SQL

Data

Data is a collection of raw facts, figures, symbols, or values that by itself may not have meaning.

It can be **numbers, text, images, audio, video**, or any input that can be processed by a computer.

Becomes Information: When data is processed, organized, or structured, it becomes **information**.

Example:

- **Data**: "John", "95", "Math"
- **Information**: "John scored 95 marks in Math."

Types of data: 1) Qualitative data(categorical), 2) Quantitative data (Numerical) -countable and measurable (subtype – discrete and continuous)

Relational Database

A relational database is a type of database that stores data in tables(relations) made up of rows and columns. It uses relationships between the data to organize and retrieve information efficiently.

A Relational Database stores data in the form of related tables.

Row → records, columns → attributes or fields, table → entity

Database Management System (DBMS)

A Database Management System (DBMS) is software that helps you create, store, manage, retrieve, and update data in databases efficiently.

It acts as a bridge between users (or applications) and the actual data stored in databases.

DBMS	RBMS
Data is stored as file system	Data is stored in tabular forms
No relationship btw data	Relationship btw data is also represented in tables
Hierarchical Form/ Navigation Form	Table
Can handle small data	Can handle large amount of data
Example: file system, XML	Example: Sql, oracle

Schema – The Design / Blueprint

A **schema** is the **logical structure** of the database.

It defines:

- Tables (relations)
- Fields (columns)
- Data types
- Relationships

- Constraints (like primary keys, foreign keys)

Instance – The Data / Snapshot

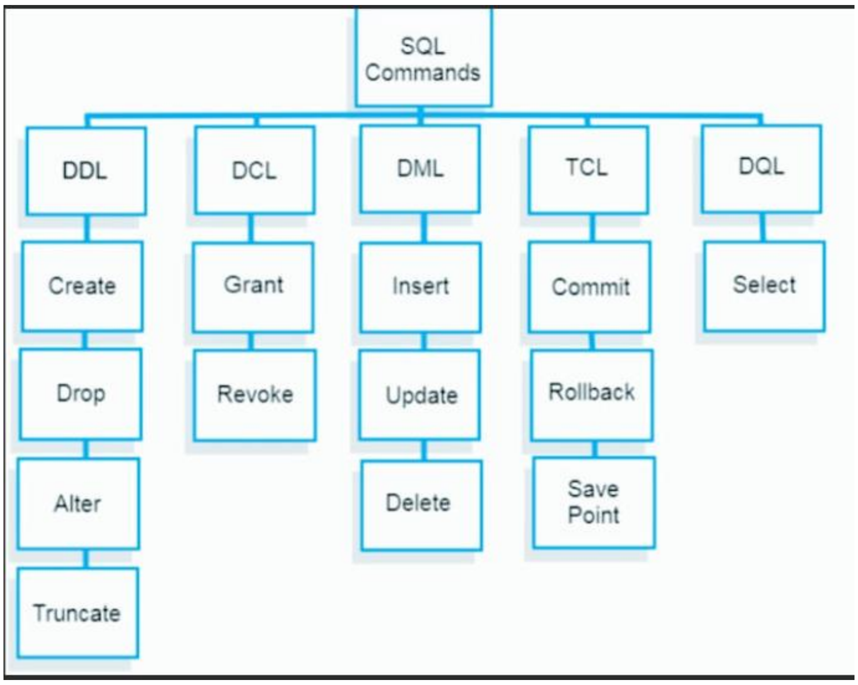
An **instance** is the **actual content/data** stored in the database at a given moment.

It changes frequently as data is inserted, updated, or deleted.

SQL Data types

1. Numeric Data type – int
2. Character – varchar(n) → variable-length, char(n) → fixed-length(n), text – (large text data like long paragraphs)
3. Boolean
4. Date and time

Data Type	**Description**	**Example**
DATE	Date only	'2025-07-15'
TIME	Time only	'14:30:00'
DATETIME	Date and time	'2025-07-15 14:30:00'
TIMESTAMP	Date + time with time zone info	'2025-07-15 14:30:00+05:30'
YEAR	Year only	2025



DDL - Data Definition Language

That deals with **defining, altering, and deleting database structures** such as **tables, schemas, indexes, and constraints**.

- DDL changes the structure of a database, not its data.

Command

Create → creates a new table, database

Alter → Modifies an existing table or structure

Drop → Deletes a table or database permanently

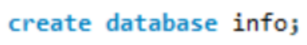
Truncate → Deletes all rows from a table (faster than delete)

CREATE

CREATE DATABASE statement is used to create a new SQL database.

Syntax

CREATE DATABASE *databasename*;



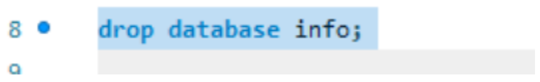
```
create database info;
```

Drop

DROP DATABASE statement is used to drop an existing SQL database.

Syntax

DROP DATABASE *databasename*;

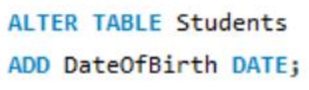


```
drop database info;
```

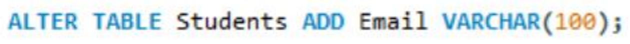
Alter

ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

Syntax: ALTER TABLE table_name ADD column_name datatype;



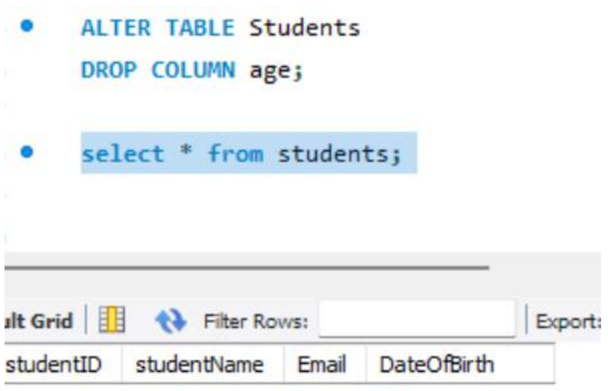
```
ALTER TABLE Students
ADD DateOfBirth DATE;
```



```
ALTER TABLE Students ADD Email VARCHAR(100);
```

ALTER TABLE - DROP COLUMN

ALTER TABLE table_name DROP COLUMN column_name;



```sql
ALTER TABLE Students
DROP COLUMN age;

select * from students;
```

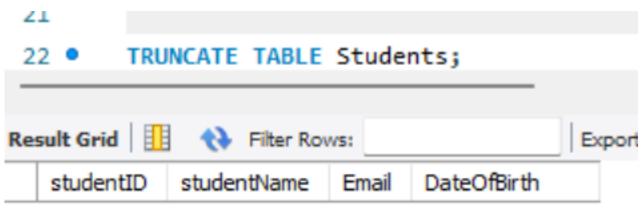
```
studentID  studentName  Email  DateOfBirth
```

TRUNCATE

The TRUNCATE command is used to quickly delete all rows from a table, while keeping the table structure intact for future use.

Syntax:

TRUNCATE TABLE table_name;



```sql
TRUNCATE TABLE Students;
```

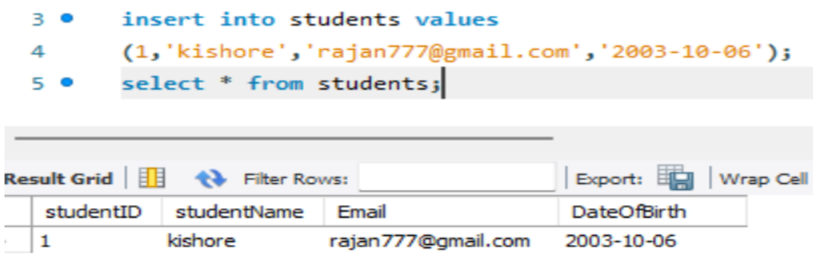
```
studentID  studentName  Email  DateOfBirth
```

Delete	Truncate	Drop
Delete specific or all rows (using condition)	Delete all rows quickly	Remove entire table from DB

DML – Data Manipulation Language

It includes SQL commands used to **insert, update, delete, and retrieve data** from database tables.

Insert



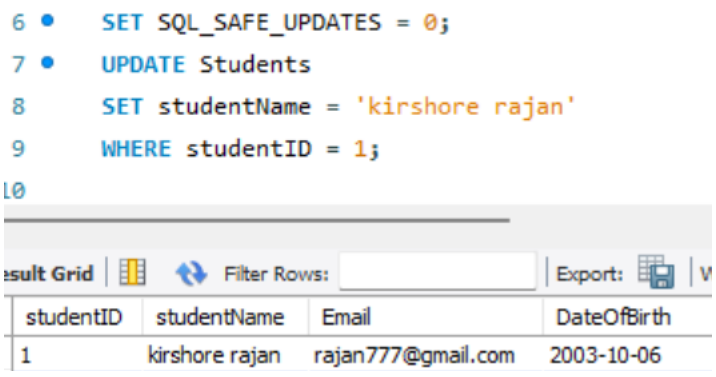
```sql
insert into students values
(1,'kishore','rajan777@gmail.com','2003-10-06');
select * from students;
```

```
studentID  studentName  Email               DateOfBirth
1          kishore      rajan777@gmail.com  2003-10-06
```

Update

Syntax:

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



```
SET SQL_SAFE_UPDATES = 0;
UPDATE Students
SET studentName = 'kirshore rajan'
WHERE studentID = 1;
```

studentID	studentName	Email	DateOfBirth
1	kirshore rajan	rajan777@gmail.com	2003-10-06

DCL (Data Control Language)

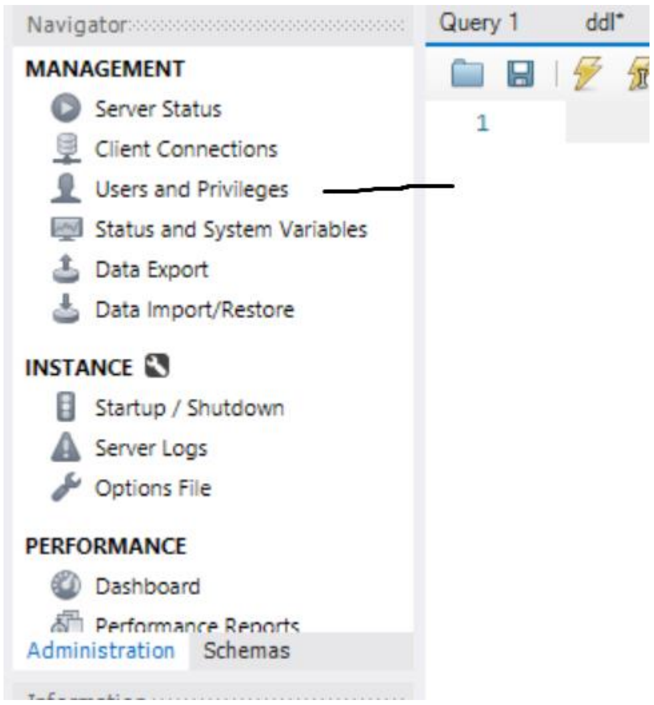
DCL that deals with permissions and access control to the database and its objects.

Grant → Gives a user privileges to the database

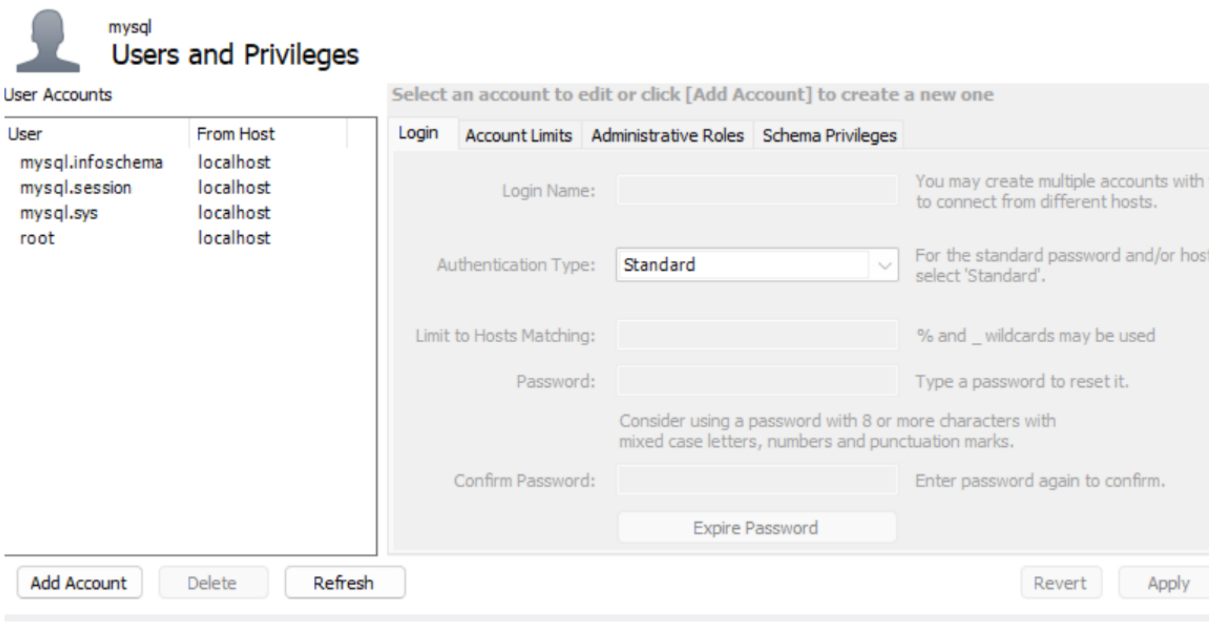
Revoke → removes access privileges from a user.

To create new user account

Goto Administration, click users and privileges



Click on add Account



mysql

Users and Privileges

User Accounts

User	From Host
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

Select an account to edit or click [Add Account] to create a new one

Login | Account Limits | Administrative Roles | Schema Privileges

Login Name: — You may create multiple accounts with to connect from different hosts.

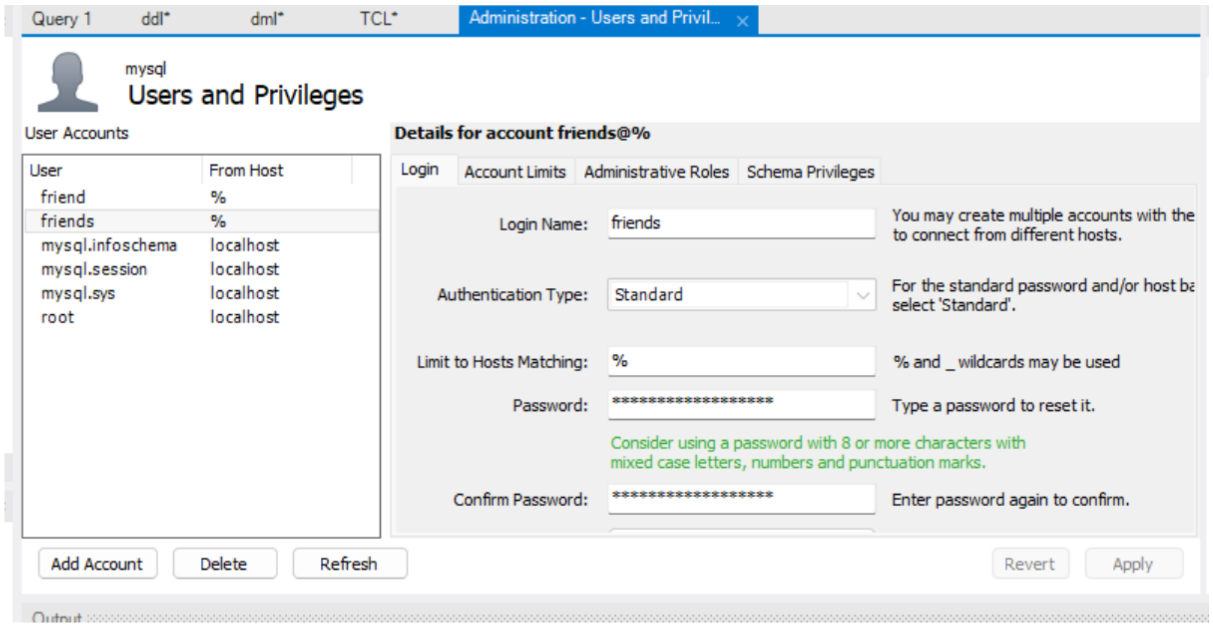
Authentication Type: Standard — For the standard password and/or hos select 'Standard'.

Limit to Hosts Matching: — % and _ wildcards may be used

Password: — Type a password to reset it.

Consider using a password with 8 or more characters with mixed case letters, numbers and punctuation marks.

Confirm Password: — Enter password again to confirm.



mysql

Users and Privileges

User Accounts

User	From Host
friend	%
friends	%
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

Details for account friends@%

Login | Account Limits | Administrative Roles | Schema Privileges

Login Name: friends — You may create multiple accounts with the to connect from different hosts.

Authentication Type: Standard — For the standard password and/or host ba select 'Standard'.

Limit to Hosts Matching: % — % and _ wildcards may be used

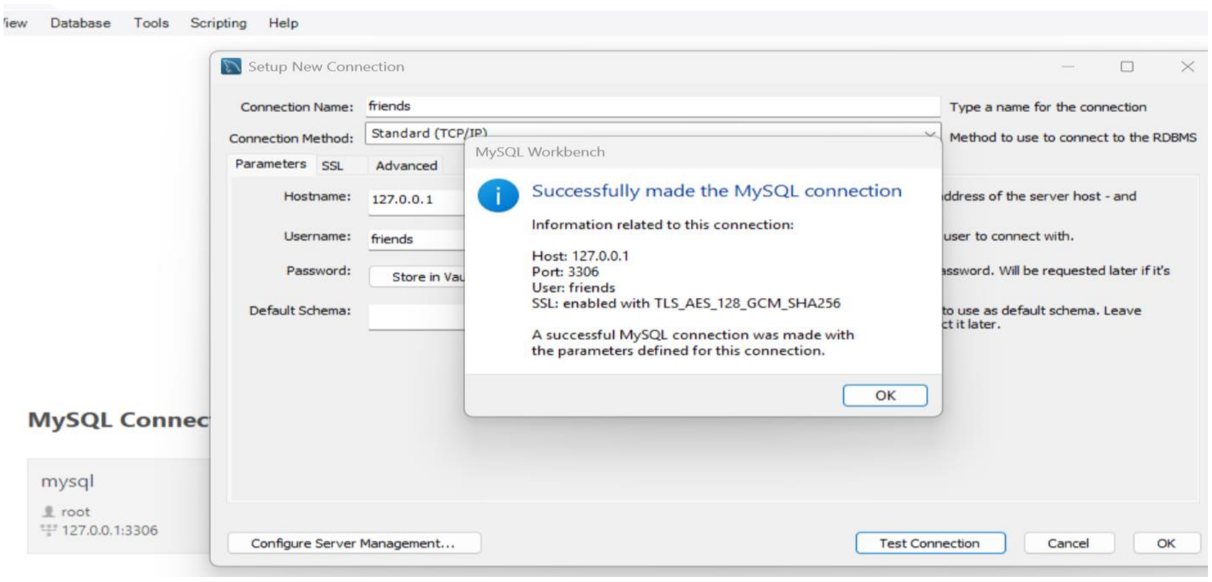
Password: ******************* — Type a password to reset it.

Consider using a password with 8 or more characters with mixed case letters, numbers and punctuation marks.

Confirm Password: ******************* — Enter password again to confirm.

NewUser#0

Check and establish the connection for new user (username – friends, password – NewUser#0)



Connection Name: friends — Type a name for the connection
Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS
Hostname: 127.0.0.1 — ddress of the server host - and
Username: friends — user to connect with.
Password: Store in Vau — ssword. Will be requested later if it's
Default Schema: — to use as default schema. Leave / ct it later.

MySQL Workbench

Successfully made the MySQL connection

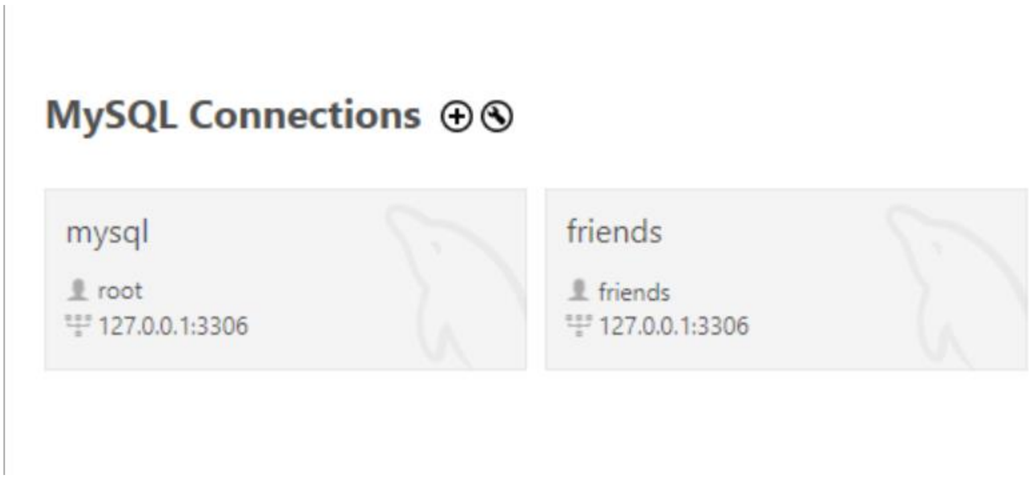
Information related to this connection:

Host: 127.0.0.1
Port: 3306
User: friends
SSL: enabled with TLS_AES_128_GCM_SHA256

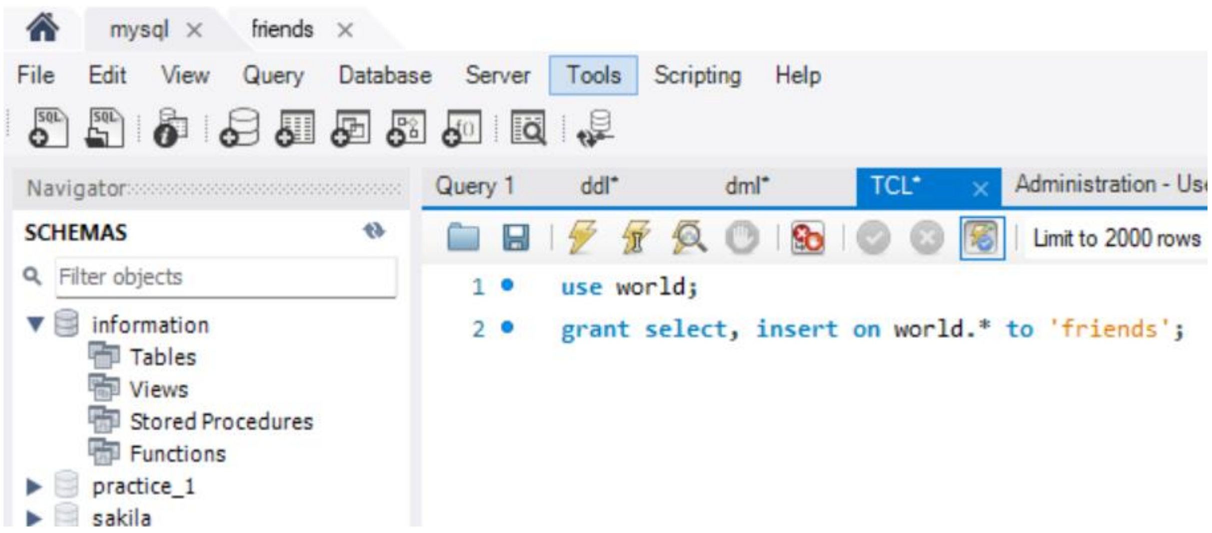
A successful MySQL connection was made with the parameters defined for this connection.

OK

New server for friends created.

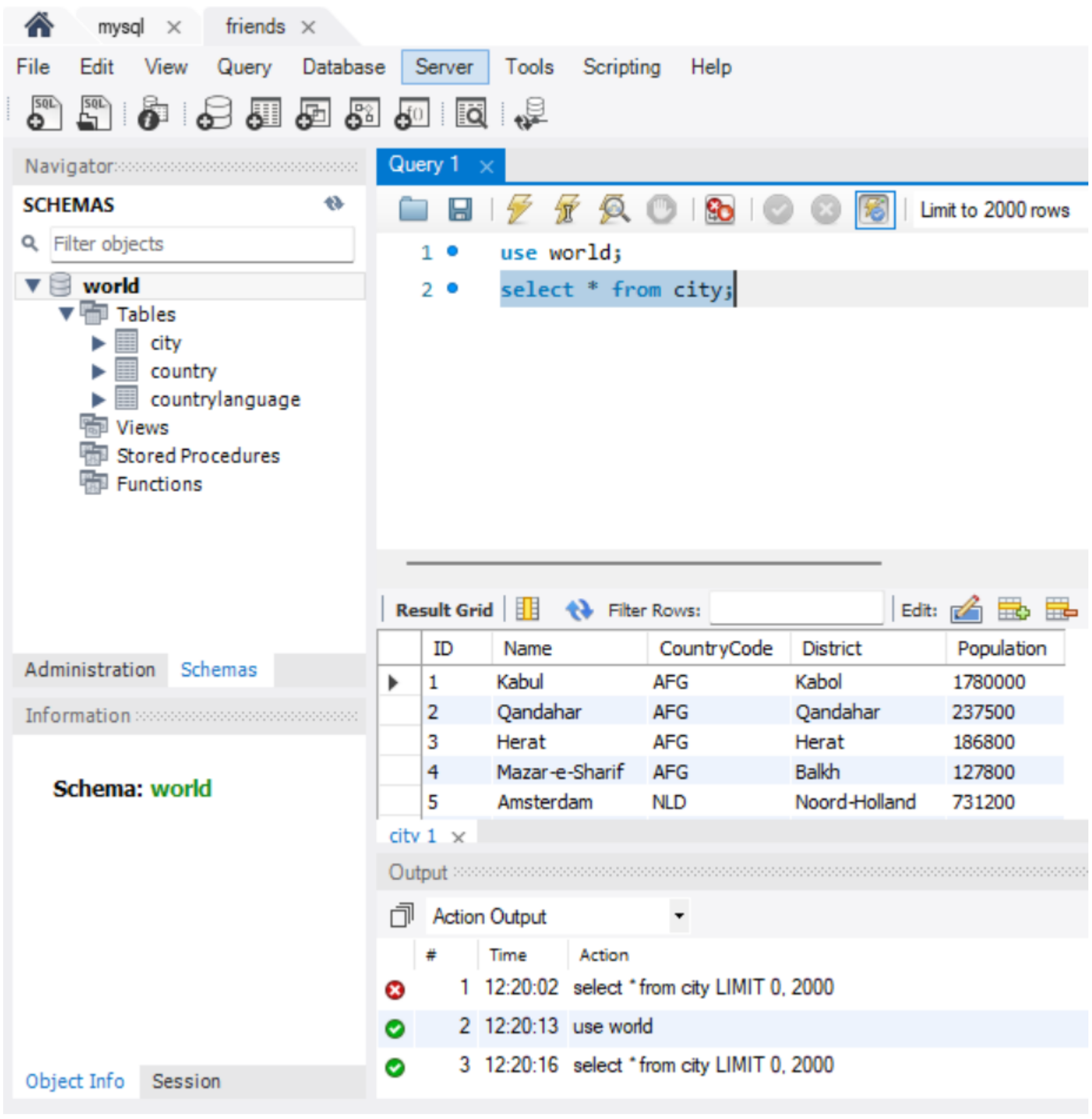


Grant permission to ‘friends’ server from ‘mysql’ server – permission given only to view and insert the records.



```
use world;
grant select, insert on world.* to 'friends';
```

Check the operation from 'friends' server

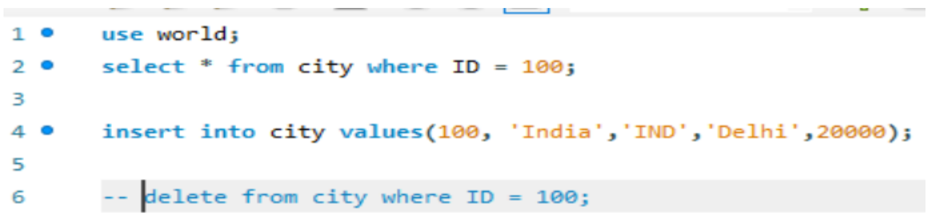


Schema: world

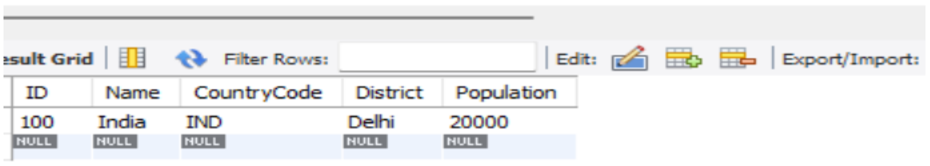
```
use world;
select * from city;
```

	ID	Name	CountryCode	District	Population
▶	1	Kabul	AFG	Kabol	1780000
	2	Qandahar	AFG	Qandahar	237500
	3	Herat	AFG	Herat	186800
	4	Mazar-e-Sharif	AFG	Balkh	127800
	5	Amsterdam	NLD	Noord-Holland	731200

	#	Time	Action
✗	1	12:20:02	select * from city LIMIT 0, 2000
✓	2	12:20:13	use world
✓	3	12:20:16	select * from city LIMIT 0, 2000



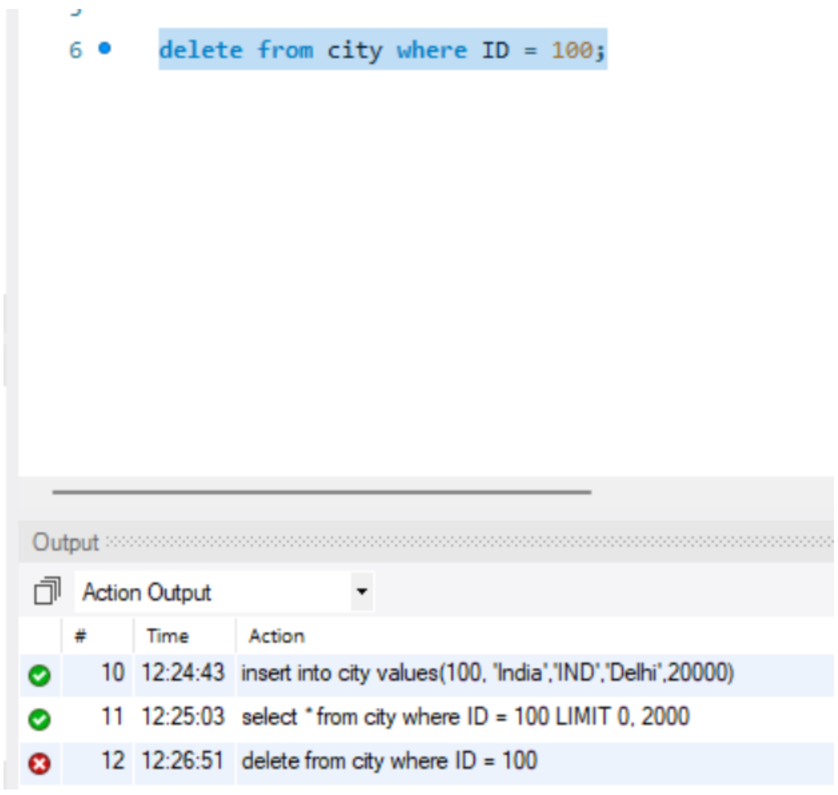
```
use world;
select * from city where ID = 100;

insert into city values(100, 'India','IND','Delhi',20000);

-- delete from city where ID = 100;
```



ID	Name	CountryCode	District	Population
100	India	IND	Delhi	20000
NULL	NULL	NULL	NULL	NULL

Delete is not possible -

delete from city where ID = 100	Error Code: 1142. DELETE command denied to user 'friends'@'localhost' for table 'city'	0.000 sec



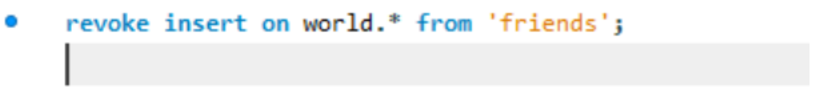
```
delete from city where ID = 100;
```

Output

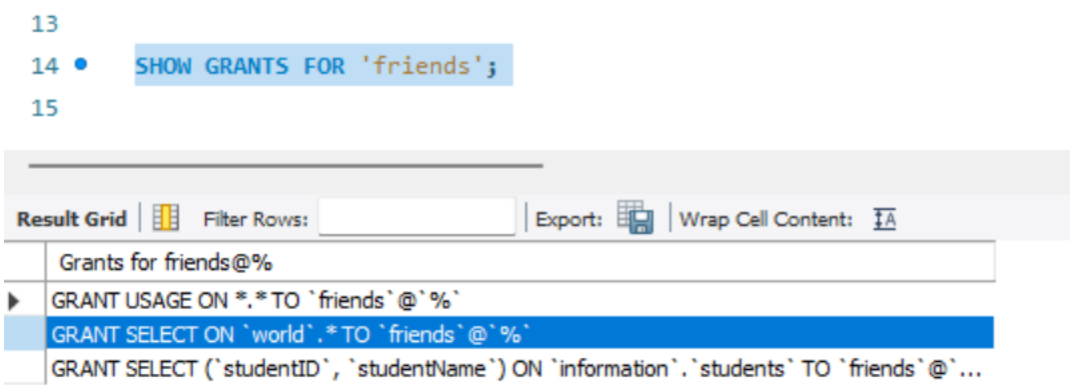
Action Output

	#	Time	Action
✓	10	12:24:43	insert into city values(100, 'India','IND','Delhi',20000)
✓	11	12:25:03	select * from city where ID = 100 LIMIT 0, 2000
✗	12	12:26:51	delete from city where ID = 100

Revoke "insert" operation to friends server.



```
revoke insert on world.* from 'friends';
```



```
SHOW GRANTS FOR 'friends';
```

```
Grants for friends@%
GRANT USAGE ON *.* TO `friends`@`%`
GRANT SELECT ON `world`.* TO `friends`@`%`
GRANT SELECT (`studentID`, `studentName`) ON `information`.`students` TO `friends`@`...
```

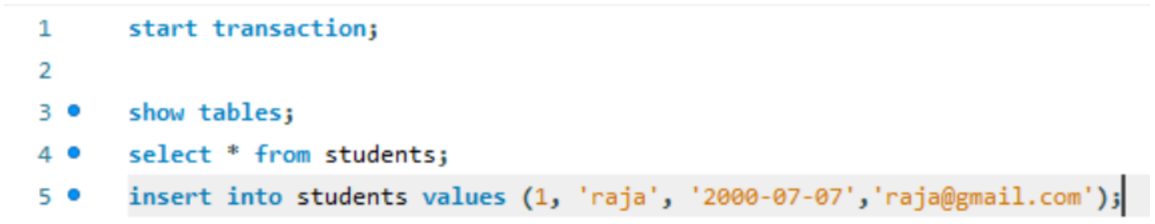
TCL (Transaction Control Language)

TCL commands are used to **manage transactions** in SQL — allowing you to **save, undo, or partially undo** changes made to the database during a session.

START TRANSACTION.

- Begins a new transaction.
- Any changes (like INSERT, UPDATE, DELETE) after this point are **temporary** until you explicitly COMMIT.

Insert operation is done.



```
start transaction;

show tables;
select * from students;
insert into students values (1, 'raja', '2000-07-07','raja@gmail.com');
```

Option 1: Save the Change

If you're happy with the insert and want to **make it permanent → COMMIT;**

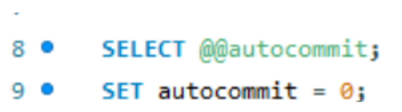
Auto-Commit Might Be Enabled (MySQL Default)

In MySQL (and some other DBMSs), if **auto-commit mode** is **ON**, then every INSERT, UPDATE, or DELETE is **immediately committed**, even if you write START TRANSACTION.

Check Auto-Commit: → SELECT @@autocommit;

- If it returns 1 → Auto-commit is **enabled**.
- If it returns 0 → Auto-commit is **disabled**

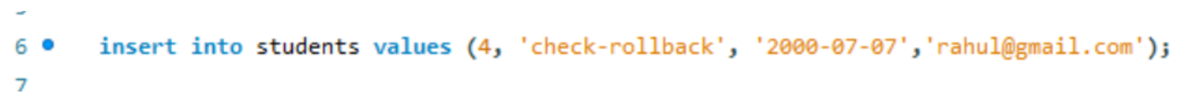
To Disable Auto-Commit: → set autocommit = 0



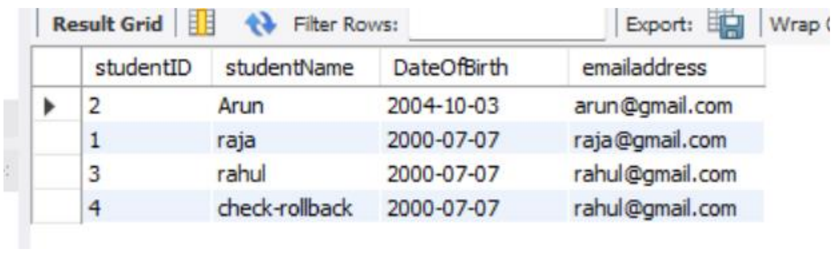
```
SELECT @@autocommit;
SET autocommit = 0;
```

Option 2: Undo the Change

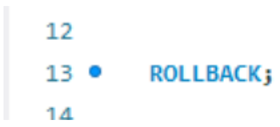
If you **made a mistake** or want to cancel the insert → ROLLBACK;

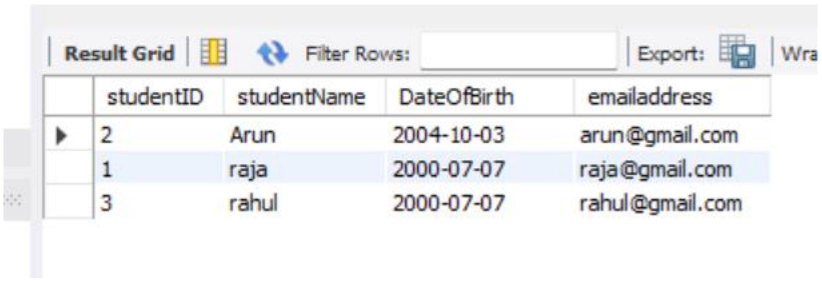


```
insert into students values (4, 'check-rollback', '2000-07-07','rahul@gmail.com');
```



studentID	studentName	DateOfBirth	emailaddress
2	Arun	2004-10-03	arun@gmail.com
1	raja	2000-07-07	raja@gmail.com
3	rahul	2000-07-07	rahul@gmail.com
4	check-rollback	2000-07-07	rahul@gmail.com


```
ROLLBACK;
```

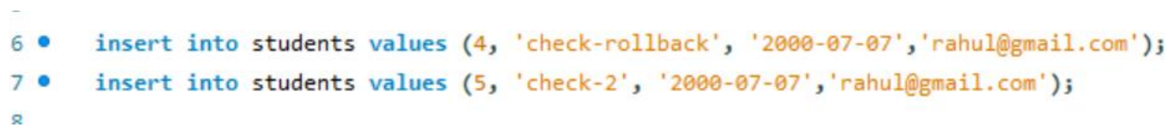


studentID	studentName	DateOfBirth	emailaddress
2	Arun	2004-10-03	arun@gmail.com
1	raja	2000-07-07	raja@gmail.com
3	rahul	2000-07-07	rahul@gmail.com

Accidentally Ran COMMIT

Double-check that you didn't run a COMMIT; before your ROLLBACK; Once committed, rollback cannot undo it.

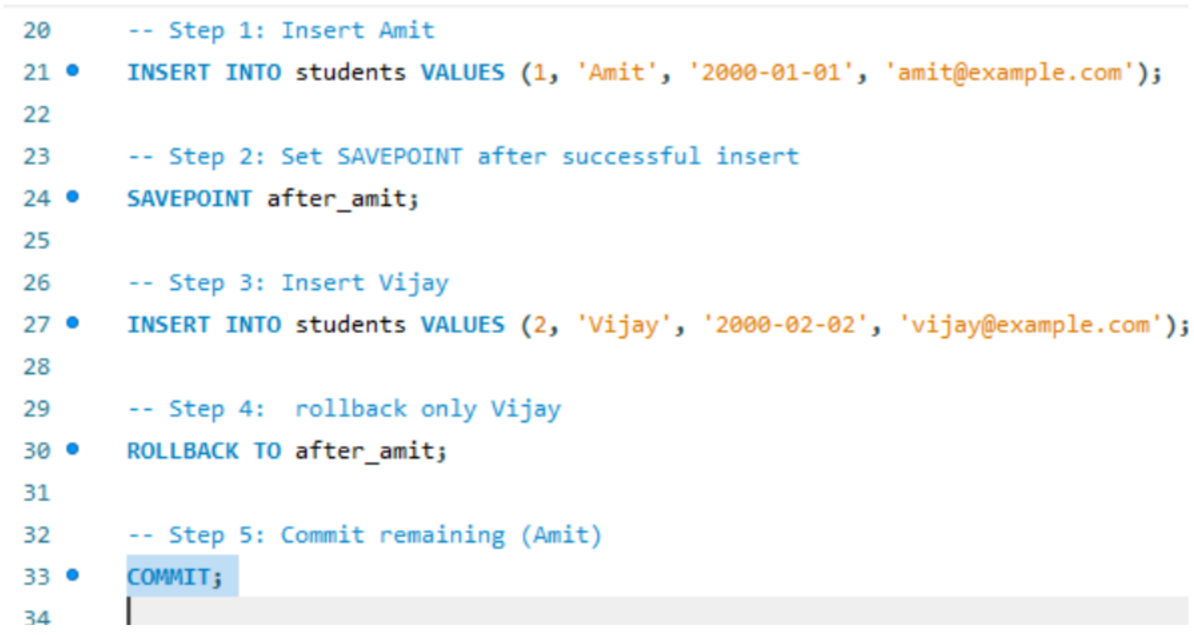
Savepoint

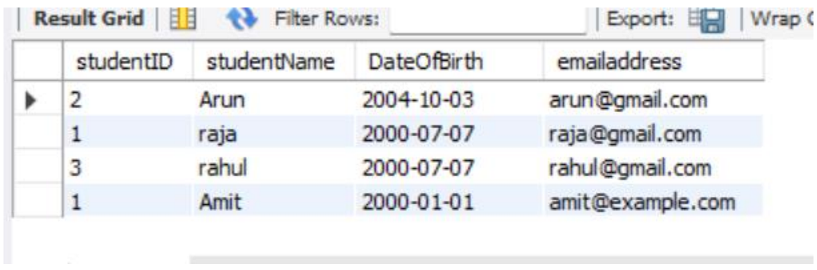

```
insert into students values (4, 'check-rollback', '2000-07-07','rahul@gmail.com');
insert into students values (5, 'check-2', '2000-07-07','rahul@gmail.com');
```

A SAVEPOINT lets you **set a named point within a transaction** that you can roll back to **without cancelling the whole transaction**.

Basic Syntax → SAVEPOINT savepoint_name;

Then you can → ROLLBACK TO savepoint_name;


```
-- Step 1: Insert Amit
INSERT INTO students VALUES (1, 'Amit', '2000-01-01', 'amit@example.com');

-- Step 2: Set SAVEPOINT after successful insert
SAVEPOINT after_amit;

-- Step 3: Insert Vijay
INSERT INTO students VALUES (2, 'Vijay', '2000-02-02', 'vijay@example.com');

-- Step 4:  rollback only Vijay
ROLLBACK TO after_amit;

-- Step 5: Commit remaining (Amit)
COMMIT;
```



Result Grid | Filter Rows: | Export: | Wrap

	studentID	studentName	DateOfBirth	emailaddress
▶	2	Arun	2004-10-03	arun@gmail.com
	1	raja	2000-07-07	raja@gmail.com
	3	rahul	2000-07-07	rahul@gmail.com
	1	Amit	2000-01-01	amit@example.com

What Happens If You Set SAVEPOINT First?

SAVEPOINT sp0;
-- Insert Amit
-- Insert Vijay
ROLLBACK TO sp0;

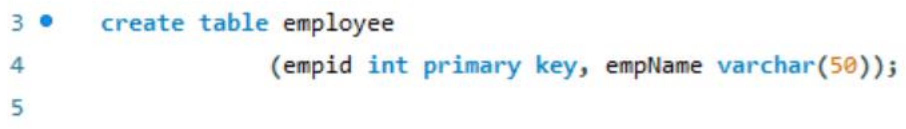
This would **undo both** Amit and Vijay — because the SAVEPOINT was created **before any insert**.

Constraints

Constraints are rules applied to table columns to enforce data integrity and ensure accuracy and consistency in a database.

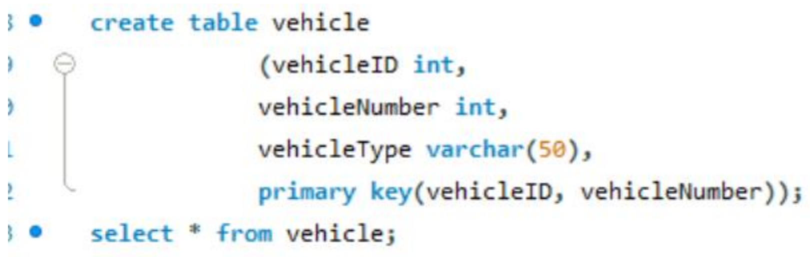
1. Primary key

Ensures that each value in the column is **unique (no duplicate) and not null.**



```
create table employee
            (empid int primary key, empName varchar(50));
```

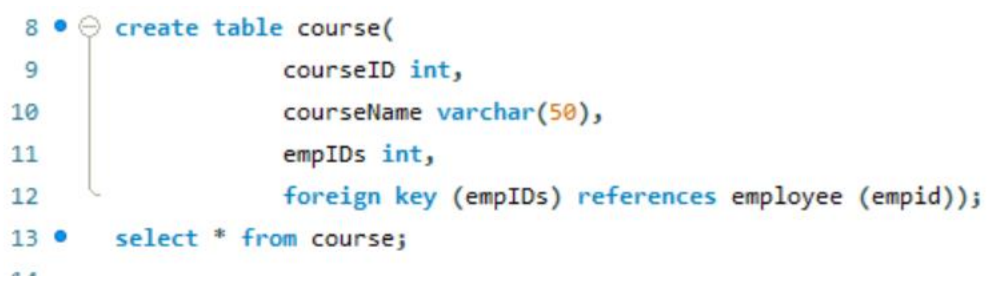
- A table can have only one primary key.
- Can be a composite key (multiple columns – primary key with 2 or more columns).



```
create table vehicle
            (vehicleID int,
            vehicleNumber int,
            vehicleType varchar(50),
            primary key(vehicleID, vehicleNumber));
select * from vehicle;
```

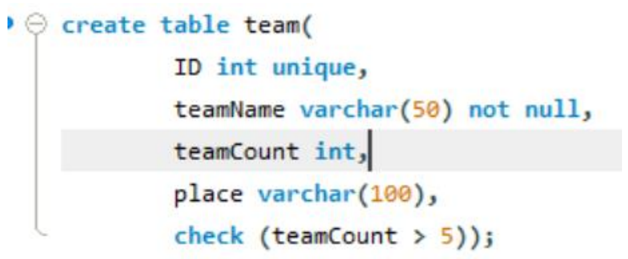
2. Foreign Key

- Establishes a relationship between two tables.
- The foreign key in one table refers to the **primary key** in another.



```
create table course(
            courseID int,
            courseName varchar(50),
            empIDs int,
            foreign key (empIDs) references employee (empid));
select * from course;
```

3. UNIQUE



```
create table team(
        ID int unique,
        teamName varchar(50) not null,
        teamCount int,
        place varchar(100),
        check (teamCount > 5));
```

Ensures all values in a column or combination of columns are **distinct**.

```
CREATE TABLE users (
    userID INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

- Like PRIMARY KEY, but allows **one NULL** value (in most databases).
- A table can have **multiple UNIQUE** constraints.

4. CHECK

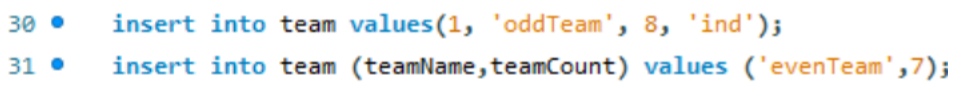
Ensures that column values satisfy a specific **logical condition**.

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    age INT,
    salary INT,
     CHECK (age >= 18 AND salary >= 10000));
```

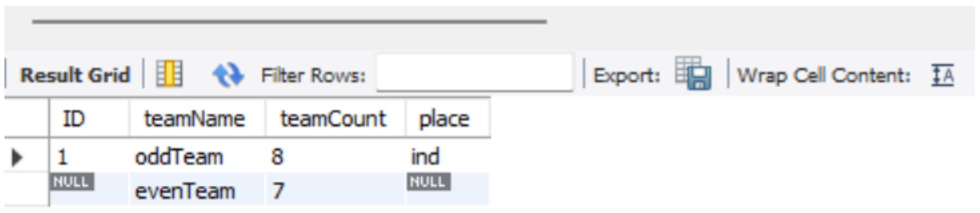
5. NOT NULL

Ensures a column **must have a value** (cannot be NULL).

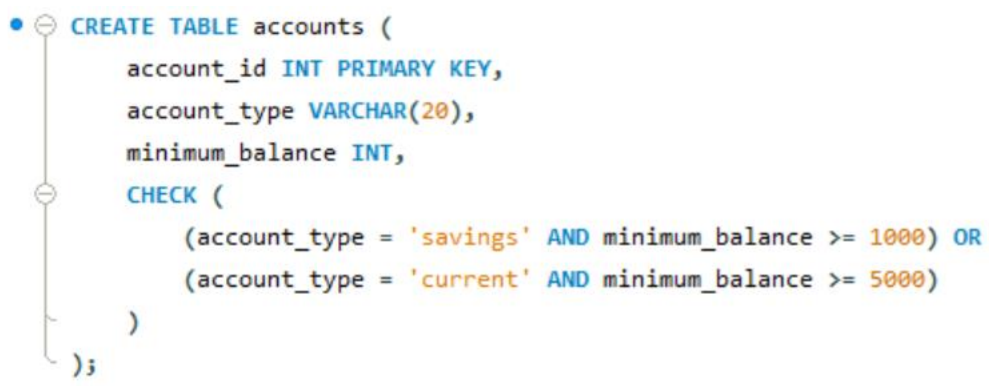
```
CREATE TABLE products (
    productID INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```



```
insert into team values(1, 'oddTeam', 8, 'ind');
insert into team (teamName,teamCount) values ('evenTeam',7);
```



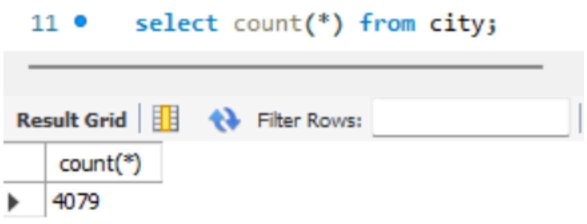
ID	teamName	teamCount	place
1	oddTeam	8	ind
NULL	evenTeam	7	NULL


```
CREATE TABLE accounts (
    account_id INT PRIMARY KEY,
    account_type VARCHAR(20),
    minimum_balance INT,
    CHECK (
        (account_type = 'savings' AND minimum_balance >= 1000) OR
        (account_type = 'current' AND minimum_balance >= 5000)
    )
);
```

Aggregation function

Sum, min, max, count, avg

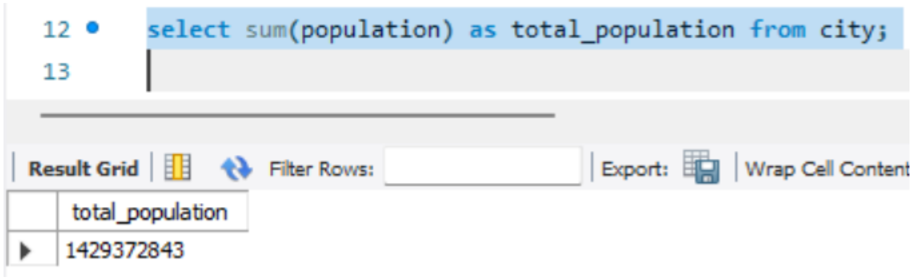
Count – it is used to count the no. of data.


```
select count(*) from city;
```

Result Grid | Filter Rows:

```
count(*)
4079
```

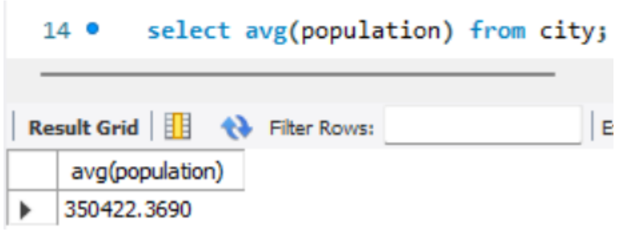
Sum – it is used to total number of value or addition of values.


```
select sum(population) as total_population from city;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

```
total_population
1429372843
```

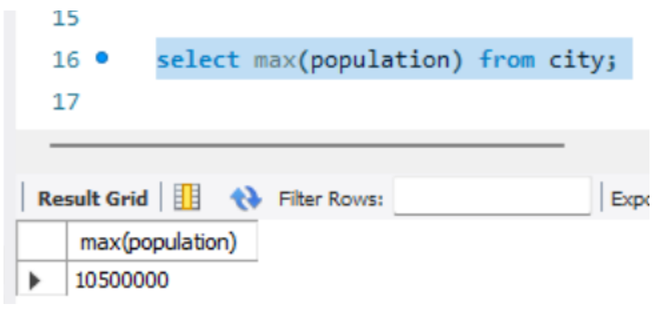
Avg


```
select avg(population) from city;
```

Result Grid | Filter Rows:

```
avg(population)
350422.3690
```

Max

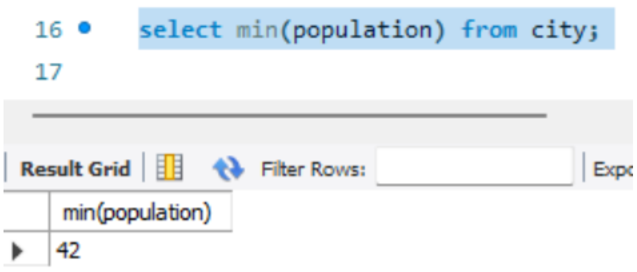


```
select max(population) from city;
```

Result Grid | Filter Rows: | Expo

```
max(population)
10500000
```

Min



```
select min(population) from city;
```

Result Grid | Filter Rows: | Expo

```
min(population)
42
```

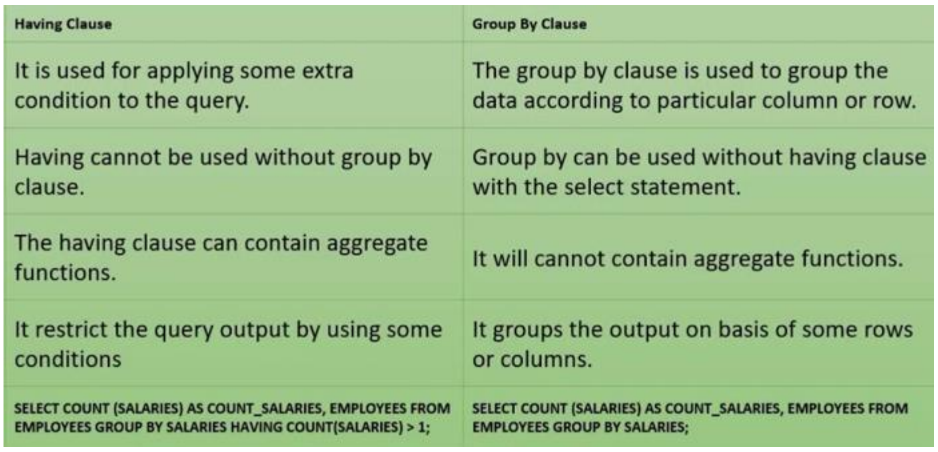
Group By

Grouping in sql is a statement groups rows that have the same values in specified columns.

Group by statement is often used with aggregate function.

Having

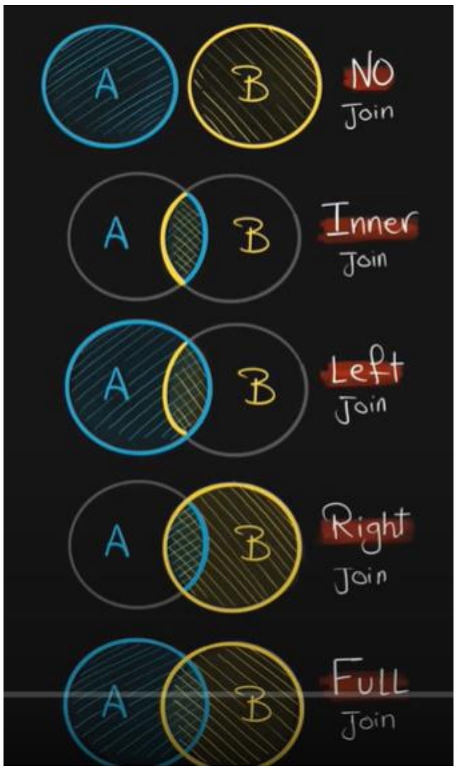
HAVING is used to **filter the results after grouping**, especially when we're applying a condition on aggregated values.



Having Clause	Group By Clause
It is used for applying some extra condition to the query.	The group by clause is used to group the data according to particular column or row.
Having cannot be used without group by clause.	Group by can be used without having clause with the select statement.
The having clause can contain aggregate functions.	It will cannot contain aggregate functions.
It restrict the query output by using some conditions	It groups the output on basis of some rows or columns.
SELECT COUNT (SALARIES) AS COUNT_SALARIES, EMPLOYEES FROM EMPLOYEES GROUP BY SALARIES HAVING COUNT(SALARIES) > 1;	SELECT COUNT (SALARIES) AS COUNT_SALARIES, EMPLOYEES FROM EMPLOYEES GROUP BY SALARIES;

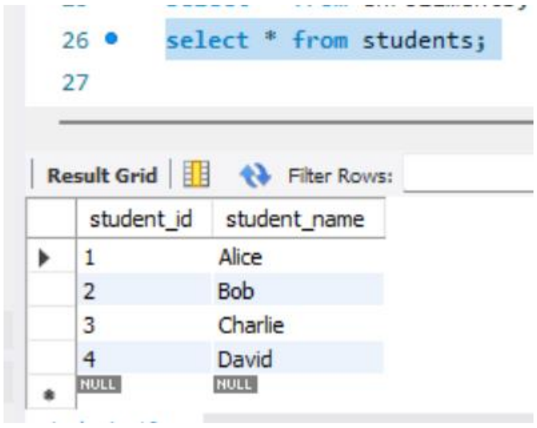
Joins

SQL joins are used to combine rows from two or more tables based on a related column between them.



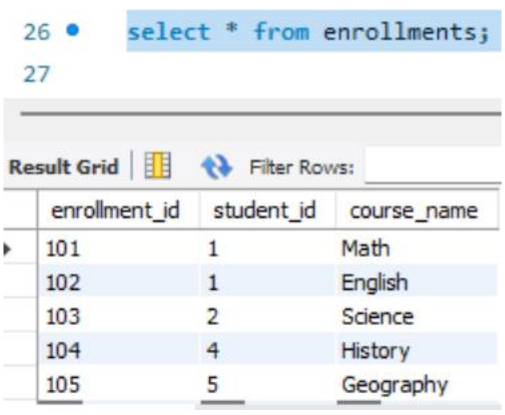
Inner joins

Returns only matching rows from both tables.



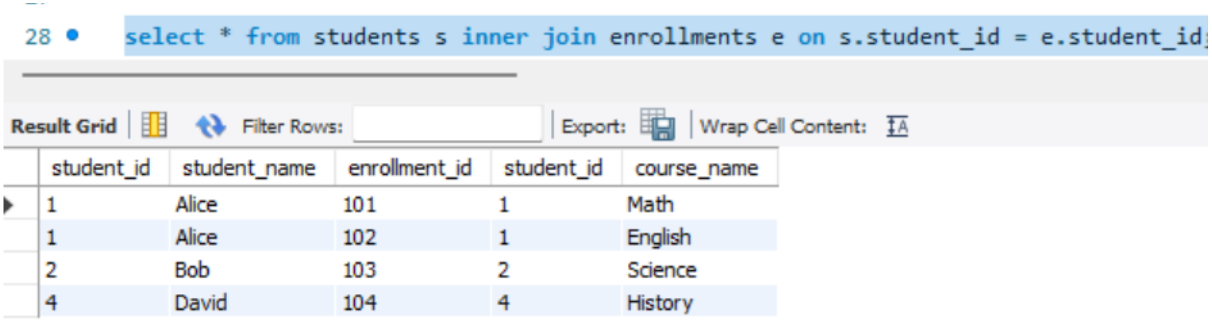
```
select * from students;
```

student_id	student_name
1	Alice
2	Bob
3	Charlie
4	David
NULL	NULL



```
select * from enrollments;
```

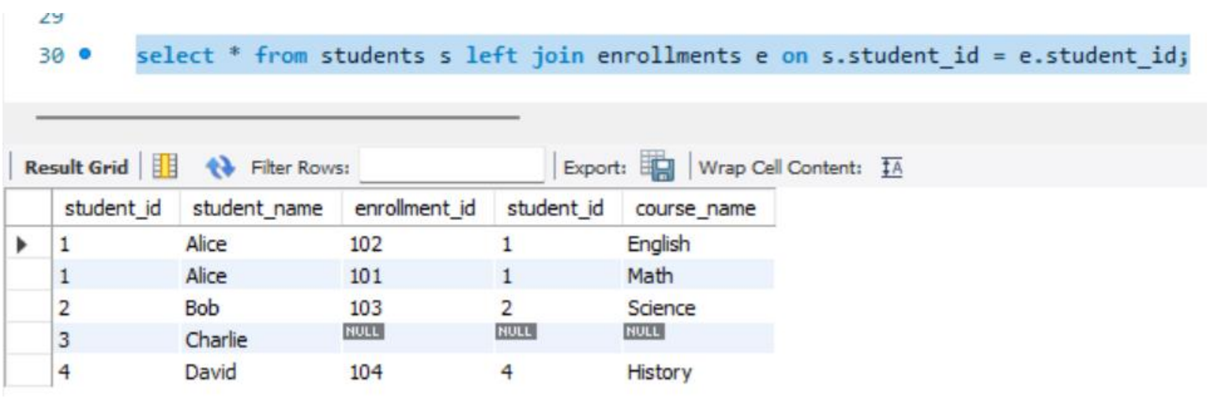
enrollment_id	student_id	course_name
101	1	Math
102	1	English
103	2	Science
104	4	History
105	5	Geography



```
select * from students s inner join enrollments e on s.student_id = e.student_id;
```

student_id	student_name	enrollment_id	student_id	course_name
1	Alice	101	1	Math
1	Alice	102	1	English
2	Bob	103	2	Science
4	David	104	4	History

Left join



```sql
select * from students s left join enrollments e on s.student_id = e.student_id;
```

student_id	student_name	enrollment_id	student_id	course_name
1	Alice	102	1	English
1	Alice	101	1	Math
2	Bob	103	2	Science
3	Charlie	NULL	NULL	NULL
4	David	104	4	History

RIGHT JOIN returns:

- All rows from the right table (e.g., Departments)
- Matching rows from the left table (Employees)
- If there's no match, columns from the left table will be NULL

```sql
-- list employee who works in department "D"

select e.emp_name, d.dept_id ,  d.dep_name from employees e inner join departments d
on e.dep_id = d.dept_id
where d.dep_name = "D";
```

```sql
-- show the employee not belong to any department
select e.emp_name, d.dep_name from employees e left join departments d on e.dep_id = d.dept_id
where d.dep_name is null;
```

```sql
-- Display each department and the number of employees in it
select d.dep_name, count(e.emp_name) as emp_count from employees e right join departments d
on e.dep_id = d.dept_id group by d.dep_name having emp_count >= 1;
```

View
A view is a virtual table in SQL based on the result of a select query.

- It does not store data physically.
- When you query a view, the underlying SELECT statement is executed.
- It can simplify complex queries and enhance security.

```sql
-- Create a View
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
```

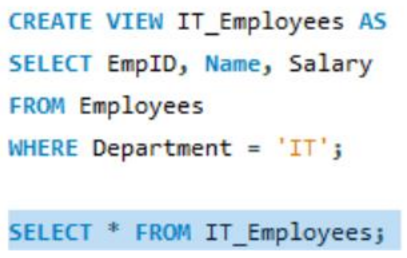
WHERE condition;

-- Query the View
SELECT * FROM view_name;

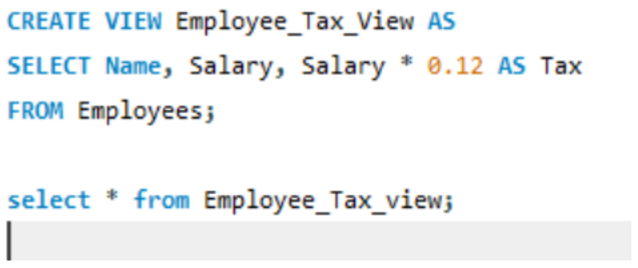
-- Update the View (if allowed)
UPDATE view_name SET column = value WHERE condition;

-- Drop the View
DROP VIEW view_name;

Improve readability and maintainability.
Create reusable query logic.



```
CREATE VIEW IT_Employees AS
SELECT EmpID, Name, Salary
FROM Employees
WHERE Department = 'IT';

SELECT * FROM IT_Employees;
```



```
CREATE VIEW Employee_Tax_View AS
SELECT Name, Salary, Salary * 0.12 AS Tax
FROM Employees;

select * from Employee_Tax_view;
```

Normalization

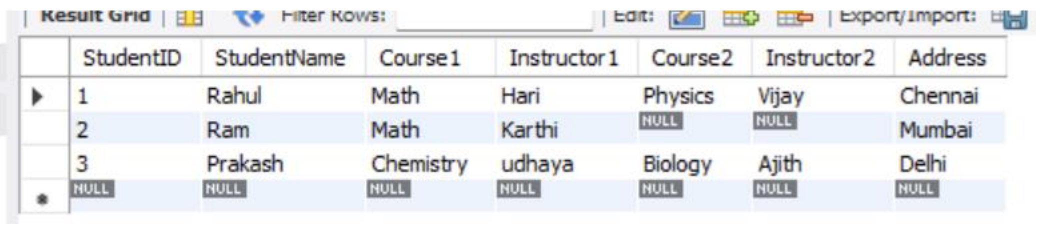
Normalization is a process of organizing data in a database to reduce redundancy (duplicate), improve data integrity (data accuracy & consistency).

Is the process also involved decomposing large, complex tables (denormalization) into smaller (normalization), related ones and defining relationships between them using foreign keys.

1. Normalization has one of the main tasks is to organize the data systematically.
2. It is mainly in use for reducing data redundancy.
3. It breaks big data tables into small sets of data and stores them into a table.
4. It eliminates the duplication of data to improve the data integrity.
5. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.

Why we need normalization?

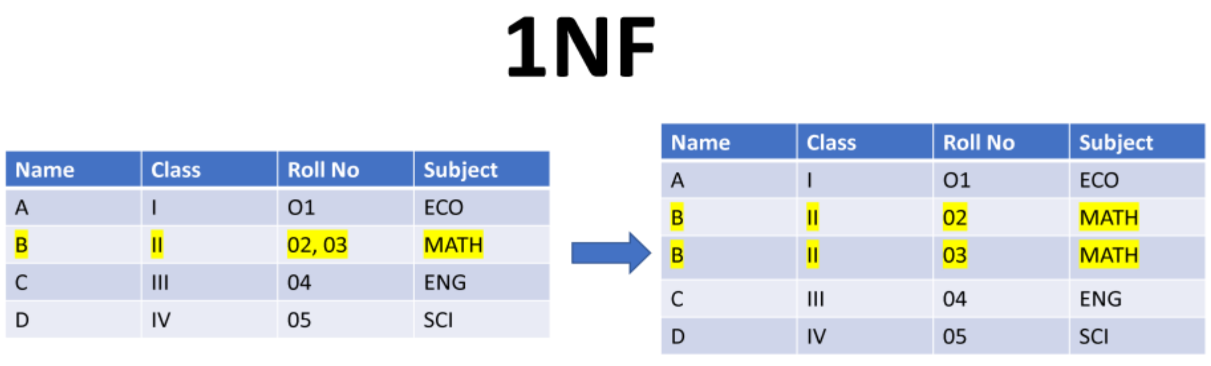
To convert the big relations data into the small set of data relations and to remove the anomalies i.e., insertion, updation and deletion that makes data difficult to read & analyze. To clean-up the data and manage it with fewer attributes to increase the searching efficiency of the data into the database.



StudentID	StudentName	Course1	Instructor1	Course2	Instructor2	Address
1	Rahul	Math	Hari	Physics	Vijay	Chennai
2	Ram	Math	Karthi	NULL	NULL	Mumbai
3	Prakash	Chemistry	udhaya	Biology	Ajith	Delhi
NULL	NULL	NULL	NULL	NULL	NULL	NULL

1st Normal Form (1NF)

This Normal Form is known for dealing with atomicity i.e., values in the table should not be further divided. In other words, a single row cannot hold the multiple values and must contain the unique value in the table.



1NF

Name	Class	Roll No	Subject
A	I	01	ECO
B	II	02, 03	MATH
C	III	04	ENG
D	IV	05	SCI

Not follows 1NF

Name	Class	Roll No	Subject
A	I	01	ECO
B	II	02	MATH
B	II	03	MATH
C	III	04	ENG
D	IV	05	SCI

Follows 1NF

2nd Normal Form (2NF)

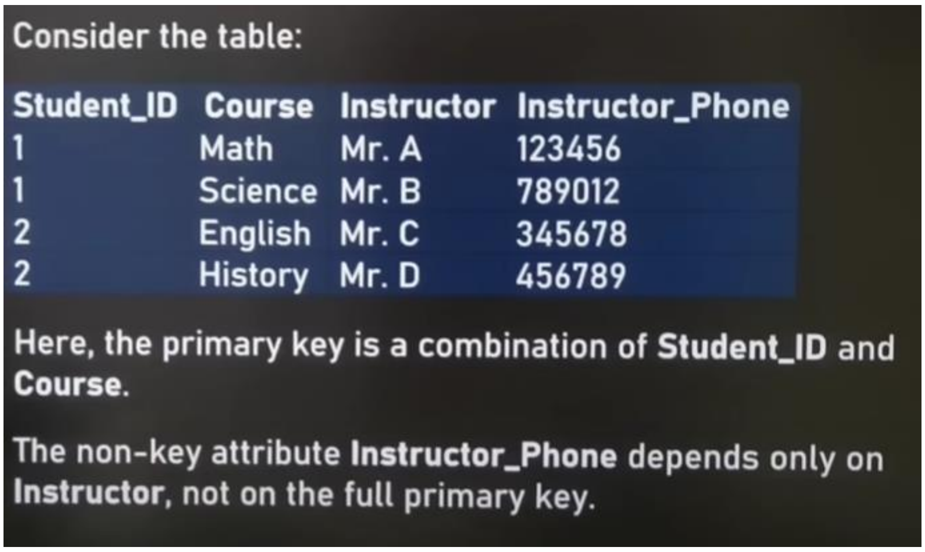
The primary condition of 2nd NF is that it should contain the 1st NF and in addition, it should not contain partial dependency i.e., non-prime attributes should not be dependent on any candidate key's proper subset.

All non-key attributes must be fully dependent on candidate key. (i.e. if a non-key column is partially dependent on candidate key, then split them into separate tables.)

Every table should have primary key and relationship between the tables should be formed using foreign key.

Candidate key- set of columns which uniquely identify a record.

A table can have multiple candidate key because there can be multiple set of columns which uniquely identify a row in a table.

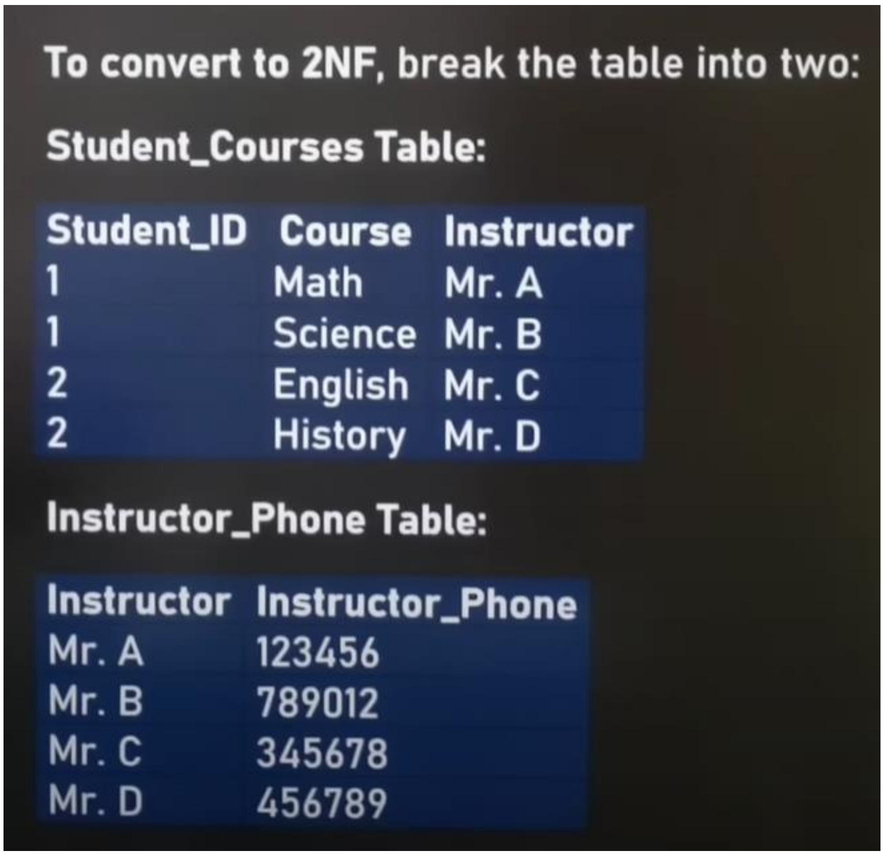


Consider the table:

Student_ID	Course	Instructor	Instructor_Phone
1	Math	Mr. A	123456
1	Science	Mr. B	789012
2	English	Mr. C	345678
2	History	Mr. D	456789

Here, the primary key is a combination of **Student_ID** and **Course**.

The non-key attribute **Instructor_Phone** depends only on **Instructor**, not on the full primary key.



To convert to 2NF, break the table into two:

Student_Courses Table:

Student_ID	Course	Instructor
1	Math	Mr. A
1	Science	Mr. B
2	English	Mr. C
2	History	Mr. D

Instructor_Phone Table:

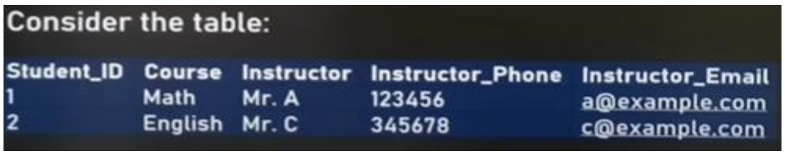
Instructor	Instructor_Phone
Mr. A	123456
Mr. B	789012
Mr. C	345678
Mr. D	456789

3NF (Third Normal form)

Rule

It is in 2NF.

There are no transitive dependencies (i.e. non-key columns should not depend on other non-key columns).

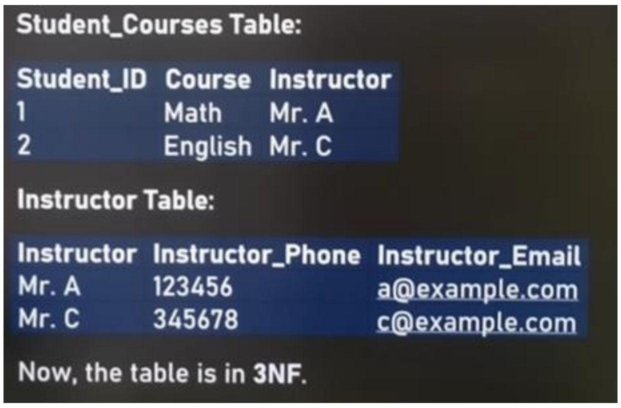


Consider the table:

Student_ID	Course	Instructor	Instructor_Phone	Instructor_Email
1	Math	Mr. A	123456	a@example.com
2	English	Mr. C	345678	c@example.com

Here, Instructor_phone and Instructor_email depend on Instructor, which is non-key column.

To convert to 3NF, remove the transitive dependency by splitting the data into two table:



Student_Courses Table:

Student_ID	Course	Instructor
1	Math	Mr. A
2	English	Mr. C

Instructor Table:

Instructor	Instructor_Phone	Instructor_Email
Mr. A	123456	a@example.com
Mr. C	345678	c@example.com

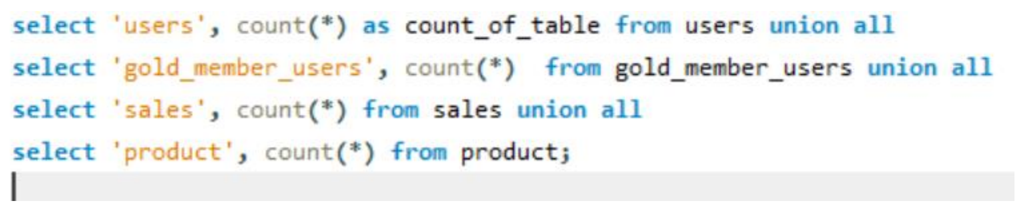
Now, the table is in 3NF.

Union All

The UNION ALL operator is used to combine the results of two or more SELECT statements into a single result set. Unlike the UNION operator, which removes duplicate rows, UNION ALL includes all rows, even if they are duplicates.

Syntax

```
SELECT column1, column2, ...
FROM table1
WHERE condition
UNION ALL
SELECT column1, column2, ...
FROM table2
WHERE condition;
```

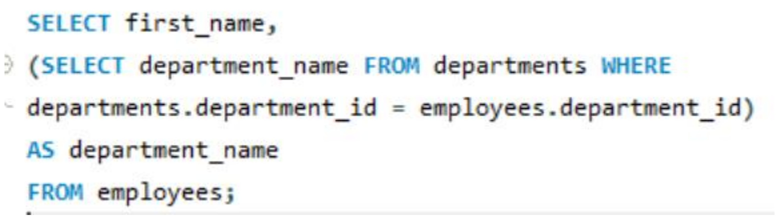


```
select 'users', count(*) as count_of_table from users union all
select 'gold_member_users', count(*)  from gold_member_users union all
select 'sales', count(*) from sales union all
select 'product', count(*) from product;
```

Subquery

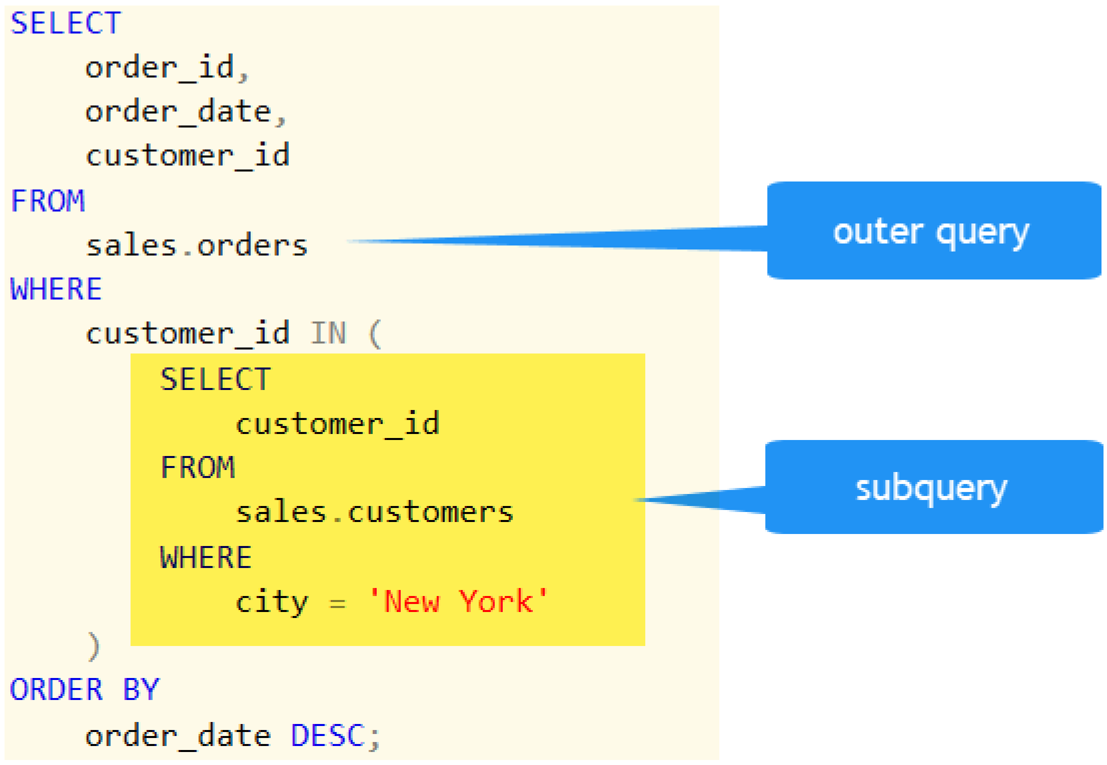
A subquery is an SQL query nested inside another query. The query that contains a subquery is known as an outer query.

- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.



```
SELECT first_name,
(SELECT department_name FROM departments WHERE
departments.department_id = employees.department_id)
AS department_name
FROM employees;
```

```sql
SELECT first_name
FROM employees
WHERE department_id IN (SELECT department_id FROM departments WHERE location_id>1500);
```



```sql
SELECT
    order_id,
    order_date,
    customer_id
FROM
    sales.orders
WHERE
    customer_id IN (
        SELECT
            customer_id
        FROM
            sales.customers
        WHERE
            city = 'New York'
    )
ORDER BY
    order_date DESC;
```