UNIT – II (RELATIONAL DATA MODEL AND LANGUAGE)

A relational database stores data in the form of relations (tables). In this model, the data is organized into a collection of two-dimensional inter-related tables

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables − in relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represent the attributes.

Tuple − A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema − A relation schema describes the relation name (table name), attributes, and their names.

Relation key − each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain − every attribute has some pre-defined value scope, known as attribute domain.

Degree: The total number of attributes which in the relation is called the degree of the relation

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity Constraints. There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called key for that relation. If there are more than one such minimal subset, these are called *candidate keys*.

Key constraints force that −

- In a relation with a key attribute, no two tuples can have identical values for key attributes.
- A key attribute cannot have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a

specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.
Every domain must contain atomic values (smallest indivisible units) it means composite and multi-valued attributes are not allowed.

We perform data type check here, which means when we assign a data type to a column we limit the values that it can contain. E.g. If we assign the data type of attribute age as int, we can't give it values other then int data type.Domain constraints are user defined data type and we can define them like this:Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.
Notation − $\sigma_p(r)$
Where σ stands for selection predicate and r stands for relation. *p* is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.
For example −
$\sigma_{subject = "database"}(Books)$
Output − Selects tuples from books where subject is 'database'.
$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$
Output − Selects tuples from books where subject is 'database' and 'price' is 450.
$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$
Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An}$ (r)
Where A_1, A_2 , A_n are attribute names of relation r.
Duplicate rows are automatically eliminated, as relation is a set.
For example −
$\prod_{subject, author}$ (Books)
Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as −
r ∪ s = { t | t ∈ r or t ∈ s}
Notation − r U s
Where r and s are either database relations or relation result set (temporary relation).
For a union operation to be valid, the following conditions must hold −

- r, and s must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)
Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
Notation − r − s
Finds all the tuples that are present in r but not in s.
$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)
Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.
Notation − r X s
Where r and s are relations and their output will be defined as −
r X s = { q t | q ∈ r and t ∈ s}
$\sigma_{author = 'tutorialspoint'}$(Books X Articles)
Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ.
Notation − ρ_x (E)
Where the result of expression E is saved with name of x.
Additional operations are −

- Set intersection
- Assignment
- Natural join

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it. The relational calculus is similar to the relational algebra, which is also part of the relational model. Relational calculus is a non-procedural query language. The relational calculus tells what to do but never explains how to do. Most commercial relational languages are based on aspects of relational calculus including SQL-QBE and QUEL.

- Tuple relational calculus which was originally proposed by Codd in the year 1972 and
- Domain relational calculus which was proposed by Lacroix and Pirotte in the year 1977

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples
Notation − {T | Condition}
Returns all tuples T that satisfies a condition.
For example −
{ T.name | Author(T) AND T.article = 'database' }
Output − Returns tuples with 'name' from Author who has written article on 'database'.
TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).

For example −
{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}
Output − The above query will yield the same result as the previous one.
Domain Relational Calculus (DRC)
In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation −

{ a_1, a_2, a_3, ..., a_n | P (a_1, a_2, a_3, ... ,a_n)}
Where a1, a2 are attributes and P stands for formulae built by inner attributes.
For example −
{< article, page, subject > | ∈ TutorialsPoint A subject = 'database'}
Output − Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.
Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.
The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

Projection (π)

Projection is used to project required column data from a relation.
Example :

R
(A B C)
----
1 2 4
2 2 3
3 2 3
4 3 4

π (BC)

B C
-----
2 4
2 3
3 4

Note: By Default projection removes duplicate data.

Selection (σ)

Selection is used to select required tuples of the relations.

for the above relation

σ (c>3)R

will select the tuples which have c more than 3.

Note: selection operator only selects the required tuples but does not display them. For displaying, data projection operator is used.

For the above selected tuples, to display we need to use projection also.

π (σ (c>3)R) will show following tuples.

```
A B  C
-------
1  2  4
4  3  4
```

Union (U)

Union operation in relational algebra is same as union operation in set theory, only constraint is for union of two relation both relation must have same set of Attributes.

Set Difference (-)

Set Difference in relational algebra is same set difference operation as in set theory with the constraint that both relation should have same set of attributes.

Rename (ρ)

Rename is a unary operation used for renaming attributes of a relation.

ρ (a/b)R will rename the attribute 'b' of relation by 'a'.

Cross Product (X)

Cross product between two relations let say A and B, so cross product between A X B will results all the attributes of A followed by each attribute of B. Each record of A will pairs with every record of B. below is the example

```
  A                          B
  (Name   Age  Sex )            (Id  Course)
  ----------------             -------------
  Ram    14   M                   1     DS
  Sona   15   F                   2     DBMS
  kim    20   M
```

```
   A X B
 Name  Age   Sex   Id   Course
---------------------------------
 Ram    14    M     1    DS
 Ram    14    M     2    DBMS
 Sona   15    F     1    DS
 Sona   15    F     2    DBMS
 Kim    20    M     1    DS
```

Kim 20 M 2 DBMS

Note: if A has ‘n’ tuples and B has ‘m’ tuples then A X B will have ‘n*m’ tuples.

Natural Join (⋈)

Natural join is a binary operator. Natural join between two or more relations will result set of all combination of tuples where they have equal common attribute.

Let us see below example

Emp (Name Id Dept_name)

A	120	IT
B	125	HR
C	110	Sale
D	111	IT

Dep (Dept_name Manager)

Sale	Y
Prod	Z
IT	A

Emp ⋈ Dep

Name	Id	Dept_name	Manager
A	120	IT	A
C	110	Sale	Y
D	111	IT	A

Conditional Join

Conditional join works similar to natural join. In natural join, by default condition is equal between common attribute while in conditional join we can specify the any condition such as greater than, less than, not equal

Let us see below example

R (ID Sex Marks)

1	F	45
2	F	55
3	F	60

S (ID Sex Marks)

10	M	20
11	M	22
12	M	59

Join between R And S with condition R.marks >= S.marks

R.ID	R.Sex	R.Marks	S.ID	S.Sex	S.Marks
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

Characteristics of SQL

1. Easy to Learn: SQL is user-friendly, English like language that makes it easy to learn. Learning SQL doesn’t require prior knowledge.

2. Portable language: SQL is a portable language, which means the software that supports SQL can be moved to another machine without affecting the capability of SQL interacting with the database on new machine.
3. Supports wide variety of commands: SQL supports various useful commands such as:

- DDL (Data Definition Language) commands like CREATE, DROP, ALTER.
- DML (Data Manipulation Language) commands like INSERT, DELETE, UPDATE.
- DCL (Data Control Language) commands like GRANT, REVOKE.
- TCL (Transaction Control Language) commands like COMMIT, ROLLBACK.
- DQL (Data Query Language) commands like SELECT.
  Reusability: SQL promotes reusability by supporting stored procedures. These stored procedures are stored SQL statements that can be used to perform a specific task any number of times. This makes it easier to write SQL statements for a re-occurring task and reusing the saved stored procedure to erform the same task without rewriting the same SQL statements again.Supports JOIN: SQL supports join hich is used to combine the data of two or more tables. This can be useful when we need to perform the operation on multiple tables.Supports UNION: UNION command can be used to join two or more DQL statement (SELECT statements).Integration: SQL allows integration to non-SQL database applications as well.Performance: Better performance even if the database size if huge.SQL is scalable and flexible. SQL is secure.

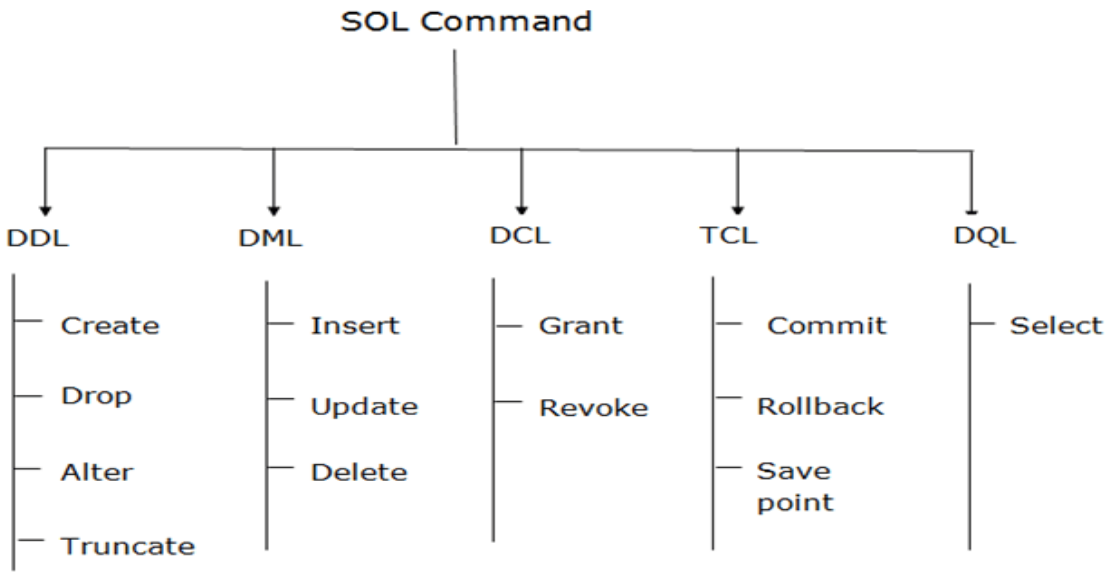
Advantages of SQL:

SQL has many advantages which makes it popular and highly demanded. It is a reliable and efficient language used for communicating with the database. Some advantages of SQL are as follows: Commonality. One of the main benefits of using SQL is the commonality of the language. ...

1. Simplicity. Another benefit of using SQL is the simplicity of the language. ...
2. Integration. ...
3. Speed. ...
4. Alter data within a table. ...
5. Create a table. ...
6. Retrieve data. ...
7. Change data structure.

Applications of SQL:

- SQL is used by developers and DBAs (Database Administrators) in writing Data Integration Scripts.
- It is used to deal with analytical queries to analyze the data and get instincts from it.
- Retrieving Information
- Modification/Manipulation of data and database table such as Insertion, Deletion and Updating.



DDL (Data Definition Language)

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

List of DDL commands:

- CREATE: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- DROP: This command is used to delete objects from the database.
- ALTER: This is used to alter the structure of the database.
- TRUNCATE: This is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT: This is used to add comments to the data dictionary.
- RENAME: This is used to rename an object existing in the database.

DQL (Data Query Language):

DQL statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it. We can define DQL as follows it is a component of SQL statement that allows getting data from the database and imposing order upon it. It includes the SELECT statement. This command allows getting the data out of the database to perform operations with it. When a SELECT is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.

List of DQL:

SELECT: It is used to retrieve data from the database.

DML (Data Manipulation Language):

The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

List of DML commands:

- INSERT : It is used to insert data into a table.
- UPDATE: It is used to update existing data within a table.
- DELETE : It is used to delete records from a database table.
- LOCK: Table control concurrency.
- CALL: Call a PL/SQL or JAVA subprogram.
- EXPLAIN PLAN: It describes the access path to data.

DCL (Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- GRANT: This command gives users access privileges to the database.
- REVOKE: This command withdraws the user's access privileges given by using the GRANT command.

Though many resources claim there to be another category of SQL clauses TCL – Transaction Control Language. So we will see in detail about TCL as well. TCL commands deal with the transaction within the database.

TCL (Transaction Control Language)

Transaction Control Language (TCL) instructions are used in the database to manage transactions. This command is used to handle the DML statements' modifications. TCL allows you to combine your statements into logical transactions.

List of TCL commands:

COMMIT: Commits a Transaction.

- ROLLBACK: Rollbacks a transaction in case of any error occurs.
- SAVEPOINT:Sets a savepoint within a transaction.
- SET TRANSACTION: Specify characteristics for the transaction.

Example of DDL commands:

CREATE

CREATE statements is used to define the database structure schema:

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[, ...]);
For example:

Create database university;
Create table students;
Create view for_students;

DROP

Drops commands remove tables and databases from RDBMS.

Syntax

DROP TABLE ;

For example:

Drop object_type object_name;
Drop database university;
Drop table student;

ALTER

Alters command allows you to alter the structure of the database.
To add a new column in the table
ALTER TABLE table_name ADD column_name COLUMN-definition;
To modify an existing column in the table:
ALTER TABLE MODIFY(COLUMN DEFINITION...);

For example:

Alter table guru99 add subject varchar;

TRUNCATE:

This command used to delete all the rows from the table and free the space containing the table.
TRUNCATE TABLE table_name;

Example:

TRUNCATE table students;

INSERT:

This is a statement is a SQL query. This command is used to insert data into the row of a table.
INSERT INTO TABLE_NAME (col1, col2, col3, ... col N)
VALUES (value1, value2, value3, ...valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, ...valueN);

For example:

INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');

UPDATE:

This command is used to update or modify the value of a column in the table.
UPDATE table_name SET [column_name1= value1,.. column_nameN = valueN] [WHERE CONDITION]

For example:

UPDATE students
SET FirstName = 'Jhon', LastName= 'Wick'
WHERE StudID = 3;

DELETE:

This command is used to remove one or more rows from a table.

DELETE FROM table_name [WHERE condition];

For example:
DELETE FROM students
WHERE FirstName = 'Jhon';

Examples of DCL commands:

Commands that come under DCL:

- Grant
- Revoke

Grant:
This command is use to give user access privileges to a database.
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
For example:
GRANT SELECT ON Users TO'Tom'@'localhost;

Revoke:
It is useful to back permissions from the user.
REVOKE privilege_nameON object_nameFROM {user_name |PUBLIC |role_name}

For example:
REVOKE SELECT, UPDATE ON student FROM BCA, MCA;

Example of TCL:

Commit
This command is used to save all the transactions to the database.
Commit;

For example:
DELETE FROM Students
WHERE RollNo =25;
COMMIT;

Rollback
Rollback command allows you to undo transactions that have not already been saved to the database.
ROLLBACK;

Example:
DELETE FROM Students
WHERE RollNo =25;

SAVEPOINT
This command helps you to sets a savepoint within a transaction.

SAVEPOINT SAVEPOINT_NAME;
Example:

SAVEPOINT RollNo;

Example of Data Query Language (DQL) command:

SELECT:
This command helps you to select the attribute based on the condition described by the WHERE clause.

SELECT expressions
FROM TABLES
WHERE conditions;

For example:
SELECT FirstName
FROM Student
WHERE RollNo > 15;

SQL Operators:

Generally, there are three types of operators that are used in SQL.

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators

Now, let's look at each one of them in detail.

1. Arithmetic SQL Operators

Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, division, and multiplication. These operators usually accept numeric operands. Different operators that come under this category are given below-

Operator	Operation	Description
+	Addition	Adds operands on either side of the operator
-	Subtraction	Subtracts the right-hand operand from the left-hand operand
*	Multiplication	Multiplies the values on each side

/	Division	Divides left-hand operand by right-hand operand
%	Modulus	Divides left-hand operand by right-hand operand and returns the remainder

2. Comparison SQL Operators

Comparison operators in SQL are used to check the equality of two expressions. It checks whether one expression is identical to another. Comparison operators are generally used in the WHERE clause of a SQL query. The result of a comparison operation may be TRUE, FALSE or UNKNOWN. When one or both the expression is NULL, then the operator returns UNKNOWN. These operators could be used on all types of expressions except expressions that contain a text, next or an image. The table below shows different types of comparison operators in SQL:

Operator	Operation	Description
=	Equal to	Checks if both operands have equal value, if yes, then returns TRUE
>	Greater than	Checks if the value of the left-hand operand is greater than the right-hand operand or not
<	Less than	Returns TRUE if the value of the left-hand operand is less than the value of the right-hand operand
>=	Greater than or equal to	It checks if the value of the left-hand operand is greater than or equal to the value of the right-hand operand, if yes, then returns TRUE
<=	Less than or equal to	Examines if the value of the left-hand operator is less than or equal to the right-hand operand
<> or !=	Not equal to	Checks if values on either side of the operator are equal or not. Returns TRUE if values are not equal
!>	Not greater than	Used to check if the left-hand operator's value is not greater than or equal to the right-hand operator's value
!<	Not less than	Used to check if the left-hand operator's value is not less than or equal to the right-hand operator's value

3. Logical SQL Operators

Logical operators are those operators that take two expressions as operands and return TRUE or False as output. While working with complex SQL statements and queries, comparison operators come in handy and these operators work in the same way as logic gates do. Different logical operations available in SQL are given in the below table.

Operator	Description
ALL	Compares a value to all other values in a set
AND	Returns the records if all the conditions separated by AND are TRUE
ANY	Compares a specific value to any other values in a set
SOME	Compares a value to each value in a set. It is similar to ANY operator
LIKE	It returns the rows for which the operand matches a specific pattern
IN	Used to compare a value to a specified value in a list
BETWEEN	Returns the rows for which the value lies between the mentioned range
NOT	Used to reverse the output of any logical operator
EXISTS	Used to search a row in a specified table in the database
OR	Returns the records for which any of the conditions separated by OR is true
NULL	Returns the rows where the operand is NULL

VIEW

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view. A **view** is simply any SELECT query that has been given a name and saved in the database. For this reason, a view is sometimes called a **named query** or a **stored query**.

Views, which are a type of virtual tables, allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+

SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;

SQL > SELECT * FROM CUSTOMERS_VIEW;
```

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```


INDEXES

An index helps to speed up **SELECT** queries and **WHERE** clauses, but it slows down data input, with the **UPDATE** and the **INSERT** statements. Indexes can be created or dropped with no effect on the data an **index**, as you would expect, is a data structure that the database uses to find records within a table more quickly. Indexes are built on one or more columns of a table; each index maintains a list of values within that field that are sorted in ascending or descending order. Rather than sorting records on the field or fields during query execution, the system can simply access the rows in order of the index.

Syntax: As you would expect by now, the SQL to create an index is:

```
CREATE INDEX <indexname> ON <tablename> (<column>, <column>...);
```

To enforce unique values, add the UNIQUE keyword:

```
CREATE UNIQUE INDEX <indexname> ON <tablename> (<column>, <column>...);
```

To specify sort order, add the keyword ASC or DESC after each column name, just as you would do in an

ORDER BY clause.
To remove an index, simply enter:

```
DROP INDEX <indexname>;
```

SUBQUERY

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

```
SELECT column_name [, column_name ]
FROM  table1 [, table2 ]
WHERE column_name OPERATOR
  (SELECT column_name [, column_name ]
  FROM table1 [, table2 ]
  [WHERE])
```

AGGREGATE FUNCTION

An aggregate function in SQL returns one value after calculating multiple values of a column. We often use aggregate functions with the GROUP BY and HAVING clauses of the SELECT statement.Various types of SQL aggregate functions are:

- Count()
- Sum()
- Avg()
- Min()
- Max()

INSERT

This is a statement is a SQL query. This command is used to insert data into the row of a table.

```
INSERT INTO TABLE_NAME  (col1, col2, col3, ... col N)
VALUES (value1, value2, value3, .....valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .....valueN);
```

For example:

```
INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');
```

UPDATE

This command is used to update or modify the value of a column in the table.

```
UPDATE table_name SET [column_name1= value1,.. column_nameN = valueN] [WHERE CONDITION]
```

For example:

```
UPDATE students
SET FirstName = 'Jhon', LastName= 'Wick'
WHERE StudID = 3;
```

DELETE

This command is used to remove one or more rows from a table.

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM students
WHERE FirstName = 'Jhon';
```

JOIN

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 Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

A. INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:

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

B. LEFT JOIN

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.


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

C. RIGHT JOIN

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as **RIGHT OUTER JOIN.**

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

D. FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain *NULL* values.

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

CARTESIAN JOIN

The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.

- In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT . i.e., the number of rows in the result-set is the product of the number of rows of the two tables.
- In the presence of WHERE condition this JOIN will function like a INNER JOIN.
- Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True

Syntax:

SELECT table1.column1 , table1.column2, table2.column1...
FROM table1
CROSS JOIN table2;

table1: First table.
table2: Second table

EXAMPLE

SELECT Student.NAME, Student.AGE, StudentCourse.COURSE_ID
FROM Student
CROSS JOIN StudentCourse;

SELF JOIN

As the name signifies, in SELF JOIN a table is joined to itself. That is, each row of the table is joined with itself and all other rows depending on some conditions. In other words we can say that it is a join between two copies of the same table.**Syntax:**

SELECT a.coulmn1 , b.column2
FROM table_name a, table_name b
WHERE some_condition;

EXAMPLE

SELECT a.ROLL_NO , b.NAME
FROM Student a, Student b
WHERE a.ROLL_NO < b.ROLL_NO;

UNION

The Union Clause is used to combine two separate select statements and produce the result set as a union of both the select statements.

NOTE:

1. The fields to be used in both the select statements must be in same order, same number and same data type.
2. The Union clause produces distinct values in the result set, to fetch the duplicate values too UNION ALL must be used instead of just UNION.

Basic Syntax:

SELECT column_name(s) FROM table1 UNION SELECT column_name(s) FROM table2;

UNION ALL

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]
UNION ALL
SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

INTERSECT

The following statement combines the results with the INTERSECT operator, which returns only those unique rows returned by both queries:

```
SELECT product_id FROM inventories
INTERSECT
SELECT product_id FROM order_items
ORDER BY product_id;
```

MINUS

The following statement combines results with the MINUS operator, which returns only unique rows returned by the first query but not by the second:

```
SELECT product_id FROM inventories
MINUS
SELECT product_id FROM order_items
```

TRIGER

Triggers can be defined on the table, view, schema, or database with which the event is associated. A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

Suppose the database Schema –

```
mysql> desc Student;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| tid   | int(4)   | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)   | YES  |     | NULL    |                |
| subj2 | int(2)   | YES  |     | NULL    |                |
| subj3 | int(2)   | YES  |     | NULL    |                |
| total | int(3)   | YES  |     | NULL    |                |
| per   | int(3)   | YES  |     | NULL    |                |
+-------+----------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SQL Trigger to problem statement.

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per = Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);

Query OK, 1 row affected (0.09 sec)

mysql> select * from Student;

```
+-----+-------+---- ----+--------+-------+--- -----+-------+
| tid | name | subj1 | subj2 | subj3 | total | per  |
+-----+-------+---- ----+--------+-------+--- -----+-------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |  36 |
+-----+-------+---- ----+--------+-------+--- -----+-------+
1 row in set (0.00 sec)
```

Operations in Triggers

We can perform many operations using triggers. Some may be simple and some may be a little complex, but once if we go through the query its easy to understand.

- **DROP A Trigger**

  DROP TRIGGER trigger name;

- **Display A Trigger**

The below code will display all the triggers that are present.
The below code will display all the triggers that are present in a particular database.

```
SHOW TRIGGERS
IN database_name;
```

Example:

```
SHOW TRIGGERS IN edureka;
```

In the above example, all the triggers that are present in the database named Edureka will be displayed.
We also look at some major variants of the triggers that are before insert and after insert. We have already seen a trigger in the example. But with the help of the table let's see how exactly this works.
As we have already understood how to create a trigger, now let's understand the two variants of the trigger those are before insert and after insert. In order to implement them, let's create a student table with various columns as shown below:

```
CREATE TABLE Student(
studentID INT NOT NULL AUTO_INCREMENT,
FName VARCHAR(20),
LName VARCHAR(20),
Address VARCHAR(30),
City VARCHAR(15),
Marks INT,
PRIMARY KEY(studentID)
);
```

Before Insert

```
CREATE TRIGGER calculate
before INSERT
ON student
FOR EACH ROW
SET new.marks = new.marks+100;
```

After insert trigger

```
CREATE TRIGGER total_mark
after insert
ON student
FOR EACH ROW
insert into Final_mark values(new.marks);
```

Here when we insert data to the table, *total_mark trigger* will store the result in the Final_mark table. That was all about the operation on triggers, lets now move ahead and look at its advantages and disadvantages.

Advantages

- Forcing security approvals on the table that are present in the database
- Triggers provide another way to check the integrity of data
- Counteracting invalid exchanges
- Triggers handle errors from the database layer
- Normally triggers can be useful for inspecting the data changes in tables
- Triggers give an alternative way to run scheduled tasks. Using triggers, we don't have to wait for the scheduled events to run because the triggers are invoked automatically before or after a change is made to the data in a table

PL SQL

." PL/SQL is a database-oriented programming language that extends SQL with procedural capabilities. It was developed by Oracle Corporation in the early 90s to boost the capabilities of SQL.

r. No.	Key	SQL	PL/SQL
1	Definition	SQL, is Structural Query Language for database.	PL/SQL is a programming language using SQL for a database.
2	Variables	SQL has no variables.	PL/SQL has variables, data types etc.
3	Control Structures	SQL has no FOR loop, if control and similar structures.	PL/SQL has FOR loop, while loop, if controls and other similar structures.
4	Operations	SQL can execute a single operation at a time.	PL/SQL can perform multiple operation at a time.
5	Language Type	SQL is a declarative language.	PL/SQL is a procedural language.
6	Embedded	SQL can be embedded in a PL/SQL block.	PL/SQL can also be embedded in SQL code.
6	Interaction	SQL directly interacts with database server.	PL/SQL does not directly interacts with database server.
7	Orientation	SQL is data oriented language.	PL/SQL is application oriented language.
8	Objective	SQL is used to write queries, create and execute DDL and DML statments.	PL/SQL is used to write program blocks, functions, procedures, triggers and packages.