECT iid, iname, age FROM Instructor WHERE age > ALL (SELECT age FROM Student)

10. Find id, and name of all students whose age is greater than that of at least one instructor

    SELECT sid, sname FROM Student WHERE age >SOME(SELECT age FROM Instructor)

11. Find id, name and enrolled course of all students whose age is greater than age of at least one instructor.

    SELECT sid, sname,cid FROM Student s LEFT OUTER JOIN Enrolledby e ON s.sid=e.sid WHERE age>SOME(SELECT age FROM Instructor)

12. Find id, name and level of all students who are instructor also.

    SELECT sid, sname,level FROM Student s WHERE EXISTS(SELECT * FROM Instructor i WHERE s.sid= i. iid)

    OR

    SELECT sid, sname, level FROM Student where sid in (SELECT iid FROM Instructor)

13. Find id, name and level of all students who are not instructors.

    SELECT sid, sname,level FROM Student s WHERE NOT EXISTS(SELECT * FROM Instructor i WHERE s.sid= i. iid)

    OR

    SELECT sid, sname, level FROM Student where sid NOT in (SELECT iid FROM Instructor)

14. Find minimum and maximum age of students.

    SELECT MIN(age), MAX(age) FROM Student

15. Find average age of all undergraduate students.

    SELECT AVG(age) FROM Student WHERE level='undergraduate'

16. Find total number of courses.

    SELECT COUNT(*) FROM Course

17. Find number of courses enrolled by student 102

    SELECT COUNT(*) FROM Course c, Enrolledby e WHERE c.cid=e.cid and sid=102

18. Find average age for all level of students.

    SELECT level, AVG(age) FROM Student GROUP by Level

19. Find number of courses enrolled by each student.

    SELECT sid, count(*) FROM Enrolledby GROUP BY sid

20. Find number of courses taught by male instructors.

    SELECT COUNT(*) FROM Taughtby t, Instructor i WHERE t.iid=i.iid AND sex='Male'

21. Find average age for all levels of students that have average age more than 22.

    SELECT level, AVG(age) FROM Student GROUP BY Level HAVING AVG(age)>22

22. Find number of courses enrolled by each student who have enrolled more than two courses.

    SELECT sid, COUNT(*) FROM Student s, Enrolledby e WHERE s.sid=e.sid GROUP BY sid HAVING COUNT(*)>2

23. Find number of courses taught by each male instructor who have taught more than two courses.

    SELECT iid, COUNT(*) FROM Taughtby t, Instructor i WHERE t.iid=i.iid AND sex='male' GROUP BY iid
    HAVING COUNT(*)>2

24. Insert new instructor record into database.

    INSERT INTO Instructor VALUES(212,'Ram', 34, 'Male')

25. Insert record of all graduate level, female students into instructor table.

    INSERT INTO Instructor(iid, iname,sex,age) (SELECT sid, sname,sex,age FROM Student WHERE level='Graduate'
    AND sex='Female')

    OR

    INSERT INTO Instructor (SELECT sid,sname, sex,age FROM Student WHERE level='Graduate' AND sex='Female')

26. Insert record of student who is enrolled in courses cmp-301 into instructor table.

    INSERT INTO Instructor (SELECT sid, sname, age, sex FROM STUDENT s, Enrolled e WHERE s.sid=e.sid AND
    cid='cmp-301')

27. Modify age of student 105 to 27

    UPDATE Student SET age=27 WHERE sid=105

28. Change name and age of instructor 301 to 'bipin' and 30 respectively.

    UPDATE Instructor SET age=30, iname='bipin' WHERE iid=301

29. Increase credit hour of course by 2 whose credit hour is more than 3 otherwise increase credit hour by 1

    UPDATE Course SET credit_hour =credit_hour+2 WHERE credit_hour>3

    UPDATE Course SET credit_hour=credit_hour + 1 WHERE credit_hourA<=3

30. Delete record of courses whose credit hour is 2

    DELETE FROM Course WHERE credit_hour=2

31. Delete student records that do not enrolled any courses.

    DELETE FROM Student WHERE sid NOT IN(SELECT sid FROM Enrolledby)