Introduction to SQL Statements

SQL, or Structured Query Language, is the standard language for managing and manipulating relational databases. It provides a powerful set of statements that allow you to create, retrieve, update, and delete data with ease.

Data Definition Language (DDL)

1 DDL Statements

DDL statements are used to define and manage the structure of a database, including tables, indexes, and other objects.

2 Key DDL Statements

The most common DDL statements are CREATE, ALTER, DROP, and TRUNCATE.

3 Structural Changes

DDL statements allow you to make structural changes to your database, such as adding new tables, modifying existing tables, or removing unnecessary objects.

SQL

```sql
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

SQL

```sql
ALTER TABLE Customers ADD Email varchar(255);
```

SQL

```sql
ALTER TABLE Customers DROP COLUMN Email;
```

SQL

```sql
TRUNCATE TABLE Categories;
```

SQL

```sql
DROP TABLE Shippers;
```

Data Definition Language (DDL)

CREATE

The CREATE statement is used to define and build new database objects, such as tables, indexes, and views.

ALTER

The ALTER statement is used to modify the structure of an existing database object, such as adding, removing, or changing columns in a table.

DROP

The DROP statement is used to remove an existing database object, such as a table or index, from the database.

TRUNCATE

The TRUNCATE statement is used to remove all data from a table, while keeping the table structure intact.

Data Definition Language (DDL)

Example: Creating a New Table

To create a new table for customer information, you can use the CREATE TABLE statement:

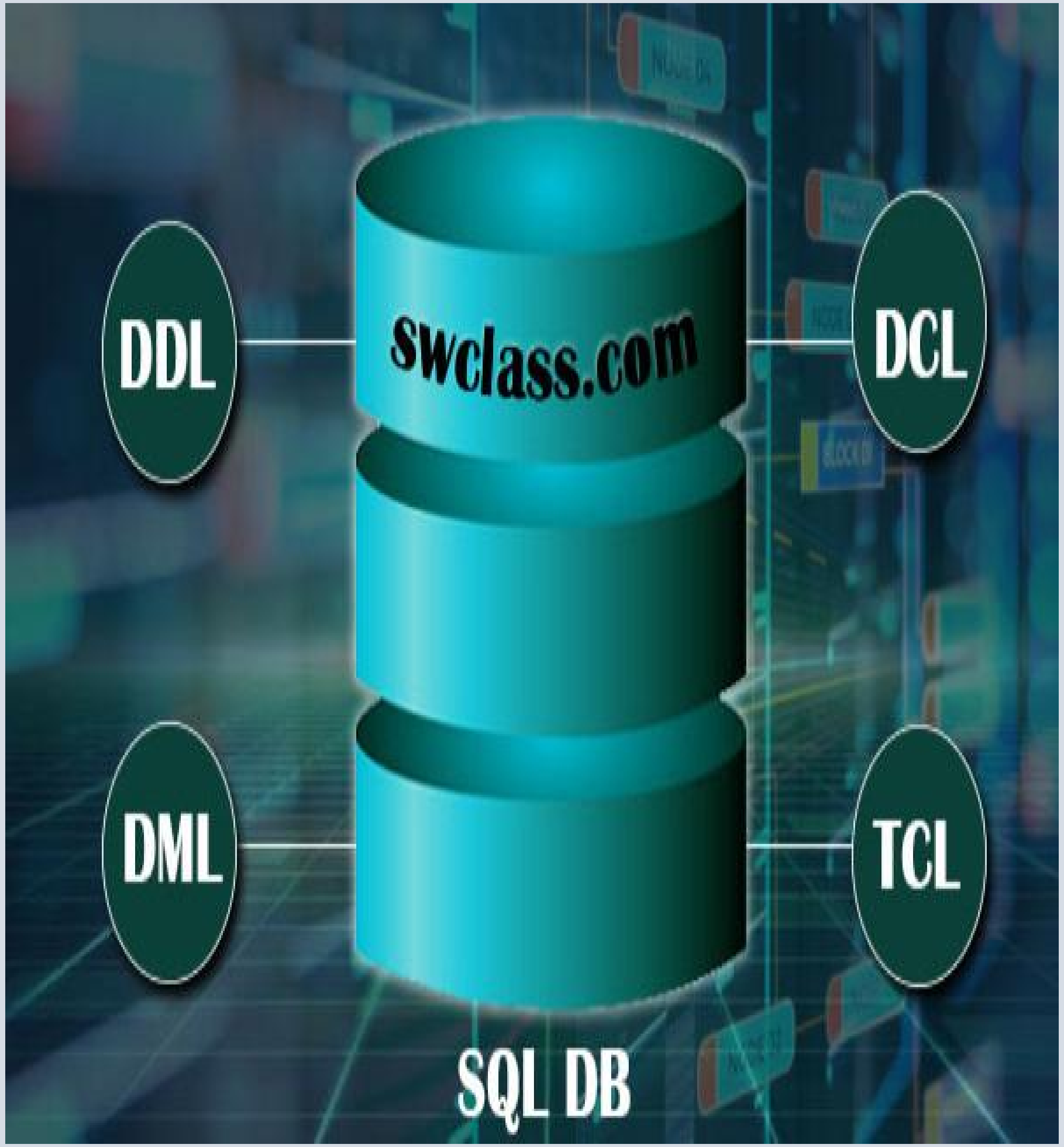
```
CREATE TABLE Customers (
    CustomerID INT PRIMARY
KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    Phone VARCHAR(20)
);
```

Table Structure

This creates a new table called "Customers" with five columns: CustomerID, FirstName, LastName, Email, and Phone.

Primary Key

The CustomerID column is defined as the primary key, which ensures unique identification of each customer record.



Data Query Language (DQL)

1 **DQL Statements**

DQL statements are used to retrieve and query data from a database, allowing you to filter, sort, and aggregate information.

2 **Key DQL Statements**

The most common DQL statement is SELECT, which is used to retrieve data from one or more tables.

3 **Powerful Querying**

DQL statements provide powerful ways to filter, sort, and group data, making it easier to find the information you need.

Data Query Language (DQL)

SELECT Statement

The SELECT statement is used to retrieve data from a database. It allows you to specify the columns you want to retrieve and the conditions for the data you want to retrieve.

Filtering with WHERE

The WHERE clause in a SELECT statement allows you to filter the results based on specific criteria, such as selecting only customers with a specific email domain.

Sorting with ORDER BY

The ORDER BY clause in a SELECT statement allows you to sort the results in ascending or descending order based on one or more columns.

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

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

```
SELECT * FROM Customers
WHERE CustomerID > 80;
```

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

```
SELECT * FROM Products
ORDER BY Price;
```

```
SELECT * FROM Products
ORDER BY Price DESC;
```

Data Query Language (DQL)

1

Example: Retrieving Customer Information

To retrieve customer information based on specific criteria, you can use a SELECT statement like this:

2

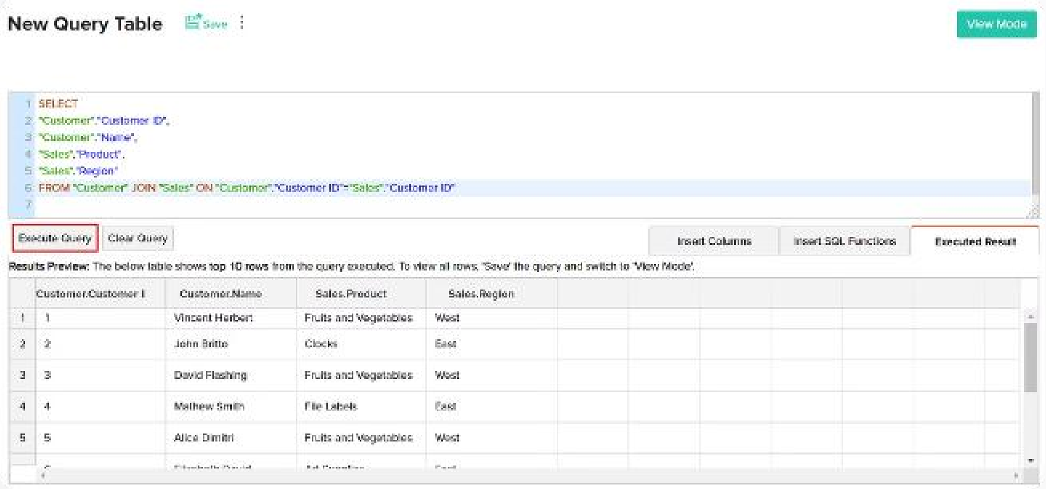
Query

SELECT FirstName, LastName, Email, Phone FROM Customers WHERE Email LIKE '%@example.com';

3

Results

This query will return the first name, last name, email, and phone number of all customers with an email address that ends with '@example.com'.



ROLE_GROUPS

Id	Name
1	SUBSCRIBER
2	NON_SUBSCRIBER

ROLES

Id	Name
1	Profile
2	Reporting

ACTIONS

Id	Name
1	VIEW
2	EXECUTE

OPERATIONS

Id	Name
1	USER_PROFILE
2	EXPENSE_REPORT
3	INVENTORY_LIST

PERMISSIONS

Id	Name
1	VIEW_USER_PROFILE
2	EXECUTE_EXPENSE_REPORT
3	VIEW_INVENTORY_LIST

MODULES

Id	Name
1	PROFILE
2	REPORTS
3	SUBSCRIPTION

Data Control Language (DCL)

Using the 'GRANT' Statement

The 'GRANT' statement is used to give user's permissions to a database. The syntax for the 'GRANT' statement is:

SYNTAX

```
GRANT privilege_name ON
object_name TO {user_name
|PUBLIC |role_name} [WITH
GRANT OPTION];
```

Let's consider an example:

```
GRANT SELECT, INSERT, DELETE
ON my db TO
'user1'@'localhost';
```

Using the 'REVOKE' Statement

The 'REVOKE' statement is used to take back permissions from a user. The syntax for the 'REVOKE' statement is:

```
REVOKE privilege_name ON
object_name FROM {user_name
|PUBLIC |role_name}
```

Here is an example:

```
REVOKE INSERT ON mydb FROM
'user1'@'localhost';
```

This command revokes the INSERT permission on the database 'mydb' from 'user1'.

Data Manipulation Language (DML)

1 **DML Statements**

DML statements are used to create, modify, and delete data in a database, allowing you to manage the actual content of your database.

2 **Key DML Statements**

The most common DML statements are INSERT, UPDATE, and DELETE.

3 **Data Management**

DML statements provide the tools to add new data, update existing data, and remove data from your database as needed.


```sql
INSERT INTO Customers (CustomerName, ContactName,
Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

```sql
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

```sql
DELETE FROM Customers WHERE CustomerName='Alfreds
Futterkiste';
```

Data Manipulation Language (DML)

INSERT

The INSERT statement is used to add new data to a table, such as inserting a new customer record.

UPDATE

The UPDATE statement is used to modify existing data in a table, such as updating a customer's email address or phone number.

DELETE

The DELETE statement is used to remove data from a table, such as removing a customer record that is no longer needed.

Data Manipulation Language (DML)

1

Example: Adding a New Customer

To add a new customer to the Customers table, you can use the INSERT statement:

2

Updating Customer Information

To update an existing customer's information, you can use the UPDATE statement:

3

Deleting a Customer

To remove a customer from the Customers table, you can use the DELETE statement:

Transaction Control Language (TCL)

ROLLBACK
Rollback command This command restores the database to last commited state.

It is also used with SAVEPOINT command to jump to a save point in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use

the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

COMMIT
COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent,

until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

Transaction Control Language (TCL)

SAVEPOINT
SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

SAVEPOINT savepoint_name;..

In short, using this command we can **name** the different states of our data in any table and then rollback to that state using

the ROLLBACK command whenever required.

RELATIONAL ALGEBRA

- Relational algebra is a procedural query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.
- On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it. We will discuss relational calculus in a separate tutorial.

RELATIONAL ALGEBRA

- Types of operations in relational algebra We have divided these operations in two categories:
- 1. **Basic Operations** 2. **Derived Operations**
- Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Intersection Operator (∩)
5. Set Difference (-)
6. Cartesian product (X)
7. Rename (ρ)

RELATIONAL ALGEBRA

- Types of operations in relational algebra We have divided these operations in two categories:
- 1. **Basic Operations** 2. **Derived Operations**

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (, ,)
3. Intersection (∩)
4. Division (÷)

RELATIONAL ALGEBRA

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

If you understand little bit of SQL then you can think of it as a where clause in SQL, which is used for the same purpose.

Syntax of Select Operator (σ)

σ Condition/Predicate(Relation/Table name)

RELATIONAL ALGEBRA

Select Operator is denoted by sigma (σ) and it is used to find the

```
σ Condition/Predicate(Relation/Table name)
```

Select Operator (σ) Example

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

RELATIONAL ALGEBRA

Select Operator is denoted by sigma (σ) and it is used to find the

Query:

```
σ Customer_City="Agra" (CUSTOMER)
```

Output:

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra

RELATIONAL ALGEBRA

Select Operator is denoted by sigma (σ) and it is used to find the

In general, we allow comparisons using =, >,< , ≥ in the selection predicate. Furthermore, we can combine several predicates into a larger predicate by using the connectives and (∧), or (∨), and not (¬).

σbranch-name = "Perryridge" ∧ amount>1200 (loan)

RELATIONAL ALGEBRA

Project Operator (∏)

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the Select statement in SQL.

Syntax of Project Operator (∏)

∏ column_name1, column_name2, ..., column_nameN(table_name)

Project Operator (∏) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator ∏.

RELATIONAL ALGEBRA

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

∏ Customer_Name, Customer_City (CUSTOMER)

Output:

Customer_Name	Customer_City
Steve	Agra
Raghu	Agra
Chaitanya	Noida
Ajeet	Delhi
Carl	Delhi

RELATIONAL ALGEBRA

Union Operator ( )

Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

Lets discuss union operator a bit more. Lets say we have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

Note: The rows (tuples) that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Syntax of Union Operator ( )

table_name1 ∪ table_name2

RELATIONAL ALGEBRA

Union Operator (∪) Example

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

RELATIONAL ALGEBRA

Query:

```
∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------------
Aditya
Carl
Paul
Lucy
Rick
Steve
```

RELATIONAL ALGEBRA

Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Lets say we have two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Note: Only those rows that are present in both the tables will appear in the result set.

Syntax of Intersection Operator (∩)

table_name1 ∩ table_name2

RELATIONAL ALGEBRA

Intersection Operator (∩) Example

Lets take the same example that we have taken above.

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16

RELATIONAL ALGEBRA

Query:

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------------
Aditya
Steve
Paul
Lucy
```

RELATIONAL ALGEBRA

Set Difference (-)

Set Difference is denoted by – symbol. Lets say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference R1 – R2.

Syntax of Set Difference (-)

table_name1 - table_name2

Set Difference (-) Example

Lets take the same tables COURSE and STUDENT that we have seen above.

RELATIONAL ALGEBRA

Query:

Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

```
∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)
```

Output:

```
Student_Name
-------------------
Carl
Rick
```

RELATIONAL ALGEBRA

Cartesian product (X)

Cartesian Product is denoted by X symbol. Lets say we have two relations R1 and R2 then the cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2. Lets take an example of this, you will be able to understand this.

Syntax of Cartesian product (X)

R1 X R2

Cartesian product (X) Example

Table 1: R

Col_A	Col_B
AA	100
BB	200
CC	300

Table 2: S

Col_X	Col_Y
XX	99
YY	11
ZZ	101

RELATIONAL ALGEBRA

Query:

Let's find the cartesian product of table R and S.

```
R X S
```

Output:

Col_A	Col_B	Col_X	Col_Y
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Note: The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has 3×3 = 9 rows.

RELATIONAL ALGEBRA

Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

Rename (ρ) Syntax:

ρ(new_relation_name, old_relation_name)

Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

RELATIONAL ALGEBRA

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

```
ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER))
```

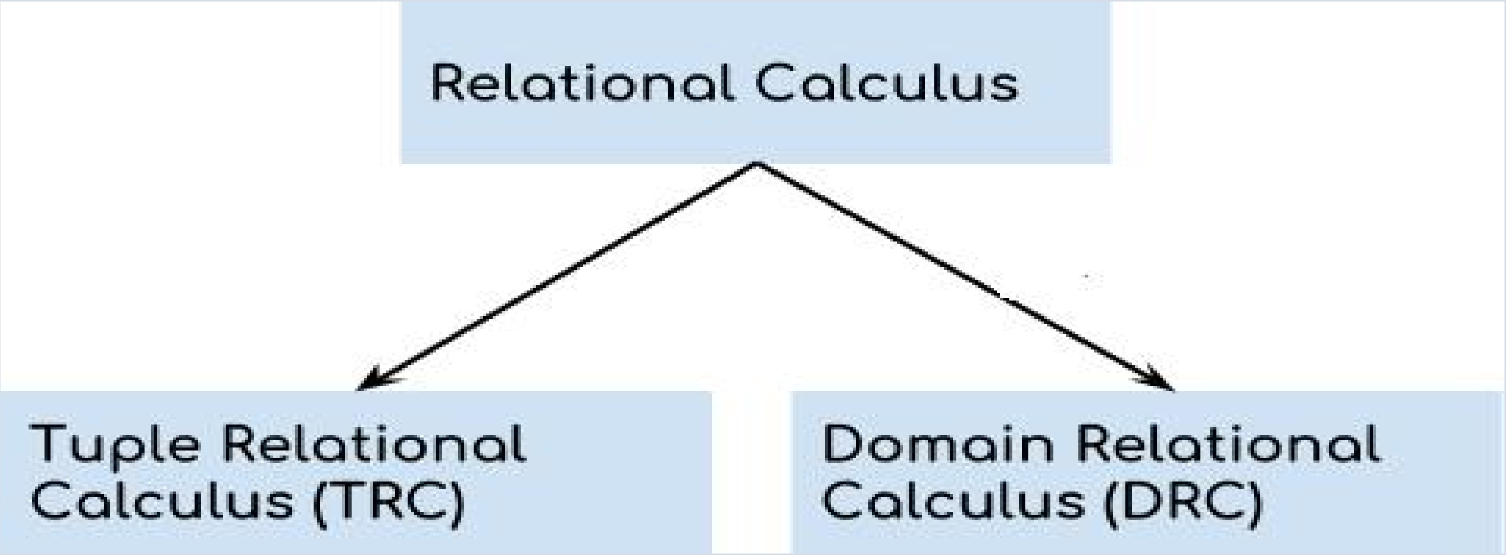
Output:

```
CUST_NAMES
---------------
Steve
Raghu
Chaitanya
Ajeet
Carl
```

RELATIONAL CALCULUS

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn’t tell how to retrieve it.

Types of Relational Calculus



RELATIONAL CALCULUS

Tuple Relational Calculus (TRC)

Tuple relational calculus is used for selecting those tuples that satisfy the given condition.

Notation:

A Query in the tuple relational calculus is expressed as following notation

> {T | P (T)} or {T | Condition (T)}

T is the resulting tuples

P(T) is the condition used to fetch T.

RELATIONAL CALCULUS

Table: Student

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31
Rajeev	Bhatia	27
Carl	Pratap	28

Lets write relational calculus queries.

Query to display the last name of those students where age is greater than 30

```
{ t.Last_Name | Student(t) AND t.age > 30 }
```

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.

The result of the above query would be:

```
Last_Name
---------------
Singh
```

RELATIONAL CALCULUS

Query to display all the details of students where Last name is ‘Singh’

```
{ t | Student(t) AND t.Last_Name = 'Singh' }
```

Output:

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31

RELATIONAL CALCULUS

Domain Relational Calculus (DRC)

In domain relational calculus the records are filtered based on the domains. Again we take the same table to understand how DRC works.

Many of the calculus expressions involves the use of Quantifiers.
There are two types of quantifiers:

Universal Quantifiers: The universal quantifier denoted by ∀ is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.

Existential Quantifiers: The existential quantifier denoted by ∃ is read as for all which means that in a given set of tuples there is at least one occurrences whose value satisfy a given condition.

Notation:

{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

RELATIONAL CALCULUS

Table: Student

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31
Rajeev	Bhatia	27
Carl	Pratap	28

Query to find the first name and age of students where student age is greater than 27

{< First_Name, Age > | ∈ Student ∧ Age > 27}

Note:

The symbols used for logical operators are: ∧ for AND, ∨ for OR and ┓ for NOT.

Output:

First_Name	Age
Ajeet	30
Chaitanya	31
Carl	28

VIEWS IN DBMS

Views in SQL are a type of **virtual table** that simplifies how users interact with data across one or more tables.
Unlike **traditional tables**, a view in **SQL** does not store data on disk; instead, it dynamically retrieves data based on a pre-defined query each time it's accessed.
SQL views are particularly useful for managing complex queries, enhancing security, and presenting data in a simplified format.

Views in SQL are considered as a virtual table. A view also contains rows and columns.
To create the view, we can select the fields from one or more tables present in the database.
A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

VIEWS IN DBMS

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

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

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

VIEWS IN DBMS

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.
In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME; To display

VIEWS IN DBMS

SELECT * FROM MarksView;

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

```
DROP VIEW view_name;
```

Example:

If we want to delete the View **MarksView**, we can do this as: DROP VIEW MarksView;

NO-SQL

NoSQL stands for "not only SQL" and refers to a type of database that stores data in a non-tabular format. NoSQL databases are designed to handle large amounts of unstructured data, such as documents, graphs, key-value, and wide columns.
NoSQL databases are increasingly used in big data and real-time web applications.
simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), finer control over availability and limiting the object-relational impedance mismatch.

Types of NoSQL databases

The data structures used by NoSQL databases (e.g. key–value pair, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL.
Key-value Key-value stores pair keys and values using a hash table. Key- value types are best when a key is known and the associated value for the key is unknown.

NO-SQL

Document databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, Booleans, arrays, or objects.

Document databases extend the concept of the key-value database by organising entire documents into groups called collections. They support nested key-value pairs and allow queries on any attribute within a document.

Wide-column stores store data in tables, rows, and dynamic columns.

Columnar, wide-column or column-family databases efficiently store data and query across rows of sparse data and are advantageous when querying across specific columns in the database.

Graph databases store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

Graph databases use a model based on nodes and edges to represent interconnected data—such as relationships between people in a social network—and offer simplified storage and navigation through complex relationships

NO-SQL

RDBMS vs NoSQL: Data Modeling Example

Let's consider an example of storing information about a user and their hobbies. We need to store a user's first name, last name, cell phone number, city, and hobbies.

In a relational database, we'd likely create two tables: one for Users and one for Hobbies.

Users

ID	first_name	last_name	contact	city
1	John	Yepp	8125552344	Vadodara

Hobbies

ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles

NO-SQL

In order to retrieve all of the information about a user and their hobbies, information from the Users table and Hobbies table will need to be joined together.

The data model we design for a NoSQL database will depend on the type of NoSQL database we choose. Let's consider how to store the same information about a user and their hobbies in a document database like MongoDB.

```
{
  "_id": 1,
  "first_name": "John",
  "last_name": "Yepp",
  "contact": "8125552344",
  "city": "Vadodara",
  "hobbies": ["scrapbooking", "eating waffles", "working"]
}
```

NO-SQL

In order to retrieve all of the information about a user and their hobbies, a single document can be retrieved from the database. No joins are required, resulting in faster queries.

Advantages of NoSQL:

There are many advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

High scalability –

NoSQL database use sharding for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding. Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement.

Examples of horizontal scaling databases are MongoDB, Cassandra etc. NoSQL canhandle huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in efficient manner.

High availability –

Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.

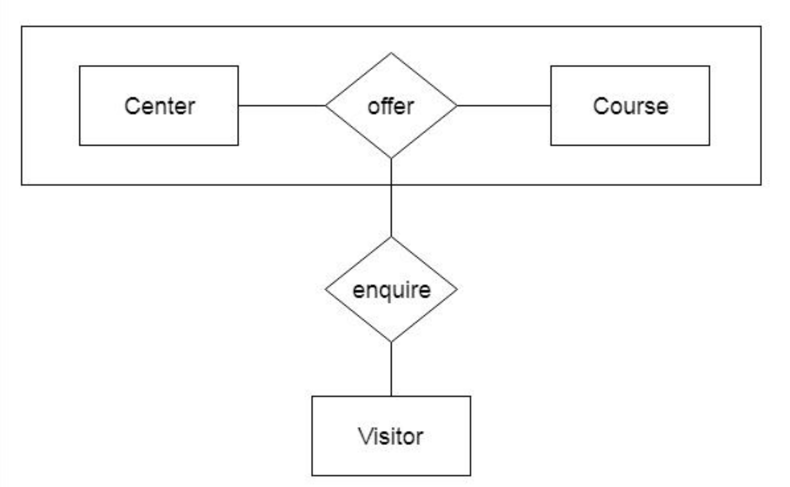
NO-SQL

When should NoSQL be used:

1. When huge amount of data need to be stored and retrieved .
2. The relationship between the data you store is not that important
3. The data changing over time and is not structured.
4. Support of Constraints and Joins is not required at database level
5. The data is growing continuously and you need to scale the database regular to handle the data.

AGGREGATION IN DBMS

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.
For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.



When it comes to statistical information in a DBMS, there are extraordinary gears and approaches to collect it.

SUM: It is used for adding things up.
AVG: It is used for finding the middle ground.
Minimum (MIN): It is used to get the singling out of the smallest values.
Maximum (MAX): It is used for singling out for the largest values.
COUNT: It is used for a headcount and **DISTINCT** for counting unique items.