CHAPTER - 4

RELATIONAL DATABASE QUERY LANGUAGES

4.1 Introduction to SQL

SQL stands for Structured Query Language pronounced as "S-Q-L" or sometimes as "See-Quel". SQL is the standard language for accessing and manipulating Relational Databases. It enables a user to create, read, update and delete relational databases and tables. It is a practical implementation of relational algebra and relational calculus. SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987 Although SQL is an ANSI/ISO standard, there are different versions of the SQL language. Relational databases like MySQL Database, Oracle, MS SQL Server, Sybase, etc. use ANSI SQL.

Example of SQL:

1. **Find the name of the customer with customer_id 7465 from customer table.**

ANS:
```
SELECT      customer. customer_name
FROM        customer
WHERE       customer.customer_id = 7465;
```

2. **Find the balances of all accounts held by the customer with customer_id 7465**

ANS:
```
SELECT account.balance
FROM depositor inner join account
on depositor.account_number = account.account_number
WHERE depositor.customer_id = 7465;
```

4.1.1 Features of SQL

Following are the features of SQL:

- SQL is easy to learn.
- The SQL Keywords are case-insensitive (SELECT, FROM, WHERE, etc.) but are often written in all caps

- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

4.1.2 SQL Basic Data Types

- **char(n)**
  A fixed-length character string with user-specified length n. Full form character(n) can be used instead. Maximum length is 8000 characters.
- **varchar(n)**
  A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent. Maximum length is 8000 characters
- **int**
  An integer (a finite subset of the integers that is machine dependent). The full form, integer is equivalent.
- **Smallint**
  Allows whole numbers between -32768 and 32768
- **float(n)**
  A floating-point number, with precision of at least n digits.
- **numeric(p, d)**
  A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, numeric (3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

- **decimal (p, d)**
  Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. p must be a value from 1 to 38. Default value of p is 18. The d parameter indicates the maximum number of digits stored to the right of the decimal point. d must be a value from 0 to p. Default value is 0.

4.1.3 Queries and Sub Queries

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. The categories of query language are:

1. **Procedural**
   User tells system what and how to do.
2. **Non-procedural**
   User tells system what to do.

Relational algebra, tuple relational calculus, domain relational calculus are the pure languages that form underlying basis of query languages such as SQL.

Basic Structures of SQL Queries

Queries written in SQL (Structured Query Language) comply to a particular syntax. Multiple clauses are joined to form the fundamental structure of SQL queries, which are used to obtain data from databases. An outline of the typical structure is provided here:

- **SELECT:** The columns or expressions to obtain from the database table are specified in this clause. It chooses the information that the query will return.
- **FROM:** The table or tables from which the data will be fetched are specified in this clause. It indicates where the data came from.
- **WHERE:** We may specify conditions in this optional clause to filter the data based on certain criteria. By utilizing logical expressions and comparison operators, it assists in reducing the number of outcomes.

- **GROUP BY:** The data are grouped using this clause according to one or more columns. To do computations on aggregated data, it is frequently used in conjunction with aggregate functions like SUM, COUNT, AVG, etc.
- **HAVING:** The data that has been grouped is further filtered using this optional clause. Similar to the WHERE clause, it applies conditions to the grouped data but works with aggregated results.
- **ORDER BY:** The result set can be sorted using this clause in either ascending or descending order depending on one or more columns. It aids in the meaningful organization of the data.
- **LIMIT/OFFSET:** By using these optional clauses, we can limit how many rows the query returns. While OFFSET skips a predetermined number of rows before beginning to return the result set, LIMIT defines the maximum number of rows to return.
- **JOIN:** Using a related column as a basis, this clause is used to merge rows from two or more tables. By creating relationships, we can obtain data from several tables.

These clauses can be combined and customized to construct complex SQL queries that retrieve, filter, sort, and aggregate data from databases.

Example:

i.
```
SELECT first_name, last_name, age
FROM employees
WHERE department = 'Sales'
ORDER BY last_name ASC;
```

ii.
```
SELECT employees.first_name, departments.department_name
FROM employees
JOIN departments ON employees.department_id = departments.department_id;
```


Subquery

A subquery is a query within another SQL query and embedded within the WHERE clause. Subquery can be used with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, < , >=, <=, IN, BETWEEN, etc. It can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause. Subqueries are on the right side of the comparison operator. It is enclosed in parentheses.

Example of subquery:

Example 1:

```
SELECT * FROM Employee
WHERE ID IN (SELECT ID FROM Employee WHERE Salary > 10000);
```

Example 2:

```
UPDATE Employee
SET Salary = Salary * 0.25
WHERE Age IN (SELECT Age FROM Customers_BKP WHERE Age >= 29);
```

4.1.4 Set Operations

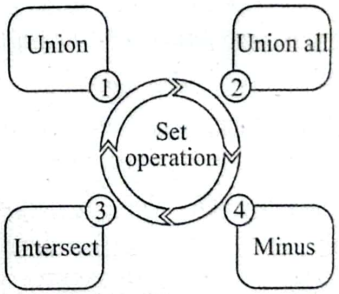

Figure: Set Operations

In SQL, set operation is used to combine two or more SQL SELECT statements. They allow the results of multiple Select queries to be combined into a single result set. Types of set operation are:

1. Union
2. Union All
3. Intersect
4. Minus/Except/Difference

1. Union

It is used to combine the result of two or more SQL SELECT queries. To perform this operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied. It eliminates the duplicate rows from the combined result set.

Syntax:

```
SELECT column_name FROM table1
  UNION
  SELECT column_name FROM table2;
```

Representation:

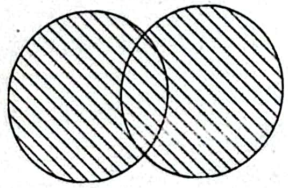

Figure: Pictorial representation of UNION operation

Example:

```
SELECT * FROM sales2005
  UNION
  SELECT * FROM sales2006;
```

sales2005

Person	Amount
Joe	1000
Alex	2000
Bob	5000

sales2005

Person	Amount
Joe	2000
Alex	2000
Zach	35000

Result:

Person	Amount
Joe	1000
Alex	2000
Bob	5000
Joe	2000
Zach	35000

2. Union All

It is almost similar to UNION operator except it preserves the duplicate rows from the result set.

Syntax:

```
SELECT column_name FROM table1
  UNION ALL
  SELECT column_name FROM table2;
```

Representation:

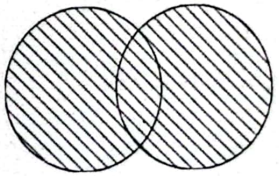

Figure: Pictorial representation of UNION ALL operation

Example:

```
SELECT * FROM sales2005
UNION All
SELECT * FROM sales2006;
```

sales 2005

Person	Amount
Kamal	1000
Bipin	2000
Arjun	5000

sales 2006

Person	Amount
Kamal	2000
Bipin	2000
Dilnath	35000

Result:

Person	Amount
Kamal	1000
Kamal	2000
Bipin	2000
Bipin	2000
Arjun	5000
Dilnath	35000

Here in this example, Alex 2000 are the duplicate rows.

3. Intersect

The Intersect operation returns the common rows from both the SELECT statements. The number of datatype and columns must be the same. Here duplicate rows are removed and result set is arranged in ascending order by default.

Syntax:

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Representation:

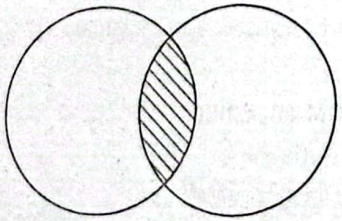

Figure: Pictorial representation of INTERSECT operation

Example:

```
SELECT * FROM Orders
INTERSECT
SELECT * FROM Sales;
```


Orders

id	items
1	Phone
2	Laptop
3	Camera

Sales

id	items
2	Laptop
3	Camera
4	Watch

Result

id	items
2	Laptop
3	Camera

4. Minus/ Except/ Difference

Minus operator is used to display the rows which are present in the first query but absent in the second query. It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Representation:

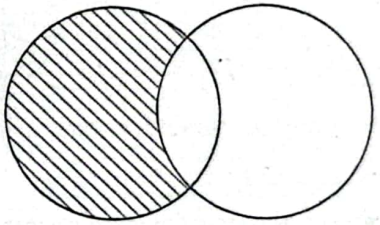

Figure: Pictorial representation of MINUS operation

Example:

```
SELECT * FROM Orders
MINUS
SELECT * FROM Sales;
```

Orders

id	items
1	Phone
2	Laptop
3	Camera

Sales

id	items
2	Laptop
3	Camera
4	Watch

Result

id	items
1	Phone

4.1.5 Joins

A SQL join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of SQL joins are:

1. INNER JOIN
2. OUTER JOIN
   a. LEFT JOIN
   b. RIGHT JOIN
   c. FULL JOIN

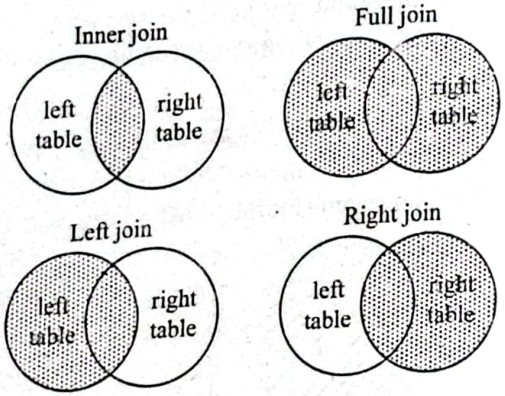

Figure: Different types of SQL join

1. Inner Join

Inner join keeps only the information from both tables that's related to each other. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join condition.

Syntax:

```
SELECT table1.column1, table2.column2 ....
FROM table1 INNER JOIN table2
ON
table1.common_field = table2.common_field;
```


Example:

```
SELECT *
FROM table1 t1 INNER JOIN table2 t2
ON
t1.value1 = t2.value2;
```

table1

Key1	Value1
1	a
2	b
3	c
4	d
5	f

table2

Key2	Value2
5	a
6	b
7	c
8	d
9	e

Result

Key1	Value1	Key2	Value2
1	a	5	a
2	b	6	b
3	c	7	c
4	d	8	d

Self join

It involves the joint operation with the table itself. It is basically used where there is any relationship between rows stored in the same table.

Example:

```
SELECT e1.Name , e2.Name AS Boss
FROM empp e1 INNER JOIN empp e2
ON
e1.Boss_id = e2.Id;
```

Empp

Id	Name	Boss_id
1	Mahesh	3
2	Manoj	1
3	Sushil	2

Result

Name	Boss
Mahesh	Sushil
Manoj	Mahesh
Sushil	Manoj

2. Outer Join

An outer keeps the information that is not related to the other table in the resulting table. Following are the types of outer join:

a. Right (Outer) Join

It selects records from the second (right-most) table with matching left table records.

Syntax:

```
SELECT table1.column1, table2.column2 ....
FROM table1 RIGHT JOIN table2
ON
table1.common_field = table2.common_field;
```

Example:

```
SELECT *
FROM table1 RIGHT JOIN table2
ON
table1.value1 = table2.value2;
```

table1

Key1	Value1
1	a
2	b
3	c
4	d
5	f

table2

Key2	Value2
5	a
6	b
7	c
8	d
9	e

Result

Key1	Value1	Key2	Value2
1	a	5	a
2	b	6	b
3	c	7	c
4	d	8	d
(NULL)	(NULL)	9	e

b. Left (Outer) Join

It selects all the records from left table with matched records from right table.

Syntax:

```
SELECT table1.column1, table2.column2 ......
FROM table1 LEFT JOIN table2
ON
table1.common_field = table2.common_field;
```

Example:

```
SELECT *
FROM table1 LEFT JOIN table2
ON
table1.value1 = table2.value2;
```

table1

Key1	Value1
1	a
2	b
3	c
4	d
5	f

table2

Key2	Value2
5	a
6	b
7	c
8	d
9	e

Result

Key1	Value1	Key2	Value2
1	a	5	a
2	b	6	b
3	c	7	c
4	d	8	d
5	e	(NULL)	(NULL)

c. Full (Outer) Join

It selects all records that match either left or right table records. MySQL does not support FULL JOIN, we can use this by using UNION of LEFT and RIGHT join.

Syntax:

```
SELECT table1.column1, table2.column2 ....
FROM table1 FULL JOIN table2
ON
table1.common_field = table2.common_field;
```

Example:

```
SELECT *
FROM table1 FULL JOIN table2
ON
table1.value1 = table2.value2;
```

table1

Key1	Value1
1	a
2	b
3	c
4	d
5	f

table2

Key2	Value2
5	a
6	b
7	c
8	d
9	e

Result

Key1	Value1	Key2	Value2
1	a	5	A
2	b	6	B
3	c	7	C
4	d	8	D
5	e	(NULL)	(NULL)
(NULL)	(NULL)	9	E

Furthermore, there are more joins which are described below:

1. Natural Join

Natural Join joins two tables based on same column name and datatypes. Main difference between inner join and natural join is the number of columns returned. The resulting table will contain all the columns of both the table but keeps only one copy common column.

Syntax:

```
SELECT *
FROM table1 NATURAL JOIN Table2;
```

If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with the USING clause.

Example:

```
SELECT *
FROM table1   NATURAL JOIN table2;
```

table1

Key1	Value1
1	a
2	b
3	e
4	d
5	f

table2

Key2	Value1
5	a
6	b
7	c
8	d
9	e

Result

Value1	Key1	Key2
a	1	5
b	2	6
d	4	8
e	3	9

Note: Above Natural Join syntax is not valid in Microsoft SQL Server (T-SQL) while PostgreSQL, MySQL and Oracle support natural joins.

2. Cartesian Join

The CARTESIAN JOIN or CROSS JOIN returns the cartesian product of the sets of records from two or more joined tables.

Syntax:

```
SELECT column_name
FROM table1 CROSS JOIN table2;
```

Example:

Orders

id	item
1	Phone
2	laptop

Sales

id	Items
2	Laptop
3	camera

```
SELECT * FROM orders
CROSS JOIN Sales;
```

Result

id	item	id	items
1	Phone	2	Laptop
1	Phone	3	Camera
2	Laptop	2	Laptop
2	laptop	3	Camera

Total no of rows in result:

Orders_row*Sales_row=2*2=4

4.2 DDL and DML Queries in SQL

4.2.1 Data Definition Language (DDL)

DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database. Operations that we can perform on database using DDL:

- CREATE: To create the database instance / object

Example:

ALTER TABLE student
ADD date_of_birth DATE;

b. Delete existing Column

Syntax:

ALTER TABLE table_name
DROP COLUMN column_name;

Example:

ALTER TABLE student
DROP COLUMN date_of_birth;

c. Change data type of column:

Syntax:

ALTER TABLE table_name
ALTER COLUMN column_name datatype;

Example:

ALTER TABLE student
ALTER COLUMN s_id VARCHAR(20);

d. Add constraint

Example:

ALTER TABLE student
ADD CONSTRAINT PK_student PRIMARY KEY (s_id);

e. Drop constraint

Example:

ALTER TABLE student
DROP CONSTRAINT PK_student;

3. Drop Operation

It is used to delete database instance and database objects like Table, View, Index etc.

a. Drop Database

Syntax:

DROP DATABASE database_name;

Example:

DROP DATABASE Nepal;

b. Drop Table

Syntax:

DROP TABLE table_name;

Example:

DROP TABLE Student;

4. Truncate Operation

It is used to delete the data from table without deleting the table structure.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE TABLE student;

5. Rename Operation

It is used to rename database Instance and Objects.

a. Rename Database:

Syntax:

ALTER DATABASE current_database_name MODIFY NAME = new_database_name ;

Example:

ALTER DATABASE Nepal MODIFY NAME = Nepal_college ;

b. Rename Table:

Syntax:

EXEC sp_rename 'old_table_name', 'new_table_name';

Example:

In MYSQL :

RENAMETable 'emp' TO 'employee' ;

In SQL SERVER:

EXEC sp_rename 'emp', 'employee';

4.2.2 Data Manipulation Language (DML)

DML is used to accessing and manipulating data in database. DML is also known as query language. The following operations on database comes under DML:

- INSERT: To insert record(s) into the table(s)
- SELECT: To read records from table(s)
- UPDATE: Update the data in table(s)
- DELETE: Delete all the records from the table

1. Insert Operation

a. Inserting into single row:

Syntax:

```
INSERT INTO table_name(column1, column2, column3...)
VALUES (value1, value2, value3...);
```

Example:

```
INSERT INTO student(s_id, s_name, s_address)
VALUES (5, 'Binay', 'Pokhara');
```

Remember:

- There is no need to mention column name and value if it is auto increment type.

  Example: if s_id is auto increment type then

  ```
  INSERT INTO student(s_name, s_address)
  VALUES ('Binay', 'Pokhara');
  ```

- If all the column from table are to be filled with values then no need to mention columns name.

Example:

```
INSERT INTO student
VALUES ('Binay', 'Pokhara');
```

b. Inserting into multiple rows:

Syntax:

```
INSERT INTO table_name(column1, column2, column3...)
VALUES (value1, value2, value3...),
VALUES (value1, value2, value3...),
VALUES (value1, value2, value3...);
```


Example:

```
INSERT INTO student(s_id, s_name, s_address)
VALUES (2, 'Bibek', 'Dharan'),
VALUES (3, 'Ajay', 'Baglung'),
VALUES (4 , 'Meera', 'Pokhara');
```

2. Select Operation

a. Selects all rows from the table

Syntax:

```
SELECT *
FROM table_name;
```

Example:

```
SELECT *
FROM emp;
```

b. Select specific columns from a table:

Syntax:

```
SELECT column1, column2, column3
FROM table_name;
```

Example:

```
SELECT s_id, s_name, s_address
FROM emp;
```

c. Select command with where clause

To select specific rows from a table we include "where" clause in the select command. It can appear only after the "from" clause.

Syntax:

```
SELECT column_name1, ....., column_namen
FROM table name
WHERE condition;
```

Example:

```
SELECT empno, empname
FROM emp
WHERE sal >40000;
```

d. Select command with order by clause

In order to display the tuples in ascending or descending order, ORDER BY statement is used.

Syntax:

SELECT column_name1, ..., column_name

FROM table name

WHERE condition ORDER BY column_name;

We may specify ***desc*** for descending order or ***asc*** for ascending order for each attribute, ascending order is the default.

Example:

List in alphabetical order the names of all instructors.

SELECT name

FROM instructor

ORDER BY name;

List names of all instructors in descending order

SELECT name

FROM instructor

ORDER BY name DESC;

e. Select command to insert records:

Syntax:

INSERT INTO tablename (SELECT columns from existing_tablename);

Example:

INSERT INTO emp1 (SELECT * from EMP);

f. Selecting column using alias

Alias is a temporary name given to table or columns of a table. Alias makes the column name more readable and user friendly. It exists only for the duration of the query.

Syntax: (table alias)

SELECT column_name(s)

FROM table_name AS alias_name;

Syntax: (column alias)

SELECT column_name AS alias_name

FROM table_name;

Example:

SELECT e.emp_id AS Id, e.ename AS Name

FROM emp AS e;

g. Like operator in SELECT

It is used to find pattern on column values. It is mostly used with wildcards. Wildcards are used with LIKE operator to find pattern on data value.

Wildcard in SQL server

Symbol	Description	Example
%	Represents zero or more characters	ma% finds ma, man, manager, mango
_	Represents a single character	c_t finds cat, cup and cut
[]	Represents any single character within the brackets	p[ou]t finds pot and put but not pit
^	Represents any character not in the brackets	h[^oa]t finds hit but not hot and hat
-	Represents a range of characters	c[a-b]t finds cat and cbt

All the wildcards can be used in combination.

Example:

i. Employee whose position ends with 'engineer'.

SELECT *

FROM emp

WHERE position like '%engineer' ;

ii. Employee whose position starts with 'Associate'.

SELECT *

FROM emp

WHERE position like 'Associate%' ;

iii. Employee whose position contains word 'engineer'.

```
SELECT *
FROM emp
WHERE position like '%engineer%';
```

iv. Employee whose position starts with 'A' and ends with 'r'.

```
SELECT *
FROM emp
WHERE position like 'A%r';
```

v. Select the record of employee whose position has second letter 's'.

```
SELECT *
FROM emp
WHERE position LIKE '_s%';
```

vi. Select the record of employee whose position has 'e' in third last letter.

```
SELECT *
FROM emp
WHERE position LIKE '%e__';
```

vii. Select the department head whose name starts with 'r' and has at least 3 letters.

```
SELECT *
FROM dept
WHERE d_head like 'r__%';
```

3. **Update Operation**

a. **Update single column:**

Syntax:

```
UPDATE table_name SET column_name = new_values
WHERE condition;
```

Example:

```
UPDATE emp SET salary = 100000 WHERE emp_id= 5;
```


b. **Update multiple column:**

Syntax:

```
UPDATE table_name SET column1 = value1, column2 =
value2, ... WHERE condition ;
```

Example:

```
UPDATE emp SET ename = 'Bijay', address= Bhaktapur'
WHERE emp_id = 5;
```

c. **Update using CASE**

Example:

```
UPDATE table SET uid = CASE
WHEN id = 1 THEN 2952
WHEN id = 2 THEN 4925
WHEN id = 3 THEN 1592
ELSE uid
END
WHERE id in (1,2,3);
```

4. **Delete operation**

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Example:

```
DELETE FROM emp
WHERE eid = 5;
```

Note: While using delete operation it is very much important to take care of where clause. If where clause is absent in delete statement, it deletes all the records from table. In above example if where clause is removed then it deletes all the records from 'emp' table. ie. DELETE FROM emp

4.2.3 DCL (Data Control Language)

DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give "rights & permissions". The activities or actions that a user is permitted to carry out on database objects like tables, views, procedures, or functions are

determined by these privileges. The ability to grant and revoke privileges is a crucial component of database security because it enables administrators to manage access and limit specific actions to authorized users or roles.

- **GRANT Command**

  GRANT is used to grant specific privileges or rights to users or user groups

  Syntax:

  GRANT ON TO [WITH GRANT OPTION];

  Here the following privileges can be specified:

  ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE

  Example:

  Give the user rabin permission to only view and modify records in the table employees.

  GRANT SELECT, UPDATE ON employees TO rabin;

  By default, a user granted privileges is not allowed to grant those privileges to other users. To allow this, we append the term WITH GRANT OPTION clause to the appropriate grant command.

  Example:

  GRANT SELECT, UPDATE ON employees TO rabin WITH GRANT OPTION;

  Now user-rabin can grant select and update privileges to all other users. Similarly, to revoke a privilege we use the REVOKE clause is used. This is very much like GRANT.

- **REVOKE Command**

  Revoking rights or privileges is done using the REVOKE statement.

  Syntax:

  REVOKE <privilege list> ON <relation or view name> FROM <user list>

  Example:

  REVOKE INSERT, SELECT ON employees FROM rabin

4. **MIN() Function**

MIN function returns minimum value from a selected column of the table.

General syntax :

SELECT MIN(column_name) FROM table_name;

5. **SUM() Function**

SUM function returns total sum of a selected columns numeric values.

General syntax :

SELECT SUM(column_name) FROM table_name;

Statements

4.2.5 Aggregation with Grouping

The group by statement groups the rows that have the same values in the summary rows like "find the number of customers in each city". The group by statement is often used with aggregate functions to group the result set by one or more columns.

GROUP BY Syntax:

SELCET column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name;

Example:

1. **Consider the table customers as:**

Customers

Cust_ID	Cust_Name	Contact_Name	Country
1	Chennai Store	Venkatesh	India
2	Pakistan Sells	Javed	Pakistan
3	Pathao	Subrato	Bangladesh
4	Rabin Store	Rabin	Nepal
5	Delhi Express	Athul	India


```
SELECT COUNT(Cust_ID),Country
FROM Customers
GROUP BY Country;
```

Output:

	Country
1	Bangladesh
2	India
1	Nepal
1	Pakistan

The following statement lists the number of customers in each country, sorted high to low.

```
SELECT COUNT(Cust_ID),Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(Cust_ID) DESC;
```

2. **Considers the tables orders and shippers as:**

Orders

OrderID	CustomerID	EmployeeID	ShipperID
10248	90	5	3
10249	81	6	1
10250	34	4	2
10251	29	3	2

Shippers

ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

```
SELECT
Shippers.ShipperName,        COUNT(Orders.OrderID)  AS
NumberOfOrder
FROM Orders LEFT JOIN Shippers
ON
Orders.ShipperID=Shippers.ShipperID
GROUP BY ShipperName;
```

Output:

ShipperName	NumberOfOrder
FederalShipping	1
Speedy Express	1
United Package	2

4.2.6 Having Clause

The having clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

General Syntax:

```
SELECT Column_Name(s)
FROM table_name
WHERE Condition
GROUP BY column_name
HAVING Condition;
```

Example:

```
SELECT COUNT(Cust_ID),Country
FROM Customers
GROUP BY country
HAVING COUNT(Cust_ID)>1;
```

Output:

	Country
2	India

4.2.7 Views

A view is a tailored presentation of the data contained in one or more tables. In other words, a view is a logical or virtual table, which does not exist physically in the database. Views are also called as dictionary objects. It takes the output of a query and treats it as a table.

To create view we have a following syntax:

CreateView <View_Name> as <Query>;

Followings are the advantages of views:

- They provide an additional level of table security by restricting access to a predetermined set of rows and/or columns of a table.
- They hide data complexity.
- Simplifies commands for the user because they allow them to select information from multiple tables without actually knowing how to perform a join.
- They isolate application from changes in definitions of base tables.
- They provide data in a different perspective than that of a base table by renaming columns without affecting the base table.

Although there are many advantages to views, the main disadvantage to using views rather than real tables is performance degradation.

Views can be classified into two categories on the way they are created.

1. **Simple View**

   If the view is created from a single table, it is called as a simple view.

   Example:

   A view of instructors without their salary.

```
CREATE VIEW faculty AS
SELECT ID, name, dept_name
FROM instructor;
```

2. **Complex View**

If the view is created on multiple tables, it is called as a complex view.

Example:

A view 'student_course' having the attributes crn, name, phone, coursename, enrolldata.

```
CREATE VIEW student_course AS
SELECT s.crn, s.name, s.phone, e.cname, e.enrolldata
FROM student s, course c, enroll e
WHERE s.crn=c.crn AND c.courseid=e.courseid;
```

A view can be updated with the CREATE OR REPLACE VIEW statement as follows:

```
CREATE OR REPLACE VIEW view_nameAS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

A view is deleted with the DROP VIEW statement as :

```
DROP VIEW view_name;
```

Materialized Views

Certain database systems allow view relations to be stored, but they make sure that, if the actual relations used in the view definition change, the view is kept up-to-date. Such views are called materialized views.

Example:

Consider the view departments total salary. If that view is materialized, its results would be stored in the database, allowing queries that use the view to potentially run much faster by using the precomputed view result, instead of recomputing it.

However, if an instructor tuple is added to or deleted from the instructor relation, the result of the query defining the view would change, and as a result the materialized view's contents must be updated.

Similarly, if an instructor's salary is updated, the tuple in departments total salary corresponding to that instructor's department must be updated.

View Maintenance

The process of keeping the materialized view up-to-date is called materialized view maintenance (or often, just view maintenance)

4.3 Stored Procedures

A stored procedure is a collection of precompiled SQL statements and procedural logic that are stored and executed on a database server. This allows us to encapsulate a set of database operations into reusable, well-defined routines. Stored procedures are commonly used in database applications to improve performance, improve security, and promote code reuse.

Syntax for creating stored procedures in MYSQL:

```
CREATE PROCEDURE procedure_name ([parameter_list])
BEGIN
    -- Procedure body (SQL statements)
END;
```

Here's an example of a simple stored procedure without parameters:

```
DELIMITER //
CREATE PROCEDURE HelloWorld()
BEGIN
    SELECT 'Hello, World!';
END //
DELIMITER;
```

And here's an example of a stored procedure with input parameters:

```
DELIMITER //
CREATE PROCEDURE GetCustomerByCountry(IN
country_name VARCHAR(50))
```

```
BEGIN
    SELECT * FROM Customers WHERE Customers.Country =
    country_name;
END //
DELIMITER;
```

To execute a stored procedure, we can use the CALL statement:

```
CALL procedure_name([argument_list]);
```

For example, to call the GetCustomerByCountry stored procedure with the argument 'USA':

```
CALL GetCustomerByCountry('USA');
```

Here are some of the key features and benefits of using stored procedures.

1. **Code Encapsulation**

   A stored procedure encapsulates a set of SQL statements and business logic into a single entity. This makes managing and maintaining complex database operations easier.

2. **Performance Optimization**

   By precompiling SQL statements within stored procedures, the database server can execute the SQL statements more efficiently. This improves performance, especially for frequently performed operations.

3. **Code Reusability**

   Stored procedures can be reused for multiple applications and database operations. This promotes code modularity and reduces code duplication.

4. **Security**

   Stored procedures can provide an additional layer of security by allowing controlled access to the database. Users can be granted privileges to execute specific stored procedures while restricting direct access to the underlying tables.

5. **Transaction Management**

   We can use stored procedures to define transaction boundaries and see if a set of database operations succeeds

challenging and may result in inconsistent behavior and incompatibility.

iv. **Limited Language Capabilities:** Typically, stored procedures are created with SQL or another particular procedural language that the DBMS supports. When compared to general-purpose programming languages, this can restrict the flexibility and possibilities of programming. Within the limitations of a stored procedure, it may be more difficult or less effective to apply complex logic and advanced programming techniques.

v. **Security Concerns:** If not properly deployed or secured, stored procedures may provide security vulnerabilities. They have immediate access to the data and system resources since they operate within the database environment. Data breaches, unauthorized data changes, or denial-of-service attacks can result from the unlawful or malicious usage of stored procedures. When employing stored procedures, access control, parameter validation, and security best practices need to be carefully considered.

vi. **Deployment Complexity:** In contexts with several database instances or distributed systems, changing stored procedures may necessitate additional steps and collaboration. It can be difficult to provide consistent deployment and versioning across many environments; this calls for careful preparation and cooperation to prevent delays or inconsistencies.

Applications of Stored Procedures

Stored procedures have numerous applications across various domains and industries. Here are some common applications of stored procedures:

- **Business Logic Implementation:** Complex business rules and logic are frequently implemented in databases using stored procedures. They enable calculations, validations, and data manipulations to be carried out directly on the database server, assuring correctness and consistency across many applications.

- **Data Validation and Enforcement:** To enforce data validation policies and guarantee data integrity, stored procedures can be employed. When new data is received, they can check it for accuracy and, if it doesn't meet the necessary standards, reject or alter it. This is especially helpful in situations where data quality and compliance are important considerations.

- **Data Retrieval and Reporting:** The process of retrieving and presenting data for reporting purposes can be made simpler by stored procedures. To produce the required output, they can apply aggregations or calculations, link various tables, and encapsulate complex queries. This makes it simpler to produce reliable reports across many programs and user interfaces.

- **Security and Access Control:** A database's security controls are enforced in part by stored procedures. By demanding correct identification and authorization before running the stored procedure, they can restrict access to sensitive data. Stored procedures aid in enforcing security regulations and preventing unauthorized access to or manipulation of data by encapsulating data access logic.

- **Batch Processing and Automation:** When a collection of operations needs to be carried out on a significant amount of data, stored procedures are frequently utilized for batch processing activities. They can be set up to run automatically on a schedule or in response to circumstances. Tasks like data imports, data transformations, and regular maintenance procedures can all benefit from this.

- **Data Integration and ETL Processes:** Data integration and ETL (Extract, Transform, Load) operations frequently use stored procedures. Data can be extracted from many sources, transformed as needed, and loaded into target tables or data warehouses. Using stored processes, data may be moved and consolidated effectively, resulting in seamless data integration.

- **Performance Optimization:** Performance can be increased with stored procedures by streamlining query execution and lowering network traffic. The necessity for repetitive parsing and optimization is removed by stored procedures by precompiling and storing the SQL statements. Faster query execution results from this, especially for complicated or often executed processes.

4.4 Query By Example (QBE)

QBE is a graphical query language based on domain relational calculus. It's a query language used in relational databases that allows users to search for information in tables and fields by providing a simple user interface where the user will be able to input an example of the data that he or she wants to access. It is merely an abstraction between the user and the real query that the database system will receive. In the background, the user's query is transformed into a database manipulation language form such as SQL, and it is this SQL statement that will be executed in the background.

QBE Skeleton Table for the Bank Example:

branch	branch_name	branch_city	assets

customer	customer_name	customer_street	customer_city

loan	loan_number	branch_name	amount

borrower	customer_name	loan_number

account	account_number	branch_name	balance

depositor	customer_name	account_number

Example :

1. **Find all loan numbers at the Perryridge branch.**

loan	loan_number	branch_name	amount
	P._x	Perryridge	

_x is a variable(optional; can be omitted in above query)

P. means print(display)

Duplicates are removed by default

To retain duplicates we can use P.ALL

loan	loan_number	branch_name	amount
	P.ALL	Perryridge	

2. **Display full details of all loans.**

loan	loan_number	branch_name	amount
	P._x	P._y	P._z

Method 2: Shorthand notation

loan	loan_number	branch_name	amount
P.			

3. **Find the loan number of all loans with a loan amount more than $700.**

loan	loan_number	branch_name	amount
	P.	P.	>700

4. **Find names of all branches that are not located in Brooklyn.**

branch	branch_name	branch_city	assets
	P.	¬Brooklyn	

5. Find the names of all customers who have a loan from Perryridge branch.

loan	loan_number	branch_name	amount
	_x	Perryridge	

borrower	customer_name	loan_number
	P._y	_x

The Condition Box

Allows the expression of constraints on domain variables that are either inconvenient or impossible to express within the skeleton tables. In condition boxes we can use complex conditions.

Example:

Find the loan numbers of all loans made to Smith, to jones or to the both.

borrower	customer_name	loan_number
	_n	P._x

Conditions
_n=Smith or _n=Jones

SOLUTION TO EXAMS' AND OTHER IMPORTANT QUESTIONS

1. Write SQL statements for the following queries using the given Employees relation.

E_id	Fname	Lname	Department	Salary	Hire_Date
01	Ramu	Bashyal	Sales	20000	2023-08-08
02	Damu	Pandey	IT	50000	2022-01-01
03	Biru	B.K.	Sales	40000	2021-02-10
04	Hiru	Dhamala	HR	35000	2023-12-18
05	Biren	Khadka	IT	60000	2012-10-22

i. Create a database named company and Employee relation
ii. Create a view that shows the E_id,Department and Hire_Date of all employees
iii. Modify the table such that the Department of Biren is HR now
iv. Delete the record of employees whose Lname is "Pandey". [2023 Spring]

ANS:

i.
```
CREATE DATABASE company;
USE company;
CREATE TABLE Employee (
                        E_id INT PRIMARY KEY,
                        Fname VARCHAR(255),
                        Lname VARCHAR(255),
                        Department VARCHAR(255),
                        Salary INT,
                        Hire_Date DATE
);

-- Insert data into the Employee table
```

```
INSERT INTO Employee (E_id, Fname, Lname,
Department, Salary, Hire_Date) VALUES
(01, 'Ramu', 'Bashyal', 'Sales', 20000, '2023-08-08'),
(02, 'Damu', 'Pandey', 'IT', 50000, '2022-01-01'),
(03, 'Biru', 'B.K.', 'Sales', 40000, '2021-02-10'),
(04, 'Hiru', 'Dhamala', 'HR', 35000, '2023-12-18'),
(05, 'Biren', 'Khadka', 'IT', 60000, '2012-10-22');
```

ii.
```
CREATE VIEW Employee_View AS
SELECT E_id, Department, Hire_Date FROM Employee;
```

iii.
```
UPDATE Employee
SET Department = 'HR'
WHERE Fname= 'Biren' and Lname = 'Khadka';
```

iv.
```
DELETE FROM Employee WHERE Lname = 'Pandey';
```

2. Consider the relation Actress_Details and write the SQL statements for the following queries:

Players_id	Actress_name	Debut_year	Recent_release	Actress_fee
1	Renu	2010	Samay	400000
2	Sita	2022	Radha	300000
3	Geeta	2001	Mato	600000
4	Amita	1990	Man	700000
5	Karishma	1989	Prem	100000

i. Create the table Actress_details relation.
ii. Delete the data of actress whose recent release is Prem.
iii. Modify the database so that Renu's new release is "Win the Race" film
iv. Insert a new record in the above table. [2022 Fall]

ANS: i.
```
CREATE TABLE Actress_Details (
Players_id INT,
Actress_name VARCHAR(50),
Debut_year INT,
Recent_release VARCHAR(50),
Actress_fee DECIMAL(10,2)
);
```

ii.
```
DELETE FROM Actress_Details
WHERE Recent_release = 'Prem';
```

iii.
```
UPDATE Actress_Details
SET Recent_release = 'Win the Race'
WHERE Actress_name = 'Renu';
```

iv.
```
INSERT INTO Actress_Details (Players_id, Actress_name,
Debut_year, Recent_release, Actress_fee)
VALUES (6, 'Priya', 2018, 'Journey', 500000);
```

3. Considering the following schemas:
Sailors (sid, sname, rating, age)
Boats (bid, bname, color)
Reserves (sid, bid, day)
Write SQL expressions for the following queries:

i. Find the records of sailors who have reserved boat number 103 (bid=103).
ii. Update the color of the boat, where bid is 104, into green.
iii. Find the names of sailors who have reserved a red or green boat.
iv. Find the names of sailors who have reserved boat number 103 on day 5.
v. Find the names of sailors whose name is not 'Ram'
vi. Find the names of all boats. [2021 Spring]

ANS: i.
```
SELECT *
FROM Sailors
JOIN Reserves ON Sailors.sid = Reserves.sid
JOIN Boats ON Reserves.bid = Boats.bid
WHERE Boats.bid = 103;
```

ii.
```
UPDATE Boats
SET color = 'green'
WHERE bid = 104;
```

iii. SELECT DISTINCT Sailors.sname
FROM Sailors
JOIN Reserves ON Sailors.sid = Reserves.sid
JOIN Boats ON Reserves.bid = Boats.bid
WHERE Boats.color = 'red' OR Boats.color = 'green';

iv. SELECT Sailors.sname
FROM Sailors
JOIN Reserves ON Sailors.sid = Reserves.sid
JOIN Boats ON Reserves.bid = Boats.bid
WHERE Boats.bid = 103 AND Reserves.day = 5;

v. SELECT sname
FROM Sailors
WHERE sname != 'Ram';

vi. SELECT bname
FROM Boats;

4. Write SQL statements for following:

i. Create a table named Chef with chef license as primary key and following attributes: chef_license, c_fname, c_lname, c_dob, c_gender, c_experience_hours, c_photograph
ii. Enter a full detailed information of a chef.
iii. Change chefs experience hours by any value.
iv. Remove all chef records whose name contains character 'r' in second position in his first name.
v. Display the total experience hours of all chef.
vi. Create a view from above table.
vii. Display details of chef ordering on descending manner in last name and by first name when last name matches. [2021 Fall]

ANS: i. CREATE TABLE Chef (
chef_license INT PRIMARY KEY,
c_fname VARCHAR(50),
c_lname VARCHAR(50),
c_dob DATE,

c_gender CHAR(1),
c_experience_hours INT,
c_photograph BLOB
);

ii. INSERT INTO Chef (chef_license, c_fname, c_lname, c_dob, c_gender, c_experience_hours, c_photograph)
VALUES (1234, 'John', 'Doe', '1985-06-15', 'M', 2000, NULL);

iii. UPDATE Chef SET c_experience_hours = 2500 WHERE chef_license = 1234;

iv. DELETE FROM Chef WHERE c_fname LIKE '_r%';

v. SELECT SUM(c_experience_hours) as total_experience_hours FROM Chef;

vi. CREATE VIEW Chef_View AS
SELECT chef_license, c_fname, c_lname, c_dob, c_gender, c_experience_hours
FROM Chef;

vii. SELECT * FROM Chef
ORDER BY c_lname DESC, c_fname;

5. Write SQL statement for the following schemas (underline indicates primary key):
Employee (Emp_No , Name , Address)
Project (PNo ,Pname)
Workon (Emp_No , PNo)
Part (Partno ,Part_name , Qty_on_hand)
Use (Emp_No , PNo , Partno ,Number)

a. Listing all the employee details who are not working yet .
b. Listing partname and Quantity on hand those were used in DBMS project.
c. List the name of the projects that are used by employee from London.
d. Modify the database so that Jones now lives in USA.
e. Update address of an employee 'Japan' to 'USA'. [2020 Spring]

ANS: a. SELECT *
FROM Employee
WHERE Emp_No NOT IN (SELECT Emp_No FROM Workon);

b. SELECT Part.Part_name, Part.Qty_on_hand
FROM Part
JOIN Use ON Use.Partno = Part.Partno
JOIN Project ON Project.PNo = Use.PNo
WHERE Project.Pname = 'DBMS';

c. SELECT DISTINCT Project.Pname
FROM Project
JOIN Workon ON Workon.PNo = Project.PNo
JOIN Employee ON Employee.Emp_No = Workon.Emp_No
WHERE Employee.Address = 'London';

d. UPDATE Employee
SET Address = 'USA'
WHERE Name = 'Jones';

e. UPDATE Employee
SET Address = 'USA'
WHERE Address = 'Japan';

6. Write SQL statements for following:

i. Create a table named Automotor with chasis_number as primary key and following attributes:
veh_brand, veh_name, veh_model, veh_year, veh_cost, veh_color, veh_weight

ii. Enter a full detailed information of an automotor.

iii. Change any Automotor's year to 2019.

iv. Remove all Automotor records whose model contains character 'i' in last position.

v. Display the total cost of all vehicles of the table Automotor.

vi. Create a view from above table having vehicles only red color.

vii. Display details of Automotor ordering on descending manner by brand name and by ascending on model when brand matches.

viii. Change data type of color so that it only takes one character. **[2020 Fall]**

ANS: i. CREATE TABLE Automotor (
chasis_number INTEGER PRIMARY KEY,
veh_brand VARCHAR(30),
veh_name VARCHAR(30),
veh_model VARCHAR(30),
veh_year INTEGER,
veh_cost REAL,
veh_color VARCHAR(10),
veh_weight REAL
);

ii. INSERT INTO Automotor VALUES (1, 'Toyota', 'Corolla', 'SE', 2020, 25000, 'Black', 1200);

iii. UPDATE Automotor SET veh_year = 2019 WHERE chasis_number = 1;

iv. DELETE FROM Automotor WHERE veh_model LIKE '%i';

v. SELECT SUM(veh_cost) AS total_cost FROM Automotor;

vi. CREATE VIEW Red_Vehicles AS
SELECT * FROM Automotor WHERE veh_color = 'Red';

vii. SELECT * FROM Automotor ORDER BY veh_brand DESC, veh_model ASC;

viii. ALTER TABLE Automotor
ALTER COLUMN veh_color CHAR(1);

7. Differentiate between join and sub query. **[2020 Fall]**

ANS: SQL uses joins and subqueries, two distinct methods, to query and retrieve data from relational databases. These are their differences:

Join:

- When combining rows from two or more tables based on similar columns, a join is utilized.
- By including the conditions that establish the relationship between the tables in the join clause, we can obtain data from many tables.
- By matching values in the specified columns, joins can be used to fetch columns from many tables, including related data.
- The way the data from the joined tables is matched and returned depends on the type of join used. Common types include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.

Subquery:

A subquery is a query that is nested inside of another query, also referred to as a nested query or inner query.

- It enables us to incorporate the inner query's output into the outer query.
- Subqueries may be used in the SELECT, FROM, WHERE, or HAVING clauses, among other portions of a query.
- Using operations on a subset of data, a subquery, which is surrounded in parenthesis, can be used to filter or manipulate data.
- The outer query uses a subquery's output to further refine the set of results that it returns.

8. Write the SQL statements for the following queries by reference of Hotel_details relation:

hotel_id	hotel_name	estb_year	hotel_star	hotel_worth
1	Hyatt	2047	Five	15M
2	Hotel Ktm	2043	Three	5M
3	Fulbari	2058	Five	20M
4	Yak and Yeti	2052	Four	11M
5	Hotel Chitwan	2055	Three	7M

i. **Create a database named hotel & table relation.**
ii. **Create a view named Price which shows hotel name & its worth.**
iii. **Modify the data so that Hotel Chitwan is now four star level.**
iv. **Delete the records of all hotels having worth more than 9M.** [2019 Spring]

ANS: i.
```
CREATE DATABASE hotel;
USE hotel;
CREATE TABLE hotel_details (
hotel_id INT NOT NULL PRIMARY KEY,
hotel_name VARCHAR(50) NOT NULL,
estb_year INT NOT NULL,
hotel_star VARCHAR(10) NOT NULL,
hotel_worth FLOAT NOT NULL
);
INSERT INTO hotel_details (hotel_id, hotel_name,
estb_year, hotel_star, hotel_worth)
VALUES (1, 'Hyatt', 2047, 'Five', 15000000),
2, 'Hotel Ktm', 2043, 'Three', 5000000),
3, 'Fulbari', 2058, 'Five', 20000000),
4, 'Yak and Yeti', 2052, 'Four', 11000000),
5, 'Hotel Chitwan', 2055, 'Three', 7000000);
```

ii.
```
CREATE VIEW Price AS
SELECT hotel_name, hotel_worth FROM hotel_details;
```

iii.
```
UPDATE hotel_details
SET hotel_star = 'Four'
WHERE hotel_name = 'Hotel Chitwan';
```

iv.
```
DELETE FROM hotel_details
WHERE hotel_worth > 9000000;
```

9. 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 (PatientLD, 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 's'.
ii. Delete the record of Doctors whose salary is greater than average salary of doctors.
iii. Increase the salary of doctors by 18.5% who works in OPD department.
iv. Find the average salary of Doctors for each address who have average salary more than 55K. [2019 Fall]

ANS: i.
```
SELECT p.PatientID
FROM Patients p
JOIN Hospitals h ON p.PatientID = h.PatientID
WHERE h.Location = 'Pokhara' AND p.PatientName LIKE
'%s';
```

ii.
```
DELETE FROM Doctors
WHERE Salary > ( SELECT AVG(Salary)
FROM Doctors
);
```

iii.
```
UPDATE Doctors
SET Salary = Salary * 1.185
WHERE Department = 'OPD';
```

iv.
```
SELECT Address, AVG(Salary) AS AvgSalary
FROM Doctors
GROUP BY Address
HAVING AVG(Salary) > 55000;
```


10. Consider the relational database model:
Users (uid, cname, city)
items (itemid, itemname, city, quantity, price)
Manager (mid, aflame, city)
Query (queryno, uid, mid, itemid, query_details, hitratio)
Write SQL statements for following:

i. Create a table named Vehicle with veh_number as primary key and following attributes:
veh_type, veh_brand, veh_year, veh_mileage, veh_owner, veh_photo, veh_price
ii. Enter a full detailed information of a vehicle.
iii. Increment vehicle's price by 10,000.
iv. Remove all vehicle's records whose brand contains character '0' in second position.
v. Display the total price of all vehicles.
vi. Create a view from above table.
vii. Display details of vehicles ordering on descending manner in brand and by mileage when brand matches.
viii. Change data type of year to datetime. [2018 Spring]

ANS: i.
```
CREATE TABLE Vehicle (
veh_number INT PRIMARY KEY,
veh_type VARCHAR(50),
veh_brand VARCHAR(50),
veh_year INT,
veh_mileage INT,
veh_owner VARCHAR(50),
veh_photo BLOB,
veh_price DECIMAL(10,2)
);
```

ii.
```
INSERT INTO Vehicle (veh_number, veh_type, veh_brand
veh_year, veh_mileage, veh_owner, veh_photo, veh_price)
VALUES (1, 'Car', 'Toyota', 2020, 5000, 'John Do
'toyota.jpg', 20000.00);
```

iii. UPDATE Vehicle
SET veh_price = veh_price + 10000;

iv. DELETE FROM Vehicle
WHERE SUBSTRING(veh_brand, 2, 1) = '0';

v. SELECT SUM(veh_price) AS TotalPrice
FROM Vehicle;

vi. CREATE VIEW VehicleView AS
SELECT *
FROM Vehicle;

vii. SELECT * FROM Vehicle
ORDER BY veh_brand DESC,veh_mileage DESC;

viii. ALTER TABLE Vehicle
ALTER COLUMN veh_year DATETIME;

11. Consider the following three relations.
Doctor(Name, age , address)
Works(Name, Depart_no, salary)
Department(Depart no, depname, floor, room)
Write down the SQL statement for the following.

i. **Display the name of doctor who do not work in any department.**
ii. **Modify the database so that Dr. Hari lives in Pokhara.**
iii. **Delete all record of Doctor working in OPD department.**
iv. **Display the name of Doctors who work in at least two departments.** [2018 Fall]

ANS: i.
```
SELECT Name
FROM Doctor
WHERE Name NOT IN (
SELECT Name
FROM Works
);
```


ii.
```
UPDATE Doctor
SET address = 'Pokhara'
WHERE Name = 'Dr. Hari';
```

iii.
```
DELETE FROM Works
WHERE Depart_no IN (
SELECT Depart_no
FROM Department
WHERE depname = 'OPD'
);
```

iv.
```
SELECT Name
FROM Works
GROUP BY Name
HAVING COUNT(DISTINCT Depart_no) >= 2;
```

12. Differentiate between SQL and MYSQL. Why access to database from general purpose programming language is required? Explain. [2018 Fall]

ANS: In the field of databases, SQL (Structured Query Language) and MySQL are similar but different ideas. These are their differences:

SQL:

- For handling relational databases, SQL is a standardized language.
- For defining, modifying, and querying data in a relational database management system (RDBMS), it offers a set of commands and syntax.
- SQL is supported by a wide range of well-known databases, including MySQL, Oracle, Microsoft SQL Server, PostgreSQL, and others. SQL is not specific to any one particular database system.
- It is a declarative language, so we may specify the data we need and the database system will take care of the specifics of retrieving or changing the data.

MySQL:

- A relational database management system (RDBMS) that employs the SQL language is specifically implemented in MySQL.
- It is a popular open-source database management system that is used for a variety of applications, from modest personal projects to extensive business systems.
- MySQL supports the SQL language and provides additional features and optimizations specific to the MySQL database system.
- It offers strong data storage, retrieval, and management features, including support with transactions, indexing, user administration, and more..

Let's talk about the need for database access from a general-purpose programming language now:

Flexibility and Control:

General-purpose programming languages like Python, Java, or C# give program logic control and a wide range of functionalities.

Developers can do calculations, build complicated business logic, modify data, and manage data transformations that go beyond what SQL by itself is capable of by directly accessing a database from a computer language.

Application Development:

General-purpose programming languages allow developers to build entire applications by combining the power of programming constructs with the data storage capabilities provided by databases.

Integration and Interoperability:

Many applications require complex business logic, user interfaces, connection with external systems, and more in addition to data storage and retrieval.

Connecting databases to general-purpose programming languages enables seamless integration with other systems and technologies.

By accessing the database from a programming language, developers can interact with APIs, web services, external libraries, and frameworks, enabling data-driven applications and system integrations.

Data Processing and Analysis:

Rich libraries and tools are available for data processing, analysis, and visualization in general-purpose programming languages.

Developers can use these tools to perform complex computations, statistical analysis, machine learning, and generate reports based on the data stored in the database by accessing the database from a computer language.

13. Write the SQL statements for the following queries by reference of Liquors_Info relation:

Serial No	Liquors	Start year	Bottles	Ready year
1	Gorkha	1997	10	1998
2	Divine Wine	1998	5	2000
3	Old Durbar	1997	12	2001
4	Khukuri Rum	1991	10	1992
5	Xing	1994	5	1995

i. **Create the Liquors_Info relation.**
ii. **Insert the records in Liquors_Info as above.**
iii. **List all the records which were ready by 2000.**
iv. **Remove all records from data base that required more than 2 years to get ready.** **[2017 Fall]**

ANS: i.
```
CREATE TABLE Liquors_Info (
Serial_No INT PRIMARY KEY,
```

```
Liquors VARCHAR(50),
Start_Year INT,
Bottles INT,
Ready_Year INT);
```

ii. INSERT INTO Liquors_Info (Serial_No, Liquors, Start_Year, Bottles, Ready_Year) VALUES

```
(1, 'Gorkha', 1997, 10, 1998),
(2, 'Divine Wine', 1998, 5, 2000),
(3, 'Old Durbar', 1997, 12, 2001),
(4, 'Khukuri Rum', 1991, 10, 1992),
(5, 'Xing', 1994, 5, 1995);
```

iii. SELECT *

```
FROM Liquors_Info
WHERE Ready_Year <= 2000;
```

iv. DELETE FROM Liquors_Info

```
WHERE (Ready_Year - Start_Year) > 2;
```

14. What is the difference between where and having clause? [2017 Fall]

ANS: In SQL queries, the WHERE and HAVING clauses are both employed to filter and restrict data according to predefined criteria. They are utilized in various parts of a query and have different functions. The where clause and the having clause differ in the following ways:

WHERE Clause:

Rows from a table are filtered using the WHERE clause according to a criteria or group of conditions. SELECT, UPDATE, and DELETE commands are where it is most frequently utilized. Before data are grouped and aggregated, the WHERE clause is used. It applies the supplied condition(s) to each individual row to filter it.

Syntax:

```
SELECT column1, column2, ...
FROM table
WHERE condition;
```

Example:

```
SELECT * FROM employees
WHERE department = 'Sales';
```

This query retrieves all the rows from the "employees" table where the department is 'Sales'.

HAVING Clause:

After data has been grouped and aggregated, rows are filtered based on conditions using the HAVING clause. It frequently goes along with the GROUP BY clause. When applied to grouped data, the HAVING clause filters the groups according to the supplied condition(s). Conditions can be applied to the output of aggregate functions like COUNT, SUM, AVG, etc.

Syntax:

```
SELECT column1, column2, ...
FROM table
GROUP BY column1, column2, ...
HAVING condition;
```

Example:

```
SELECT department, COUNT(*) as total_employees
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

This query retrieves the departments from the "employees" table along with the count of employees in each department. However, it only includes departments that have more than 5 employees in the result.

15. Consider a simple relational database of Hospital Management System.
(Underlined attributes represent Primary key attributes)
Doctors (DoctorID, DoctorName, Department, Address, Salary)

Patients (PatientID, PatentName, 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'.
ii. Delete the record of Doctors whose salary is greater than average salary of doctors.
iii. Increase the salary of doctors by 18.5% who works in OPD department.
iv. Find the average salary of Doctors for each address who have average salary more than 55K. [2016 Spring]

ANS: i.
```
SELECT PatientID FROM Hospitals
WHERE Location = 'Pokhara' AND PatientID IN
(SELECT PatientID
FROM Patients
WHERE PatentName LIKE '%a');
```

ii.
```
DELETE FROM Doctors
WHERE Salary > (SELECT AVG(Salary) FROM Doctors);
```

iii.
```
UPDATE Doctors
SET Salary = Salary + (Salary * 0.185)
WHERE Department = 'OPD';
```

iv.
```
SELECT Address, AVG(Salary)
FROM Doctors
GROUP BY Address
HAVING AVG(Salary) > 55000;
```

16. How does a view differ with relation? [2016 Spring]

ANS: A view and a relation (table) in a database differ in the following ways:

i. **Data Storage:**
Relation: Stores data directly in a physical structure (table) within the database.

View: Provides a virtual representation of data drawn from underlying tables but does not actually store the data itself.

ii. **Data Modification:**
Relation: Allows direct modifications (insert, update, delete) to the stored data.
View: Generally used for data retrieval and has limited or restricted modifications.

iii. **Data Representation:**
Relation: Represents a complete set of data with all attributes and records.
View: Represents a subset or customized representation of data from one or more relations.

iv. **Schema:**
Relation: Has its own defined schema specifying attributes, data types, and constraints.
View: Inherits the schema from the underlying relation(s) it is based on.

v. **Data Abstraction:**
Relation: Provides a concrete representation of data without abstraction.
View: Offers a level of abstraction by presenting a tailored, filtered, or aggregated view of the data.

vi. **Query Simplification:**
Relation: Requires complex SQL statements for joins, aggregations, and selections.
View: Can simplify queries by encapsulating complex operations into a single view definition.

17. Consider a relational Schema:
Teacher(TeacherID,TeacherName, Office)
Write SQL statement for the following task:
i. **To create a table from a table.**
ii. **To eliminate duplicate rows.**
iii. **To add a new column 'Gender' in the table.**

iv. To sort data in a table.
v. To delete rows.
vi. Count number of rows based on Office. [2016 Fall]

ANS: i.
```
CREATE TABLE Teacher_Details AS
SELECT TeacherID, TeacherName, Office FROM Teacher;
```
ii.
```
SELECT DISTINCT * FROM Teacher;
```
iii.
```
ALTER TABLE Teacher ADD Gender VARCHAR(10);
```
iv.
```
SELECT * FROM Teacher ORDER BY TeacherName ASC;
```
v.
```
DELETE FROM Teacher;
```
vi.
```
SELECT Office, COUNT(*) AS NumOfTeachers FROM
Teacher GROUP BY Office;
```

18. Suppose we are given the following table definitions with certain records in each table.
EMPLOYEE (EID, NAME, POST, AGE)
POST (POST_TITLE, SALARY)
PROJECT (PID, PNAME, DURATION, BUDGET)
WORK_IN (PID, EID, JOIN_DATE)
Write the SQL statement for
i. List the name of employees whose age is greater than the average age of all employees.
ii. Display all employee numbers of those employee who are not working in any project
iii. List name of employee and their salary who are working in the project "DBMS".
iv. Update the database so that "Rishab" now lives in "Butwal". [2015 Spring]

ANS: i.
```
SELECT NAME
FROM EMPLOYEE
WHERE AGE > (SELECT AVG(AGE)
FROM EMPLOYEE);
```
ii.
```
SELECT EID
FROM EMPLOYEE
WHERE EID NOT IN ( SELECT EID
FROM WORK_IN);
```


iii.
```
SELECT E.NAME, P.SALARY
FROM EMPLOYEE E
JOIN POST P
ON E.POST = P.POST_TITLE
JOIN WORK_IN W ON E.EID = W.EID
JOIN PROJECT PJ ON W.PID = PJ.PID
WHERE PJ.PNAME = 'DBMS';
```
iv.
```
ALTER TABLE EMPLOYEE
ADD ADDRESS VARCHAR(100);
UPDATE EMPLOYEE
SET ADDRESS = 'Butwal'
WHERE NAME = 'Rishab';
```

19. Write SQL statements for the following queries in reference to relation Emp_time provided.

Eid*	Name	Start_time	End time
E101	Hari	10:15	18:00
E102	Malati	8:00	15:30
E103	Kalyan	9:30	17:00

i. Create the table Eid* as primary key and insert the values provided.
ii. Display the name of the employee whose name start from letter 'M' and who work for more than seven hours.
iii. Delete the entire contents of the table so that new data can be inserted. [2015 Fall]

ANS: i.
```
CREATE TABLE Emp_time (
Eid VARCHAR(10) PRIMARY KEY,
Name VARCHAR(50),
Start_time TIME,
End_time TIME);
```

```
INSERT INTO Emp_time (Eid, Name, Start_time, End_time)
VALUES ('E101', 'Hari', '10:15', '18:00'),
('E102', 'Malati', '8:00', '15:30'),
('E103', 'Kalyan', '9:30', '17:00');
```

ii.
```
SELECT Name
FROM Emp_time
WHERE Name LIKE 'M%' AND TIMEDIFF(End_time,
Start_time) > '7:00';
```

iii.
```
DELETE FROM Emp_time;
```

20. Consider the employee database of figure given below, where primary keys are underlined. Give an expression in SQL for each of following queries.
Employee (employee-name, street, city)
Works (employee-name, company—name, salary)
company (company—name, city)
manages (employee— name, manager-name)

i. **Modify the database so that Ram now lives in Kathmandu.**
ii. **Give all employees of First Bank Corporation a 10 percent raise.**
iii. **Give all managers of First Bank Corporation a 10 percent raise.**
iv. **Delete all tuples in the works relation for employees of small Bank Corporation.**
v. **Find all employees who earn more than the average salary of all employees of their company.** **[2014 Spring]**

ANS: i.
```
UPDATE Employee
SET city = 'Kathmandu'
WHERE employee_name = 'Ram';
```

ii.
```
UPDATE Works
SET salary = salary * 1.1
WHERE company_name = 'First Bank Corporation';
```


iii.
```
UPDATE Works
SET salary = salary * 1.1
WHERE company_name = 'First Bank Corporation' AND
employee_name IN (SELECT manager_name FROM
Manages);
```

iv.
```
DELETE FROM Works
WHERE company_name = 'Small Bank Corporation';
```

v.
```
SELECT e.employee_name
FROM Employee e
INNER JOIN Works w ON e.employee_name =
w.employee_name
INNER JOIN (
SELECT company_name, AVG(salary) AS avg_salary
FROM Works
GROUP BY company_name
) AS company_avg ON w.company_name =
company_avg.company_name
WHERE w.salary > company_avg.avg_salary;
```

21. Consider following relations:
employee (emp_name, street, city)
works (emp_name, company, salary)
company (comp_name, city)
manages (emp_name, manager_name)
Write SQL statements for:

i. **Find employee names that lives in the city same as the company city.**
ii. **List all employee details who earn more than 25000.**
iii. **Update address of an employee 'Sriyash' to 'Pokhara'.**
iv. **Create a view for which employee earns Rs. 20,000 or more.**
v. **Delete all employees from the table employee.**

[2014 Fall]

ANS: i.
```
SELECT e.emp_name
FROM employee e, company c
WHERE e.city = c.city;
```

ii.
```
SELECT e.*
FROM employee e, works w
WHERE e.emp_name = w.emp_name AND w.salary >
25000;
```

iii.
```
UPDATE employee
SET city = 'Pokhara'
WHERE emp_name = 'Sriyash';
```

iv.
```
CREATE VIEW high_earners AS
SELECT e.*
FROM employee e, works w
WHERE e.emp_name = w.emp_name AND w.salary >
20000;
```

v.
```
DELETE FROM employee
```

◆◆◆