**DBMS**

**What is DBMS?**

|  |
| --- |
| 1. It is a software which is used to manage the database   **Example:** mysql, oracle, SqlServer, Postgress SQL etc. |

**What is a database?**

|  |
| --- |
| 1. Organized Collection of interrelated data which represents real world entities   Ex: social media database  Ex: College data base: student data, employee data, dept data, course details, etc..  Ex: bank data base |

**What is data?**

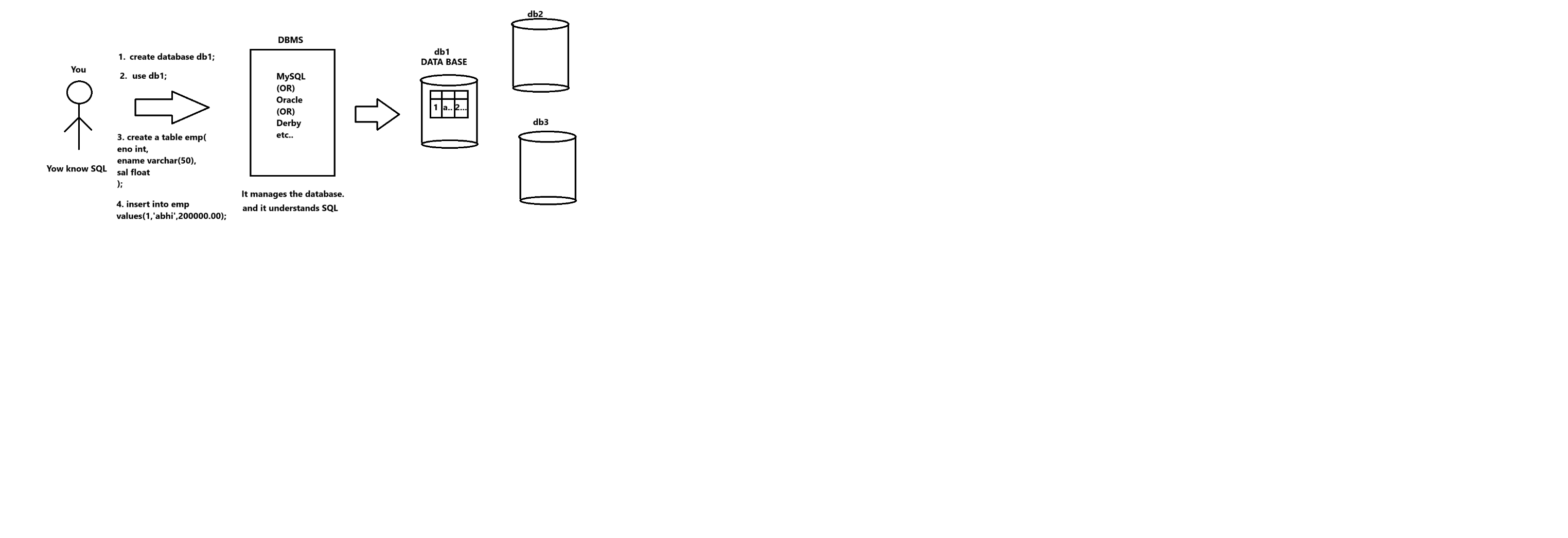
|  |
| --- |
| 1. Data is nothing but raw facts   **Example:** name, email, id, password, profile pic, description, posts, comments, likes, of  social media account. |

**What is an information?**

|  |
| --- |
| Processed data  Ex: market analysis, whether forecast, over all pass percentage, student with reg no 11111 got state first. |

**What is SQL?**

|  |
| --- |
| It is a language which is used to interact any database management system. |



**SQL**

**What is SQL(Structured Query Language)?**

|  |
| --- |
| 1. It is a language which is used to interact with DMBS. 2. It is very easy to learn |

**SQL commands**

|  |
| --- |
| **All SQL commands are divided into 5 categories**   1. DDL (Data Definition Language) 2. DML (Data Manipulation Language) 3. DCL (Data Control Language) 4. TCL (Transaction Control Language) 5. DQL (Data Query Language) |

**DDL**

|  |
| --- |
| We use DDL commands to create, alter, the data base schema. The following are the commands available in DDL.   1. Create (used to create database, tables, indexes, triggers, views etc…) 2. Alter (used to add column, rename column rename table, to modify column etc.,) 3. Drop (used to drop daabase or table or column etc…) 4. Rename (it is used to rename table) 5. Truncate (to delete the whole data from table permanently) |

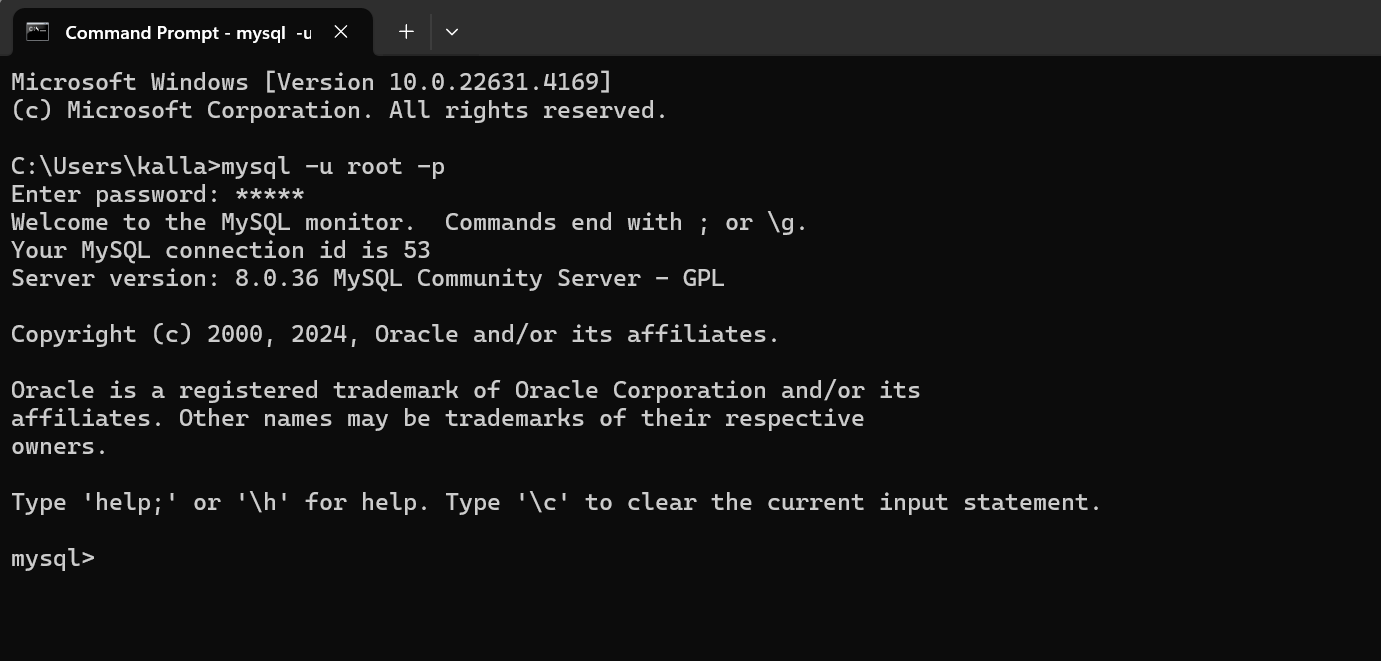
**What is database schema?**

|  |
| --- |
| A database schema is the blueprint of a database.  Database schema design 101 for relational databases ... |

**What is root user?**

|  |
| --- |
| Whenever you install MYSQL In your computer. By default it comes along with a user called root user. |

**Esatablish the Connection to the root user of the mysql using command prompt.**

****

**Command to see the list of databases available**

|  |
| --- |
|  |

**Command to create a database**

|  |
| --- |
|  |

**How to drop the database from the user?**

|  |
| --- |
|  |

**How to clear the mysql prompt screen?**

|  |
| --- |
|  |

**Command to connect to a particular database**

|  |
| --- |
|  |

**Command to see the list of tables available in a database?**

|  |
| --- |
|  |

**Command to create a table**

|  |
| --- |
| Creating a student table  create table student(  sno int,  sname varchar(20),  branch varchar(10),  year int,  sem int  ); |

**Command to describe the table**

|  |
| --- |
|  |

**Command to add new column to a table**

|  |
| --- |
| We need to use alter command to add a new column to a table.    Adding multiple columns at a time |

**Command to modify the column data type**

|  |
| --- |
|  |

**Command to drop a column**

|  |
| --- |
|  |

**Command to rename a table**

|  |
| --- |
| **Another way** |

**Command to rename a column**

|  |
| --- |
|  |

**What is the use of truncate command?**

|  |
| --- |
| 1. Truncate is a command which deletes the whole data permanently from the table. We can’t get back the data once truncated in mysql. 2. By using truncate command, we can’t delete some rows of data from table. |

**DML(Data Manipulation Language) commands**

|  |
| --- |
| These commands are used to insert, update and delete the data from a table.   1. Insert (used to insert a row or rows) 2. Update 3. Delete   Let us create a table |

**Which date format mysql supports?**

|  |
| --- |
| When you are inserting date or string it must be represented by using pair of single quotations.  Ex: ‘madhu’  Ex: ‘2023-12-26’ ‘yyyy-mm-dd’ |

**What is the mysql format?**

|  |
| --- |
| YYYY-MM-DD |

**What is the oracle date format?**

|  |
| --- |
| DD-MMM-YYYY  **Ex: ’26-Dec-2023’** |

**Command to insert a row into emp table**

|  |
| --- |
|  |

**DQL commands**

|  |
| --- |
| This command is used to get the data from a table   1. Select |

**How to get the data from a table**

|  |
| --- |
|  |

**Another Command to insert a row into emp table**

|  |
| --- |
|  |

**Command to insert the data into a single column of a row**

|  |
| --- |
|  |

**Why we specify the column name while inserting?**

|  |
| --- |
| 1. If you want to insert the data in few columns of a table, then we should specify the column names. 2. If you are not following order in which the columns present in the table, at the time of inserting the data then you have to specify the column names.   Example: |

**inserting a row into a table without specifying column names**

|  |
| --- |
|  |

**Inserting multiple rows at a time in MySQL?**

|  |
| --- |
| If you are inserting the data by following the order of columns then no need to specify the column names while insertion.    **If you try to insert the data without following column order like below then you will get Data Truncated error**    **Because I am following the correct column order it is getting inserted without any error** |

**Command to see the data existed in some columns of a table**

|  |
| --- |
| **Another example** |

**What is the default value of every column**

|  |
| --- |
| 1. By default, it is null but we can change the default value of a column at the time of creating a table or by altering the table |

**In SQL How we can represent a string?**

|  |
| --- |
| By using pair of single quotations |

**In SQL How we can represent a date?**

|  |
| --- |
| By using pair of single quotations |

**Command Insert the data without specifying the column names**

|  |
| --- |
| 1. At the time of inserting data into a table without specifying column names then the rule is you have to insert the data into all the columns and must follow the table columns order. |

**Insert command to insert the data by using set keyword**

|  |
| --- |
|  |

**Inserting multiple rows at a time**

|  |
| --- |
|  |

**Command to delete all the data from table;**

|  |
| --- |
|  |

**Update command**

|  |
| --- |
| **Updating salary of a specified employee**    **Another example** |

**How to delete the a particular row from a table**

|  |
| --- |
|  |

**What is commit?**

|  |
| --- |
| 1. Here commit means save |

**What is the default commit behavior of mysql?**

|  |
| --- |
| 1. By default mysql commits after every command |

**Why we use set autocommit=False;**

|  |
| --- |
| By using this command I am requesting mysql that don’t commit after execution of dml commands. |

**Can we apply commit and rollback commands after ddl commands?**

|  |
| --- |
| 1. DDL commands are auto commit commands so no need to use commit command explicitly 2. Roll back will not work for ddl commands |

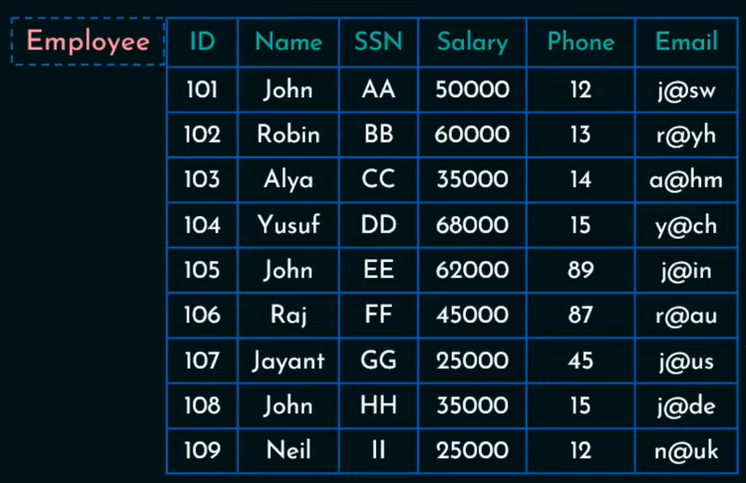
**Can I use commit and roll back with DML commands**

|  |
| --- |
| Yes, but in mysql you need to set autocommit=false, then only we can use commit and rollback explicitly. Other wise dml commands also works like default commit commands. |

**Keys**

Why do we need keys in RDBMS?

|  |
| --- |
| We need a key to identify a particular tuple or record or row in a table |



**Super keys(All possible keys)**

1. Id
2. Email
3. ssn
4. Id, email, ssn, phone,
5. Id,name as a key
6. Id,salary as a key
7. Id,phone as a key
8. Id,email,
9. Id,email,name
10. Email,name
11. Phone,name
12. Ssn,name
13. Ssn,salary
14. Ssn,email
15. Ssn,eno
16. Etc…..

**Candidate Keys(minimal superkey)**

1. Id
2. Email
3. Ssn
4. Phone,name

**Primary key(most appropriate key which is taken from candidate key):**

1. id

**Alternate keys (Except primary key the remaining candidate keys)**

1. **phone,email and ssn**

**Composite key**

|  |
| --- |
| If 2 or more columns are taken as a primary key in a table then it is called as composite key |

In RDBMS we come across many Key terms those are

|  |
| --- |
| 1. Super key (all possible keys existed in a table) 2. Candidate key (minimal super key) 3. Primary key (most appropriate key which is taken from candidate key) 4. Alternate key (all the keys except primary key of candidate key set) 5. Unique 6. Composite (if a primary is a combination of 2 or more) 7. Foreign |

**Super key**

|  |
| --- |
| 1. A super key is an attribute or collection of attributes so that we can uniquely identify a tuple.    1. {id} can be super key    2. {SSN} can be super key    3. {id,name} can be super key    4. {id,email} can be super key    5. {Name,ssn,phone} can be super key    6. {Name,Email} can be super key    7. {id,ssn,phone} can be super key    8. {id,phone} can be super key    9. Etc.. 2. Super key is a super set from which we can derive other keys |

**Candidate key (minimal super key)**

|  |
| --- |
| We can derive candidate keys or key from the super key or super key set. But remember to pick the candidate keys don’t use the same column name repeatedly. So now I got these candidate keys from super key set.   1. {id} 2. {ssn} 3. {email} |

What is primary key?

|  |
| --- |
| 1. Most appropriate candidate key is used as primary key 2. A primary key is the combination of unique + not null   Q) Among candidate keys {id}, {ssn}, {Email} which one can be the most appropriate key  Ans) id  Who decides which key is primary key?  Ans) DBA  **Note: there will be only one primary in a table it is a rule you have to follow it.** |

**What is alternate key?**

|  |
| --- |
| **Among all the candidate keys except primary key the remaining keys are called as alternate keys**  **Ex: Candidate keys( {ID},{SSN},{Email} )**  Ex: Alternate keys({SSN},{Email} )  Ex: Primary key( {Id}) |

**Unique Key**

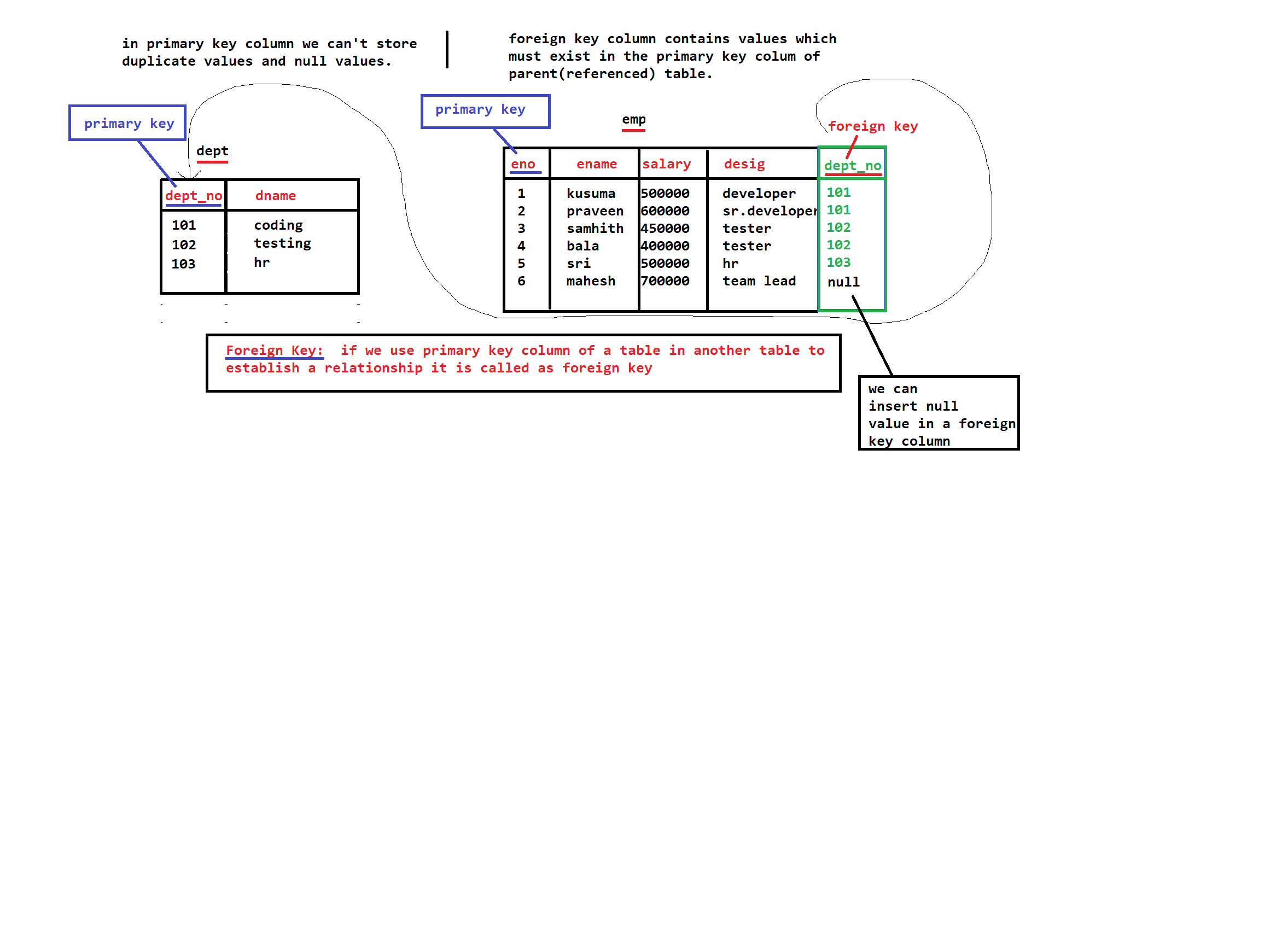
|  |
| --- |
| 1. **In Unique key column either we can store different values or null** 2. **If a column(attribute) is declared as unique key then in that column we can store only unique(different) values and we can store any no.of null values.**   Ex: Phone, Email can be declared as unique key  Ex: {Name, Phone} combination can be also used as unique key |

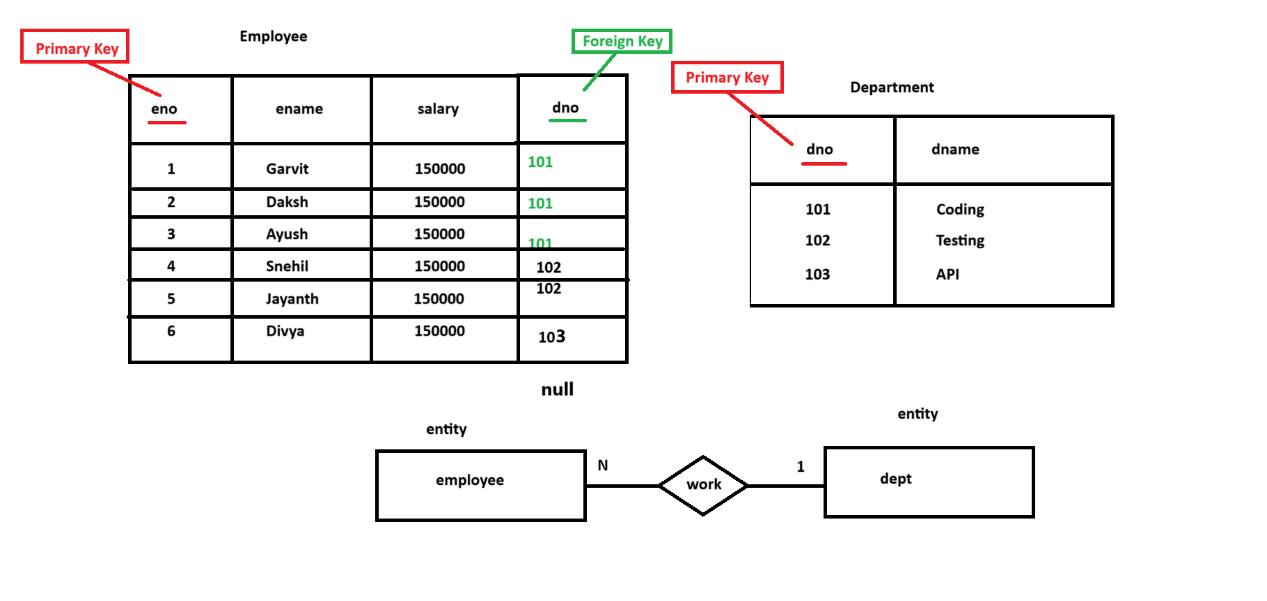
**How many null values we can store in a unique key column?**

|  |
| --- |
| **Many** |

**Foreign Key**

|  |
| --- |
| 1. If we use primary key column of a table in another table to establish the relationship between them, then it is called as foreign key. 2. A FOREIGN KEY is a field (or collection of fields) of one table, refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) of another table. 3. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table. 4. Foreign key column must contain only the values which are present in the referenced column or null. 5. Foreign key column may contain duplicates |

****



**Create Emp and dept table and establish the relationship between them**

|  |  |  |
| --- | --- | --- |
| **Creating dept table(parent table)**  create table dept(  dept\_no int,  dname varchar(20)unique,  location varchar(50)not null,  primary key(dept\_no)  );  **Creating child table**  create table emp(  eno int,  ename varchar(50),  salary float,  desig varchar(30),  dept\_no int,  primary key(eno),  foreign key(dept\_no) references dept(dept\_no)  );  **Insert the data into dept table**  insert into dept values(101,'coding','1stFloor'),(102,'testing','2nd Floor'),(103,'API','3rd Floor'),(104,'HR','1st Floor');    **Inserting data into emp table**  insert into emp values(1,'PoojaSri',100000,'developer',101),  (2,'Kishanth',200000,'developer',101),  (3,'Krithik',100000,'tester',102),  (4,'Prasath',150000,'tester',102),  (5,'Akash',200000,'developer',103),  (6,'CharuRamesh',100000,null,null);  DQL (display the data existed in tables)    **Sub Query:**  **DML command**  **Can I delete the dept table data now?**   |  | | --- | |  |   **How we can delete the child table data whenever we delete the corresponding data existed in parent table?**   |  | | --- | | **create table dept(**  **dno int,**  **dname varchar(20),**  **primary key(dno)**  **);**  **insert into dept values(101,'coding'),(102,'testing'),(103,'API'),(104,'HR');**  **create table emp(**  **eno int,**  **ename varchar(20),**  **sal float,**  **desig varchar(20),**  **doj date,**  **dno int,**  **primary key(eno),**  **foreign key(dno) references dept(dno) on delete cascade**  **);**  **Now I am able to delete the parent table data like below**    **Data will be deleted from dept, emp table** | |

**Sub Query**

|  |
| --- |
| If we write a query within another query it is called as sub query. |

**Write a command to display all the employees working in department ‘coding’**

|  |
| --- |
|  |

**Composite key**

|  |
| --- |
| In a table sometimes we need to create a primary key by using the combination of 2 or more columns. It is called as composite key. |

|  |
| --- |
| If we use eno,ename combinedly as a primary key we can store the values like below  **ENO ENAME**  1 null Wrong  null madhu wrong  1 shekar correct |

**In a composite key (primary key contains 2 or more columns) can we store null value in a combination?**

|  |
| --- |
| No, we can’t |

**Example on how to establish a relation ship between two table which are participating in many to many relation ship and also discussed about composite key**



**Creating tables customer, orders, products, ordered\_products tables to demonstrate how to establish many-> many relation ship along with composite key.**

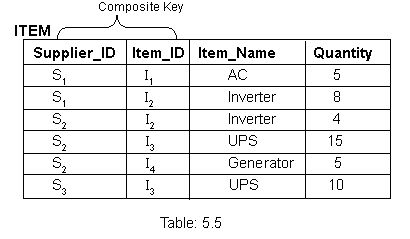
|  |
| --- |
| create table customer(  cust\_id int,  cname varchar(25),  phone varchar(10),  email varchar(25),  primary key(cust\_id)  );  insert into customer values(1,'Shreya','6767676767','shreya@gmail.com'),  (2,'Aditi','6666666666','aditi@gmail.com'),  (3,'Sachin','5656565656','sachin@gmail.com'),  (4,'Nitin','8787878787','nitin@gmail.com'),  (5,'Paramjeet','1234543212','paramjeet@gmail.com');  create table orders(  order\_id int,  amount float,  order\_date date,  cust\_id int,  primary key(order\_id),  foreign key(cust\_id) references customer(cust\_id)  );  insert into orders values(10,200000,'2023-07-09',1),  (20,100000,'2023-07-09',1),  (30,10000,'2023-08-11',2),  (40,20000,'2023-03-09',2),  (50,3000,'2023-06-09',3),  (60,5000,'2023-04-09',4),  (70,6000,'2023-07-22',5);  create table products(  prod\_id int,  pname varchar(25),  sale\_price float,  primary key(prod\_id)  );  insert into products values(101,'phone',30000),  (102,'laptop',130000),  (103,'keyboard',1000),  (104,'mouse',700),  (105,'charger',2000);  create table Ordered\_Products(  order\_id int,  prod\_id int,  quantity int,  foreign key(order\_id) references orders(order\_id),  foreign key(prod\_id) references products(prod\_id),  primary key(order\_id,prod\_id)  );  insert into ordered\_products values(10,101,1),  (10,102,1),  (10,103,1),  (20,101,1),  (20,102,1),  (30,102,1),  (30,104,1),  (30,103,1),  (40,102,1),  (50,101,1),  (60,104,1),  (70,105,1); |

**Write a command to the products details of an order called 10?**

|  |
| --- |
|  |

What is compound or composite key?

|  |
| --- |
| A composite key is the [DBMS key](https://www.javatpoint.com/dbms-keys) having two or more attributes that together can uniquely identify a tuple in a table. Such a key is also known as **Compound Key**, where each attribute creating a key is a foreign key in its own right. |



**Normalizations**

**What is normalization?**

|  |
| --- |
| It is a technique to reduce redundancy from a table. |

**Duplication can be of two types in a data base table**

1. Column level
2. Row level

Row level duplication

|  |
| --- |
| If a table has more than one row with same values, it is called as row level duplication. |

**Student Table(with duplicate rows)**

|  |  |  |
| --- | --- | --- |
| Student-No | Name | Group |
| 1 | Vinutna | CSE |
| 2 | Pranutna | ECE |
| 1 | Vinutna | CSE |
| 2 | Pranutna | ECE |

How we can remove row level duplication?

|  |
| --- |
| By using the concept of Primary key |

**Student Table(Without duplicate rows)**

|  |
| --- |
| Primary Key |
| Student-No | Name | Group |
| 1 | Vinutna | CSE |
| 2 | Pranutna | ECE |

What is a primary column?

|  |
| --- |
| It is a column where we can’t store null values and also we can’t store duplicates |

**Student Table with (duplicate columns)**

|  |
| --- |
| Primary Key |
| Student\_id | Name | Course\_id | Course\_name | Faculty\_id | Faculty\_name |
| s1 | Madhu | C1 | Java | F1 | Amit |
| s2 | Priya | C1 | Java | F1 | Amit |
| s3 | Shekar | C2 | SQL | F2 | Ayush |
| S4 | Ram | C3 | Python | F3 | Shoib |
| S5 | Krishna | C2 | SQL | F2 | Ayush |

Because of duplicate columns there is a change of facing insertion, deletion and updating anomalies.

**Anomalies**

**Insertion Anomaly**

|  |
| --- |
| if you are facing problem while inserting a new row into a table then it is called as insertion anomaly  **Ex:** Can we assign a row to the above table with only new course details, The answer is no because student\_id is primary key you should assign non null element otherwise insertion will be failed. |

|  |
| --- |
| Primary Key |
| Student\_id | Name | Course\_id | Course\_name | Faculty\_id | Faculty\_name |
| s1 | Madhu | C1 | Java | F1 | Amit |
| s2 | Priya | C1 | Java | F1 | Amit |
| s3 | Shekar | C2 | SQL | F2 | Ayush |
| S4 | Ram | C3 | Python | F3 | Shoib |
| S5 | Krishna | C2 | SQL | F2 | Ayush |
|  |  | C4 | DevOps | F4 | GiriBabu |
|  |  |  |  | F5 | Mahesh |

**Deletion Anomaly**

|  |
| --- |
| It occurs at the time of performing deletion operation on a table. In the above example if you delete student ram details then faculty and course details will be missed out right? So it is called as deletion anomaly. |

**Students table**

|  |
| --- |
| Primary Key |
| Student\_id | Name | Course\_id | Course\_name | Faculty\_id | Faculty\_name |
| s1 | Madhu | C1 | Java | F1 | Amit |
| s2 | Priya | C1 | Java | F1 | Amit |
| s3 | Shekar | C2 | SQL | F2 | Ayush |
| S4 | Ram | C3 | Python | F3 | Shoib |
| S5 | Krishna | C2 | SQL | F2 | Ayush |

Update the faculty name to ‘Amar’ who Is taking classes for s2 student

|  |
| --- |
| Update students set faculty\_name=’amar’ where student\_id=’s2’; |

Command to delete student s4 details from students table(above table)

|  |
| --- |
| delete from students where student\_id=’s4’; |

Update Anomaly

Note: One student can join in one course only

Note: One faculty teaches only one subject

Update institute set faculty\_name=’Amar’ where student\_id=s2;

**Table name: institute**

|  |
| --- |
| Primary Key |
| Student\_id | Name | Course\_id | Course\_name | Faculty\_id | Faculty\_name |
| s1 | Madhu | C1 | Java | F1 | Amit |
| s2 | Priya | C1 | Java | F1 | Amar |
| s3 | Shekar | C2 | SQL | F2 | Ayush |
| S4 | Ram | C3 | Python | F3 | Shoib |
| S5 | Krishna | C2 | SQL | F2 | Ayush |

Update institute name=’amar’ where student\_id=’s2’;

|  |
| --- |
| While updating data, if you got some error, then it is called as Updation anomaly  Example: if you change the faculty name of student\_id s2 from Amit to Amar then faculty with F1 details will become ambiguous, Because in first row F1 name is Amit but in the second row F1 name becomes Amar.  Here in the above example Updating is not done in every row where the F1 facutly details are existed. |

How we can solve the above problems?

|  |
| --- |
| Divide the table into sub tables like below |

Table-1(Student)

|  |
| --- |
| Primary Key |
| Student\_id | Name |
| s1 | Madhu |
| s2 | Priya |
| s3 | Shekar |
| S4 | Ram |
| S5 | Krishna |

**Course table**

|  |  |  |
| --- | --- | --- |
| Course\_id(PK) | Course\_name | Faculty\_id(FK) |
| C1 | Java | F4 |
| C3 | Python | F2 |
| C2 | SQL | F3 |

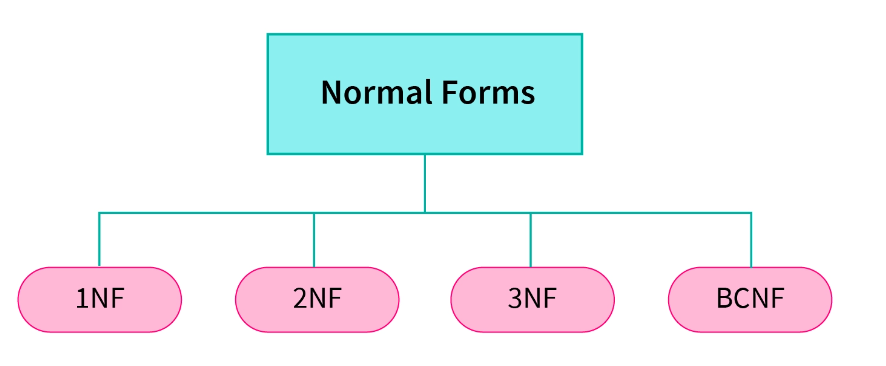
**Faculty Table**

|  |  |  |
| --- | --- | --- |
| Faculty\_id | Faculty\_name | Status |
| F1 | Amit | Notworking |
| F2 | Ayush | Working |
| F3 | Shoib | Working |
| F4 | Paramjeet | Working |

To establish the relationship between student table and course table we need a mediator table because there is a many-to-many relationship.

|  |  |
| --- | --- |
| Student\_id(FK) | Course\_id(FK) |
| S1 | C1 |
| S2 | C1 |
| S3 | C2 |
| S4 | C3 |
| S5 | C2 |

**Normal Forms**



1. 1NF: A relation is in 1NF if all attributes of the table contains only atomic values(no multi-valued attribute).
2. 2NF: A relation is in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the key attribute.
3. 3NF: A relation is in 3NF if it is in 2NF and there is no transitive dependency.
4. BCNF: A relation is in BCNF if it is in 3NF and non-key attribute should not identify key attribute.

**First Normal Form**

A table(relation) is said to be in 1NF if all attributes of the table contain only atomic values (no multi-valued attribute).

|  |  |  |
| --- | --- | --- |
| Employee Code | Employee Name | Employee Phone Number |
| 101 | John | 98765623,998234123 |
| 102 | Ravi | 89023467 |
| 103 | Ryan | 76213908 |
| 104 | Stephanie | 98132452 |

**Change the above table like below (Now below table is in 1NF)**

|  |  |  |
| --- | --- | --- |
| Employee Code | Employee Name | Employee Phone Number |
| 101 | John | 98765623 |
| 101 | John | 998234123 |
| 102 | Ravi | 89023467 |
| 103 | Ryan | 76213908 |
| 104 | Stephanie | 98132452 |

**2nd Normal Form**

|  |
| --- |
| A relation is in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the key attribute. |

|  |  |  |  |
| --- | --- | --- | --- |
| Employee Code | Project ID | Employee Name | Project Name |
| 101 | P03 | John | Project103 |
| 101 | P01 | John | Project101 |
| 102 | P04 | Ryan | Project104 |
| 103 | P02 | Stephanie | Project102 |
| 104 | PO3 | John | Project103 |

1. In the above table (Employee Code, Project ID) is a primary key(composite key), and non-key attributes not fully functionally depending on key attributes, so the above table is not in 2NF.

Note: employe name(non-key attribute) is not depending on composite key((Employee Code, Project ID)) but it is only depending on employee code.

Note: project name (non-key attribute) is not depending on composite key((Employee Code, Project ID)) totally, but it is only depending on project id.

In the above table, the prime attributes of the table are Employee Code and Project ID. We have partial dependencies in this table because Employee Name can be determined by Employee Code and Project Name can be determined by Project ID. Thus, the above relational table violates the rule of 2NF.

To remove partial dependencies from this table and normalize it into second normal form, we can decompose the <EmployeeProjectDetail> table into the following three tables:

**<EmployeeDetail>**

|  |  |
| --- | --- |
| Employee Code | Employee Name |
| 101 | John |
| 102 | Ryan |
| 103 | Stephan |
| 104 | Amir |

**<ProjectDetail>**

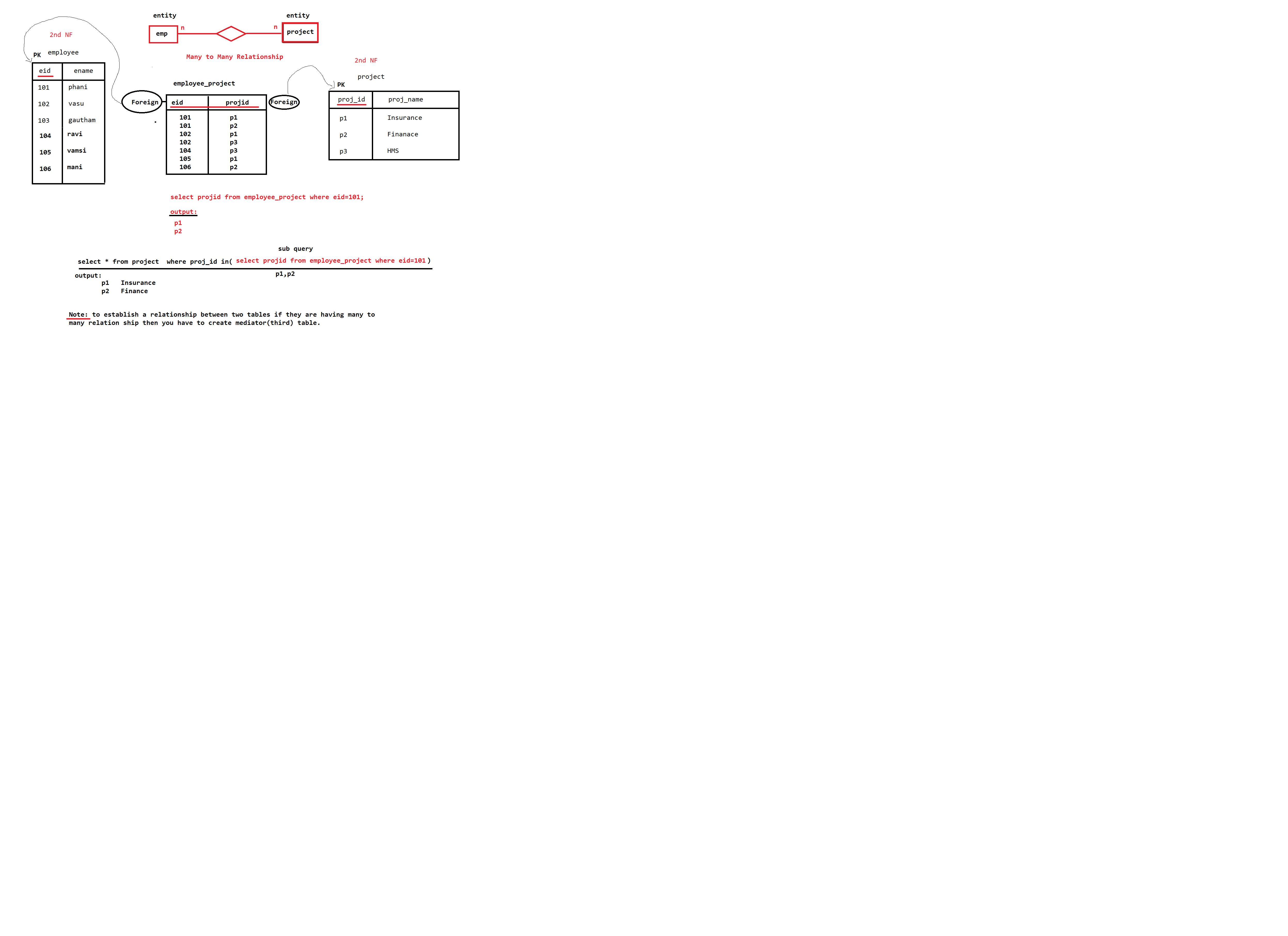
|  |  |
| --- | --- |
| Project ID | Project Name |
| P03 | Project103 |
| P01 | Project101 |
| P04 | Project104 |
| P02 | Project102 |

**<ProjectAllotment >**

|  |  |
| --- | --- |
| Employee Code(FK) | Project ID(FK) |
| 101 | P03 |
| 101 | P01 |
| 102 | P04 |
| 103 | P02 |
| 104 | P01 |

<ProjectAllotment>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Allotment\_id | Employee Code(FK) | | Project ID(FK) | |
| 1 | | 101 | | P03 |
| 2 | | 101 | | P01 |
| 3 | | 102 | | P04 |
| 4 | | 103 | | P02 |
| 5 | | 104 | | P01 |



Thus, we’ve converted the <EmployeeProjectDetail> table into 2NF by decomposing it into <EmployeeDetail>, <ProjectDetail> and <EmployeeProject> tables. As you can see, the above tables satisfy the following two rules of 2NF as they are in 1NF and every non-prime attribute is fully dependent on the primary key.

The relations in 2NF are clearly less redundant than relations in 1NF. However, the decomposed relations may still suffer from one or more anomalies due to the transitive dependency. We will remove the transitive dependencies in the Third Normal Form.

The below commands to implement 2nf example

|  |  |  |
| --- | --- | --- |
| Create database  mysql> create database companydb;  Query OK, 1 row affected (0.01 sec)  Connect to that database  mysql> use companydb;  Database changed  Create employee table    Create projects table    Create emp\_projects table    Inserting data into employee table  insert into employee values(1,'praveen',500000,'2024-01-30'),  (2,'samhith',500000,'2024-01-30'),  (3,'balakiran',500000,'2024-01-30'),  (4,'kusuma',500000,'2024-01-30'),  (5,'Yashasree',500000,'2024-01-30'),  (6,'mahesh',500000,'2024-01-30'),  (7,'gautham',500000,'2024-01-30');  Inserting data into projects table  insert into projects values(101,'Finance','2022-01-20'),  (102,'HMS','2022-01-20'),  (103,'CMS','2022-01-20'),  (104,'EMS','2022-01-20');  Insert data into emp\_projects table  insert into emp\_projects values(1,101),  (1,102),  (2,101),  (2,102),  (3,101),  (3,103),  (4,104),  (5,104),  (6,104),  (7,104);  **Task:** write an sql query to get the all the employees details who are working in project 'EMS'    How to get/display the data from multiple tables?   |  | | --- | | For that we have to joins concept |   **Task:**  Get eid,ename,salary,proj\_id,proj\_name, from employee and projects.  **Joins**  **We use joins to get the data from multiple tables**  CARTESIAN JOIN: The CARTESIAN JOIN is also known as CROSS JOIN.   1. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. 2. This usually happens when the matching column or WHERE condition is not specified. 3. In the absence of a WHERE conditions 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 4. In the presence of WHERE condition this JOIN will function like a INNER JOIN. 5. Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True   Cross join commands   |  | | --- | | cross join (or) CARTESIAN JOIN  ------------------------------------------   1. select eid,ename,salary,proj\_id,proj\_name from employee,projects; 2. select employee.eid,employee.ename,employee.salary,projects.proj\_id,projects.proj\_name from employee,projects; 3. select e.eid,e.ename,e.salary,p.proj\_id,p.proj\_name from employee e,projects p; 4. select eid,ename,salary,proj\_id,proj\_name from employee cross join projects; |   **Inner Join(cross join with where clause):**   1. There should be a common column in both the tables.   **Note:** if you are getting the data from more than one table, which are in many to many relationship then you have to use the 3rd table which we are using to establish a relation.   1. select e.eid,e.ename,e.salary,p.proj\_id,p.proj\_name from employee e inner join projects p inner join emp\_projects ep where e.eid=ep.eid and p.proj\_id=ep.proj\_id; 2. select e.eid,e.ename,e.salary,p.proj\_id,p.proj\_name from employee e join projects p join emp\_projects ep where e.eid=ep.eid and p.proj\_id=ep.proj\_id;   mysql> select e.eid,e.ename,e.salary,p.proj\_id,p.proj\_name from employee e,projects p,emp\_projects ep where e.eid=ep.eid and p.proj\_id=ep.proj\_id;  +-----+-----------+--------+---------+-----------+  | eid | ename | salary | proj\_id | proj\_name |  +-----+-----------+--------+---------+-----------+  | 1 | praveen | 500000 | 101 | Finance |  | 2 | samhith | 500000 | 101 | Finance |  | 3 | balakiran | 500000 | 101 | Finance |  | 1 | praveen | 500000 | 102 | HMS |  | 2 | samhith | 500000 | 102 | HMS |  | 3 | balakiran | 500000 | 103 | CMS |  | 4 | kusuma | 500000 | 104 | EMS |  | 5 | Yashasree | 500000 | 104 | EMS |  | 6 | mahesh | 500000 | 104 | EMS |  | 7 | gautham | 500000 | 104 | EMS |  +-----+-----------+--------+---------+-----------+  10 rows in set (0.00 sec)  Task: get all the employee details who are working in two projects EMS,HMS.  Note: we can use in operator just like logical or operator, see the below example  mysql> select proj\_id from projects where proj\_name='hms' or proj\_name='ems';  +---------+  | proj\_id |  +---------+  | 102 |  | 104 |  +---------+  2 rows in set (0.01 sec)  mysql> select proj\_id from projects where proj\_name in('hms','ems');  +---------+  | proj\_id |  +---------+  | 102 |  | 104 |  +---------+  2 rows in set (0.00 sec)  Getting all employee ids of employees who are working in ems and hms projects  mysql> select eid from emp\_projects where proj\_id in(select proj\_id from projects where proj\_name in ('ems','hms'));  +-----+  | eid |  +-----+  | 1 |  | 2 |  | 4 |  | 5 |  | 6 |  | 7 |  +-----+  6 rows in set (0.04 sec)  This is the command which displays the result of the given task    Task: Getting employee details along with project details who are working in ‘ems’ and ‘hms’ projects.  mysql> select e.eid,e.ename,e.salary,p.proj\_id,p.proj\_name from employee e join projects p join emp\_projects ep where e.eid=ep.eid and ep.proj\_id=p.proj\_id and ep.proj\_id in(select proj\_id from projects where proj\_name in('ems','hms'));  +-----+-----------+--------+---------+-----------+  | eid | ename | salary | proj\_id | proj\_name |  +-----+-----------+--------+---------+-----------+  | 1 | praveen | 500000 | 102 | HMS |  | 2 | samhith | 500000 | 102 | HMS |  | 4 | kusuma | 500000 | 104 | EMS |  | 5 | Yashasree | 500000 | 104 | EMS |  | 6 | mahesh | 500000 | 104 | EMS |  | 7 | gautham | 500000 | 104 | EMS |  +-----+-----------+--------+---------+-----------+  6 rows in set (0.00 sec)  mysql> select e.eid,e.ename,e.salary,p.proj\_id,p.proj\_name from employee e join projects p join emp\_projects ep where e.eid=ep.eid and ep.proj\_id=p.proj\_id and p.proj\_name in('ems','hms');  +-----+-----------+--------+---------+-----------+  | eid | ename | salary | proj\_id | proj\_name |  +-----+-----------+--------+---------+-----------+  | 1 | praveen | 500000 | 102 | HMS |  | 2 | samhith | 500000 | 102 | HMS |  | 4 | kusuma | 500000 | 104 | EMS |  | 5 | Yashasree | 500000 | 104 | EMS |  | 6 | mahesh | 500000 | 104 | EMS |  | 7 | gautham | 500000 | 104 | EMS |  +-----+-----------+--------+---------+-----------+  6 rows in set (0.00 sec) |

## Third Normal Form (3NF)(Must be in 2NF and no transitive dependencies)

Let us take an example of the following <EmployeeDetail> table to understand what transitive dependency is and how to normalize the table to the third normal form:

**<EmployeeDetail>**

|  |  |  |  |
| --- | --- | --- | --- |
| Employee Code | Employee Name | Employee Zipcode | Employee City |
| 101 | John | 110033 | Model Town |
| 102 | Ryan | 110028 | Naraina |
| 103 | Stephanie | 110064 | Hari Nagar |

The above table is not in 3NF because it has Employee Code -> Employee City transitive dependency because:

* Employee Code -> Employee Zipcode
* Employee Zipcode -> Employee City

To remove transitive dependency from this table and normalize it into the third normal form, we can decompose the <EmployeeDetail> table into the following two tables:

**<EmployeeDetail>**

|  |  |  |
| --- | --- | --- |
| Employee Code | Employee Name | Employee Zipcode |
| 101 | John | 110033 |
| 102 | Ryan | 110028 |
| 103 | Stephan | 110064 |
| 104 | Amit | 110033 |

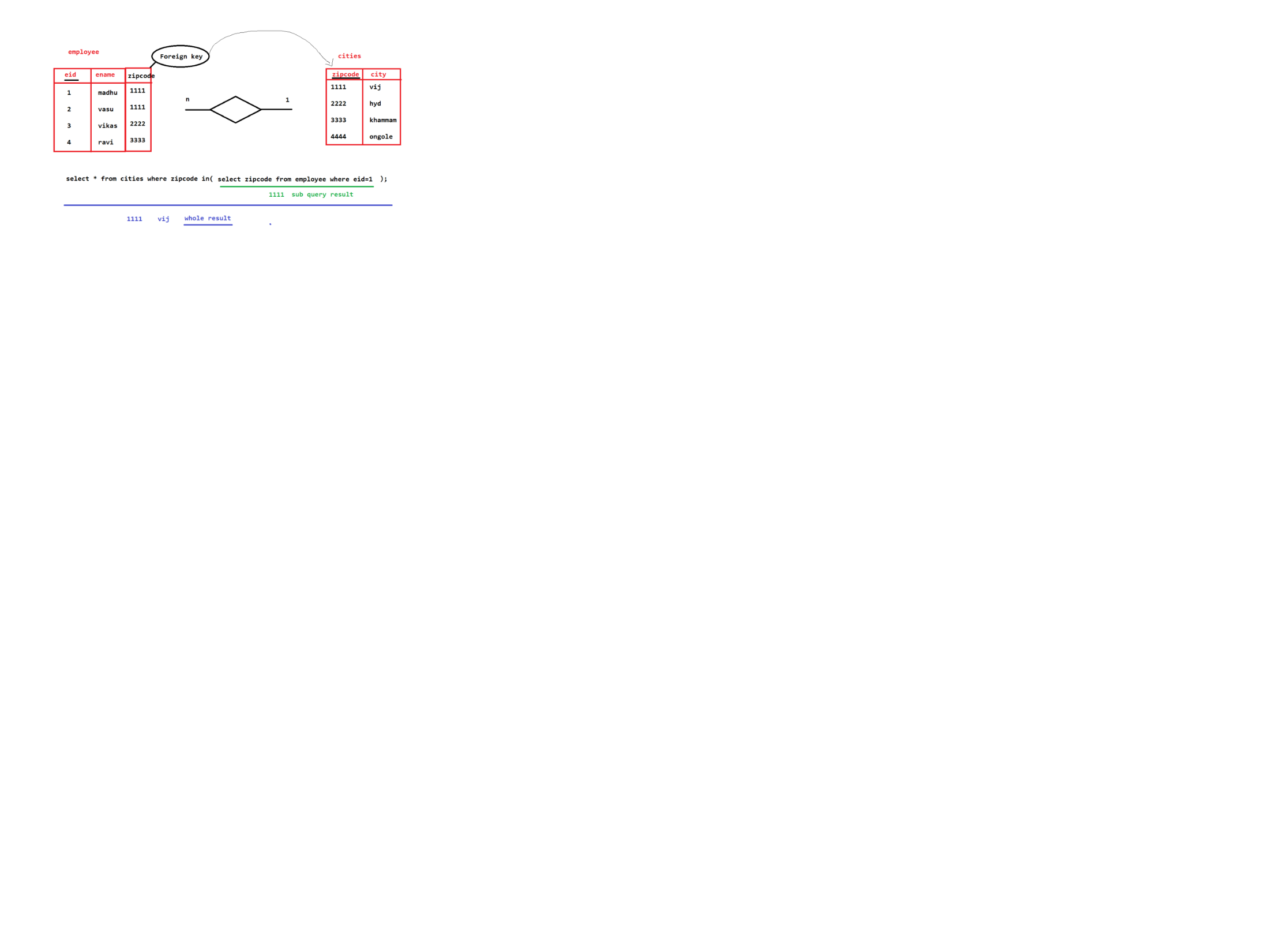
**<EmployeeLocation>**

|  |  |
| --- | --- |
| Employee Zipcode | Employee City |
| 110033 | Model Town |
| 110044 | Badarpur |
| 110028 | Naraina |
| 110064 | Hari Nagar |

Thus, we’ve converted the <EmployeeDetail> table into 3NF by decomposing it into <EmployeeDetail> and <EmployeeLocation> tables as they are in 2NF and they don’t have any transitive dependency.

The 2NF and 3NF impose some extra conditions on dependencies on candidate keys and remove redundancy caused by that. However, there may still exist some dependencies that cause redundancy in the database. These redundancies are removed by a more strict normal form known as BCNF.

|  |
| --- |
| **address table creation**  mysql> create table address(  -> zipcode int,  -> city varchar(50)not null,  -> primary key(zipcode)  -> );  Query OK, 0 rows affected (0.14 sec)  Modifying(altering) employee table (adding foreign key column)  mysql> alter table employee add zipcode int references zipcode(address);  Query OK, 0 rows affected (0.07 sec)  Records: 0 Duplicates: 0 Warnings: 0  **Command to insert the data into address table**  insert into address values(520001,'Vijayawada'),  (500001,'Hyderabad'),  (600001,'Chennai'),  (560001,'Bangalore');  Updating employee table(alloting zipcode to every employee);  update employee set zipcode=520001 where eid=1;  update employee set zipcode=500001 where eid=2;  update employee set zipcode=500001 where eid=3;  update employee set zipcode=600001 where eid=4;  update employee set zipcode=600001 where eid=5;  update employee set zipcode=560001 where eid=6;  update employee set zipcode=560001 where eid=7;  now employee table data is  mysql> select \* from employee;  +-----+-----------+--------+------------+---------+  | eid | ename | salary | doj | zipcode |  +-----+-----------+--------+------------+---------+  | 1 | praveen | 500000 | 2024-01-30 | 520001 |  | 2 | samhith | 500000 | 2024-01-30 | 500001 |  | 3 | balakiran | 500000 | 2024-01-30 | 500001 |  | 4 | kusuma | 500000 | 2024-01-30 | 600001 |  | 5 | Yashasree | 500000 | 2024-01-30 | 600001 |  | 6 | mahesh | 500000 | 2024-01-30 | 560001 |  | 7 | gautham | 500000 | 2024-01-30 | 560001 |  +-----+-----------+--------+------------+---------+  7 rows in set (0.00 sec)  Address table data  mysql> select \*from address;  +---------+------------+  | zipcode | city |  +---------+------------+  | 500001 | Hyderabad |  | 520001 | Vijayawada |  | 560001 | Bangalore |  | 600001 | Chennai |  +---------+------------+  4 rows in set (0.00 sec)  Task: **Display employee details along with city name**  mysql> select e.\*,d.city from employee e inner join address d where e.zipcode=d.zipcode;  +-----+-----------+--------+------------+---------+------------+  | eid | ename | salary | doj | zipcode | city |  +-----+-----------+--------+------------+---------+------------+  | 1 | praveen | 500000 | 2024-01-30 | 520001 | Vijayawada |  | 2 | samhith | 500000 | 2024-01-30 | 500001 | Hyderabad |  | 3 | balakiran | 500000 | 2024-01-30 | 500001 | Hyderabad |  | 4 | kusuma | 500000 | 2024-01-30 | 600001 | Chennai |  | 5 | Yashasree | 500000 | 2024-01-30 | 600001 | Chennai |  | 6 | mahesh | 500000 | 2024-01-30 | 560001 | Bangalore |  | 7 | gautham | 500000 | 2024-01-30 | 560001 | Bangalore |  +-----+-----------+--------+------------+---------+------------+  7 rows in set (0.00 sec) |



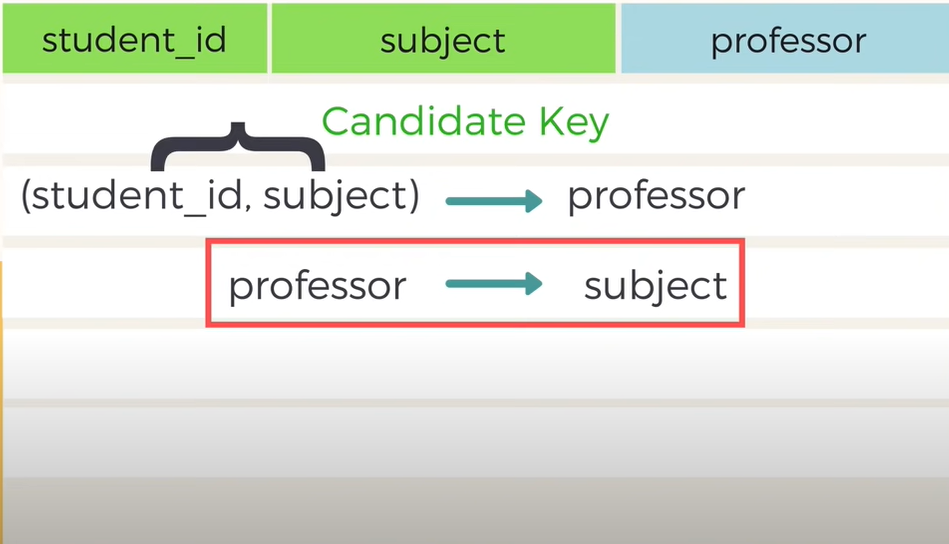
BCNF (Boyce-Code Normal Form) (non-key attribute must not identify the prime(key) attribute).

Note: One student can take training on multiple subjects

Note: One trainer can teach only one subject but to any no.of students

|  |  |  |
| --- | --- | --- |
| Student\_id | Subject | trainer\_id |
| 101 | Java | tid1 |
| 101 | Cpp | tid2 |
| 102 | Java | tid3 |
| 103 | C# | tid4 |
| 104 | Java | tid1 |

Yes: we can find professor by using primary key



Subject is prime attribute because it is part of composite key (primary key) and here professor is non-prime attribute and we are able to find the subject by using professor.

Note: professor can teach only one subject in a college

Student

|  |  |
| --- | --- |
| Student\_id | Professor\_id (ForeignKey) |
| 101 | Pid1 |
| 101 | Pid2 |
| 102 | Pid3 |
| 103 | Pid4 |
| 104 | Pid1 |

Professor (here professor\_id is primary key and subject is non-primary key)

|  |  |
| --- | --- |
| Professor\_id(PK) | Subject |
| Pid1 | Java |
| Pid2 | CPP |
| Pid3 | Java |
| Pid4 | C# |
| Pid5 | RDBMS |

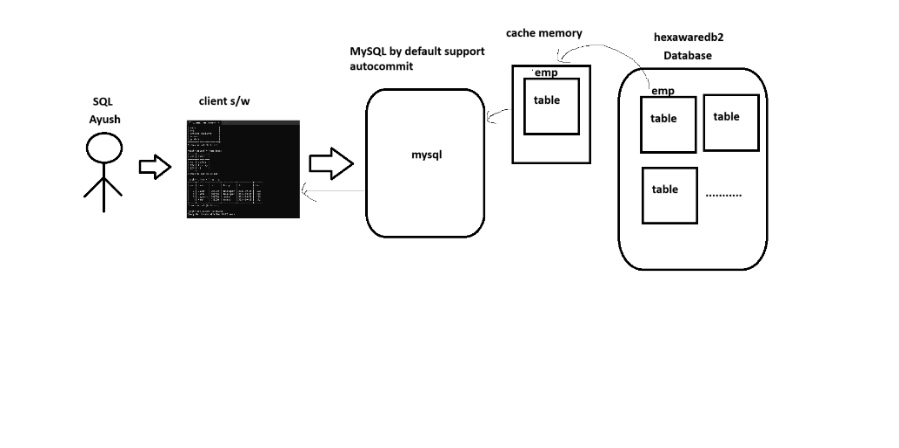
**DML(Commands)**

**Update command**

|  |
| --- |
| Update command is used to update the data existed in a table. |

**What is the use of set autocommit=false;?**

|  |
| --- |
| 1. After performing DML commands the results will be reflected in the tables which are existed in the permanent storage area. Because mysql supports autocommit.   **Note:** In Oracle autocommit is not set to true by default. So We have to give commit command explicitly to save. |



**Usage of set autocommit and rollback**

|  |
| --- |
|  |

**Write a command to get the data of employees who are working in dno 101**

|  |
| --- |
|  |

**Write a command to display the employees who are working in ‘testing’ department.**

|  |
| --- |
| select \* from emp where dno=(select dno from dept where dname='testing'); |

**Example on usage of set autocommit=false, rollback to <savepoint-name>, savepoint.**

|  |
| --- |
| **Continue…**    **Continue…** |

**TCL (Transaction Control Language)**

|  |
| --- |
| 1. Commit 2. Rollback 3. Savepoint   We use the above TCL Commands to manage the transaction. |

**What is transaction?**

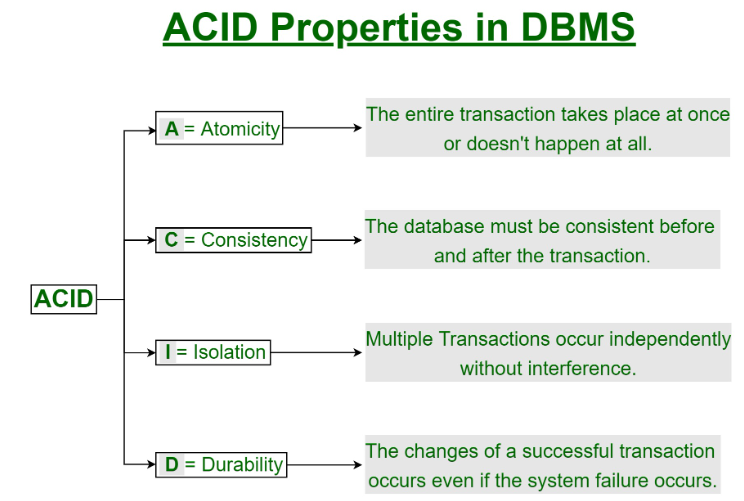
|  |
| --- |
| A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database.  **Example(Bank Transaction):**  **In a Transaction(it is a single logical unit of work) 1 or more operations will be performed**  operation-1: Get amount from your account  operation-2: Check whether enough balance is there or not  operation-3: if enough balance is there, then we debit the amount(withdraw(subtraction)) from your account  operation-4: Credits(add) the amount to father's account |

**Transaction Management**

A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.

**ACID properties**

****

|  |  |  |  |
| --- | --- | --- | --- |
| Atomicity | Consistency | Isolation | Durability |
| Either all or none. | Before the transaction start and after the transaction completed sum of money should be same. | When multiple tasks are executing then cpu switch from one transaction to another transaction while executing them in parellal. Can we make it imparallel without interference of CPU between transaction. | Whatever the changes you are doing, all the changes must be permanent. Even though system failure occurs |

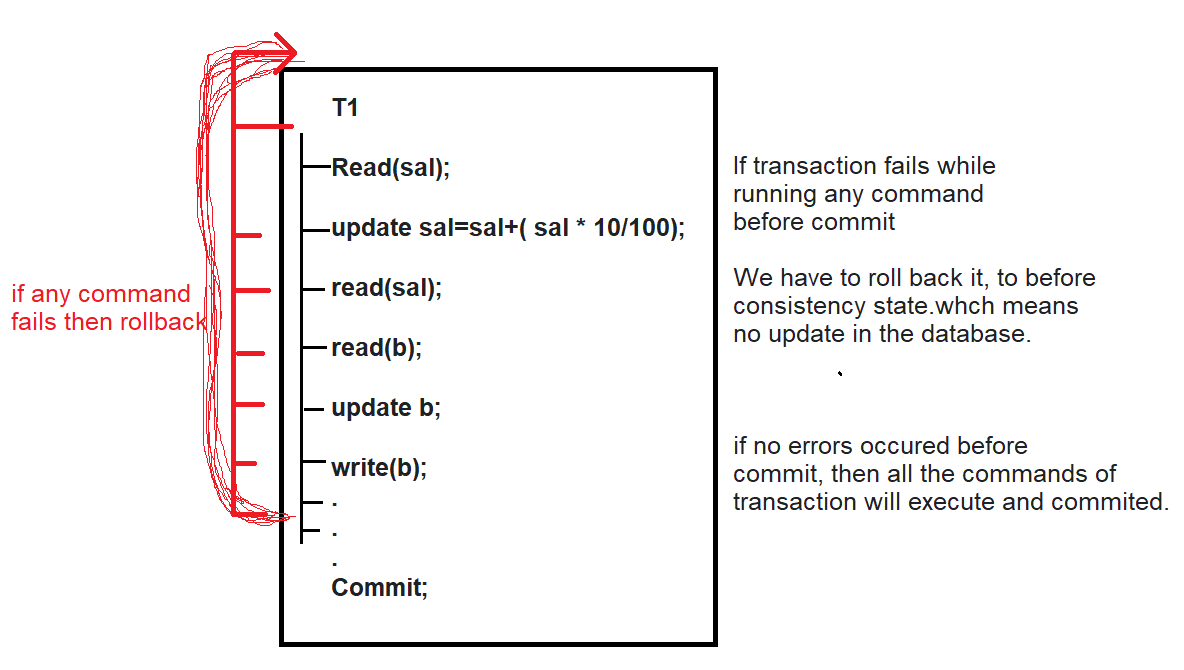
**Parallel vs Serial**

Serial execution − In serial execution, the second transaction can begin its execution only after the first transaction has completed. This is possible on a uniprocessor system.

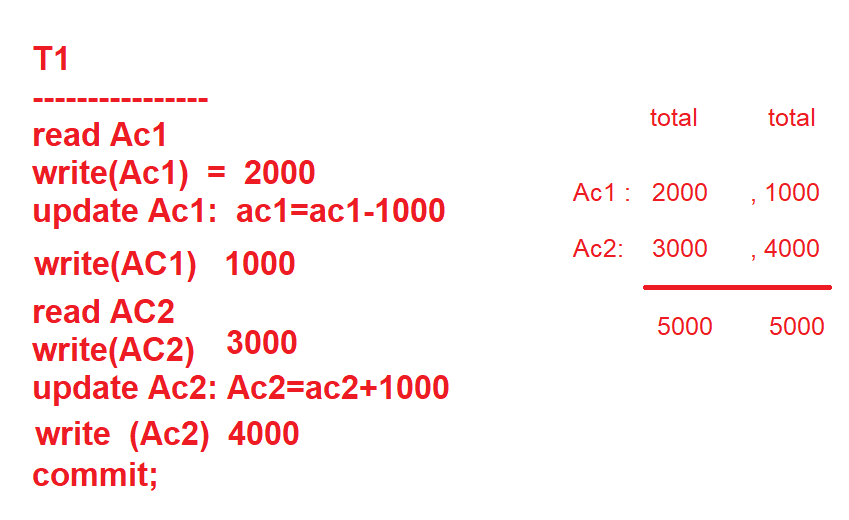
Parallel execution − In parallel execution, two transactions can start their execution at exactly the same instant of time.

Note: failed transaction will not be resumed but it will restart.

1. Either all or none.



1. Before the transaction start and after the transaction completed sum of money should be same. Means Consistent.

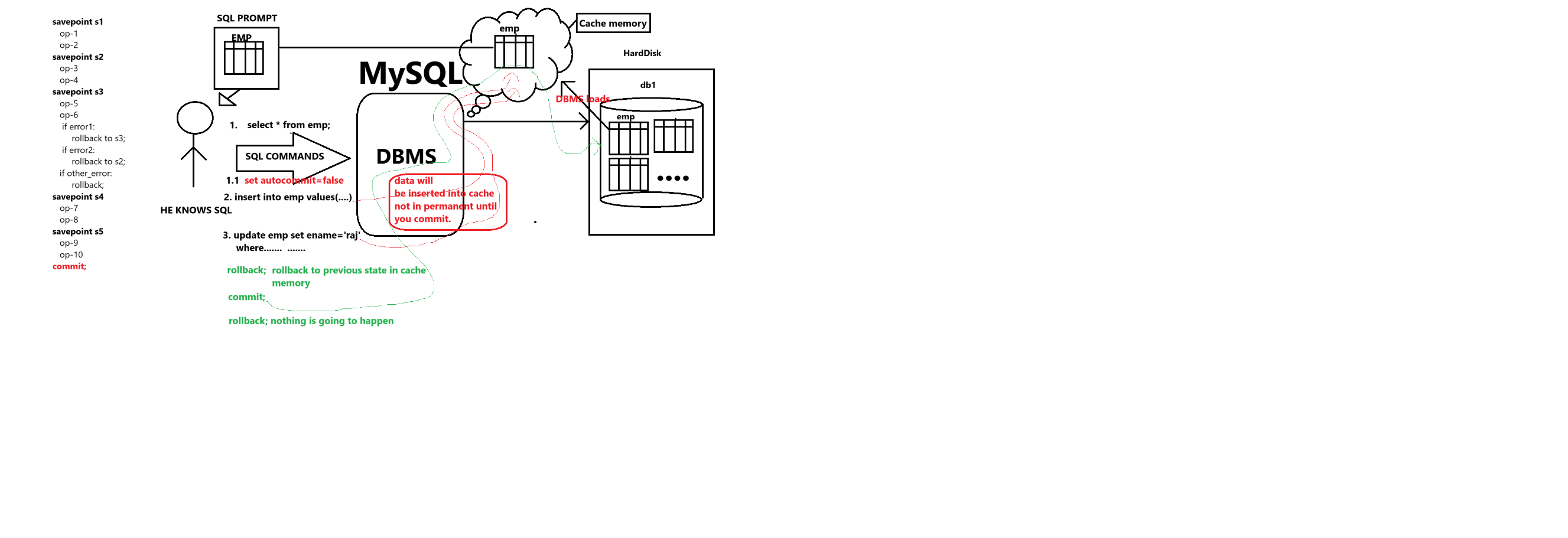


**Isolation**

|  |
| --- |
| When you are performing parallel transactions, then cpu switch from one transaction to another transaction while executing them in parellal. |

****

**Explaining about cache memory**



**What is savepoint**

|  |
| --- |
| A SAVEPOINT is a certain point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction. |

Creating new bankdb databse

|  |
| --- |
| mysql> create database bankdb;  Query OK, 1 row affected (0.02 sec)  **Create table**  CREATE TABLE bank\_account (  account\_id INT PRIMARY KEY AUTO\_INCREMENT,  account\_number VARCHAR(20) UNIQUE NOT NULL,  account\_holder\_name VARCHAR(100) NOT NULL,  balance DECIMAL(10, 2) DEFAULT 0.0,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  );  Insert data into bank\_account table  INSERT INTO bank\_account (account\_number, account\_holder\_name, balance)  VALUES  ('1234567890', 'Balakiran', 10000.00),  ('9876543210', 'Gautham', 20000.00);  Performing transaction    Continue    Continue    Note: Once the commit is done we can’t roll back |

Usage of savepoint and rollback

|  |
| --- |
| mysql> set autocommit=false;  Query OK, 0 rows affected (0.00 sec)  mysql> savepoint before\_insert\_account;  Query OK, 0 rows affected (0.00 sec)  mysql> insert into bank\_account(account\_number,account\_holder\_name,balance) values('1111111111','praveen',1000000);  Query OK, 1 row affected (0.02 sec)  mysql> savepoint before\_withdraw;  Query OK, 0 rows affected (0.00 sec)  mysql> update bank\_account set balance=balance-10000 where account\_number='1111111111';  Query OK, 1 row affected (0.03 sec)  Rows matched: 1 Changed: 1 Warnings: 0  mysql> savepoint before\_deposit;  Query OK, 0 rows affected (0.00 sec)  mysql> update bank\_account set balance=balance+10000 where account\_number='1234567890';  Query OK, 1 row affected (0.00 sec)  Rows matched: 1 Changed: 1 Warnings: 0  mysql> select \* from bank\_account;  +------------+----------------+---------------------+-----------+---------------------+  | account\_id | account\_number | account\_holder\_name | balance | created\_at |  +------------+----------------+---------------------+-----------+---------------------+  | 1 | 1234567890 | Balakiran | 20000.00 | 2024-02-02 06:36:02 |  | 2 | 9876543210 | Gautham | 35000.00 | 2024-02-02 06:36:02 |  | 3 | 1111111111 | praveen | 990000.00 | 2024-02-02 06:53:35 |  +------------+----------------+---------------------+-----------+---------------------+  3 rows in set (0.00 sec)  mysql> rollback to before\_deposit;  Query OK, 0 rows affected (0.00 sec)  mysql> select \* from bank\_account;  +------------+----------------+---------------------+-----------+---------------------+  | account\_id | account\_number | account\_holder\_name | balance | created\_at |  +------------+----------------+---------------------+-----------+---------------------+  | 1 | 1234567890 | Balakiran | 10000.00 | 2024-02-02 06:36:02 |  | 2 | 9876543210 | Gautham | 35000.00 | 2024-02-02 06:36:02 |  | 5 | 1111111111 | praveen | 990000.00 | 2024-02-02 07:07:22 |  +------------+----------------+---------------------+-----------+---------------------+  3 rows in set (0.00 sec)  mysql> rollback to before\_withdraw;  Query OK, 0 rows affected (0.00 sec)  mysql> select \* from bank\_account;  +------------+----------------+---------------------+------------+---------------------+  | account\_id | account\_number | account\_holder\_name | balance | created\_at |  +------------+----------------+---------------------+------------+---------------------+  | 1 | 1234567890 | Balakiran | 10000.00 | 2024-02-02 06:36:02 |  | 2 | 9876543210 | Gautham | 35000.00 | 2024-02-02 06:36:02 |  | 5 | 1111111111 | praveen | 1000000.00 | 2024-02-02 07:07:22 |  +------------+----------------+---------------------+------------+---------------------+  3 rows in set (0.00 sec)  mysql> rollback to before\_insert;  ERROR 1305 (42000): SAVEPOINT before\_insert does not exist  mysql> rollback to before\_insert\_account;  Query OK, 0 rows affected (0.00 sec)  mysql> select \* from bank\_account;  +------------+----------------+---------------------+----------+---------------------+  | account\_id | account\_number | account\_holder\_name | balance | created\_at |  +------------+----------------+---------------------+----------+---------------------+  | 1 | 1234567890 | Balakiran | 10000.00 | 2024-02-02 06:36:02 |  | 2 | 9876543210 | Gautham | 35000.00 | 2024-02-02 06:36:02 |  +------------+----------------+---------------------+----------+---------------------+  2 rows in set (0.00 sec)  mysql> select \* from bank\_account;  +------------+----------------+---------------------+----------+---------------------+  | account\_id | account\_number | account\_holder\_name | balance | created\_at |  +------------+----------------+---------------------+----------+---------------------+  | 1 | 1234567890 | Balakiran | 10000.00 | 2024-02-02 06:36:02 |  | 2 | 9876543210 | Gautham | 35000.00 | 2024-02-02 06:36:02 |  +------------+----------------+---------------------+----------+---------------------+  2 rows in set (0.00 sec)  mysql> insert into bank\_account(account\_number,account\_holder\_name,balance) values('1111111111','praveen',1000000);  Query OK, 1 row affected (0.00 sec)  mysql> commit;  Query OK, 0 rows affected (0.03 sec)  mysql> select \* from bank\_account;  +------------+----------------+---------------------+------------+---------------------+  | account\_id | account\_number | account\_holder\_name | balance | created\_at |  +------------+----------------+---------------------+------------+---------------------+  | 1 | 1234567890 | Balakiran | 10000.00 | 2024-02-02 06:36:02 |  | 2 | 9876543210 | Gautham | 35000.00 | 2024-02-02 06:36:02 |  | 6 | 1111111111 | praveen | 1000000.00 | 2024-02-02 07:12:06 |  +------------+----------------+---------------------+------------+---------------------+  3 rows in set (0.00 sec)  mysql> rollback;  Query OK, 0 rows affected (0.00 sec)  mysql> select \* from bank\_account;  +------------+----------------+---------------------+------------+---------------------+  | account\_id | account\_number | account\_holder\_name | balance | created\_at |  +------------+----------------+---------------------+------------+---------------------+  | 1 | 1234567890 | Balakiran | 10000.00 | 2024-02-02 06:36:02 |  | 2 | 9876543210 | Gautham | 35000.00 | 2024-02-02 06:36:02 |  | 6 | 1111111111 | praveen | 1000000.00 | 2024-02-02 07:12:06 |  +------------+----------------+---------------------+------------+---------------------+  3 rows in set (0.00 sec) |

**Command with example-1**

|  |
| --- |
| set autocommit=0;  select \* from emp;  insert into emp values(1,"madhu1@gmail.com");  insert into emp values(2,"madhu2@gmail.com");  savepoint sp1;  insert into emp values(3,"madhu3@gmail.com");  insert into emp values(4,"madhu4@gmail.com");  rollback to sp1;  select \* from emp; |

Example-2

|  |
| --- |
| SELECT \* FROM ecommerce.customer;  -- transaction started  set autocommit=0;  savepoint sp1;  insert into customer values(1,'Shoib');  savepoint sp2;  insert into customer values(2,'Dinesh');  savepoint sp3;  insert into customer values(3,'Nitish');  rollback to sp3;  commit;  select \* from customer;  -- transaction ended here |

How to add a primary key to a table

|  |
| --- |
|  |

**How to drop a primary key:**

|  |  |
| --- | --- |
| Command to drop a primarykey  **alter table dept drop primary key;**    **Before dropping a primary key, if it is used as foreign key then you should drop it first. But remember we need the foreign key name to drop it.**  **How to find foreign key name?**   |  | | --- | | **Show create table emp**    **Here the foreign key name is `emp\_ibfk\_1`**  **Command to drop the foreign key:**  **Alter table emp drop foreign key emp\_ibfk\_1;**    **Command to drop primary key** | |

**How to modify the non-key attribute as primary key attribute**

|  |
| --- |
|  |

**Command to modify a column as foreign key column**

|  |
| --- |
| mysql> alter table emp add foreign key(dno) references dept(dno);  Query OK, 4 rows affected (0.07 sec)  Records: 4 Duplicates: 0 Warnings: 0 |

**Command to modify a non-key column as primary key**

|  |
| --- |
|  |

**JOINS**

|  |
| --- |
| By using joins, we can get the data from more than one table at a time. |

**Inner Join (Equi join)**

|  |
| --- |
| 1. It gets the common data from both the tables. 2. SQL Inner Join or Equi Join is the simplest join where all rows from the intended tables are cached together if they meet the stated condition. Two or more tables are required for this join. Inner Join can be used with various SQL conditional statements like WHERE, GROUP BY, ORDER BY, etc. |

**Command**

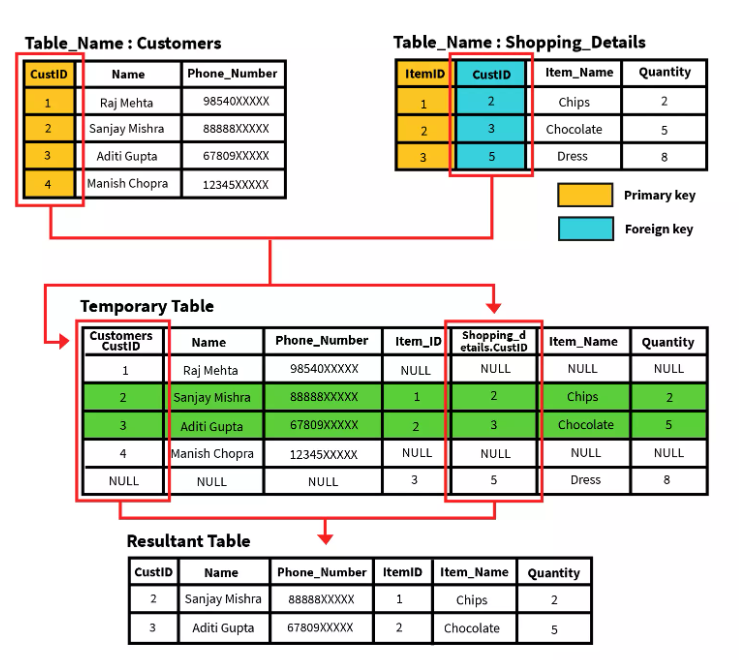
|  |
| --- |
| **Command using inner join keyword and on keyword**    **We can get only some columns from the tables**    **Same command using alias names for tables** |

**Syntax:**

|  |
| --- |
| SELECT column-name  FROM table-1 INNER JOIN table-2  WHERE table-1.column-name = table-2.column-name;  **We can alternately use just the “JOIN” keyword instead of “INNER JOIN”.** |

**Special Case of INNER JOIN: NATURAL JOIN**

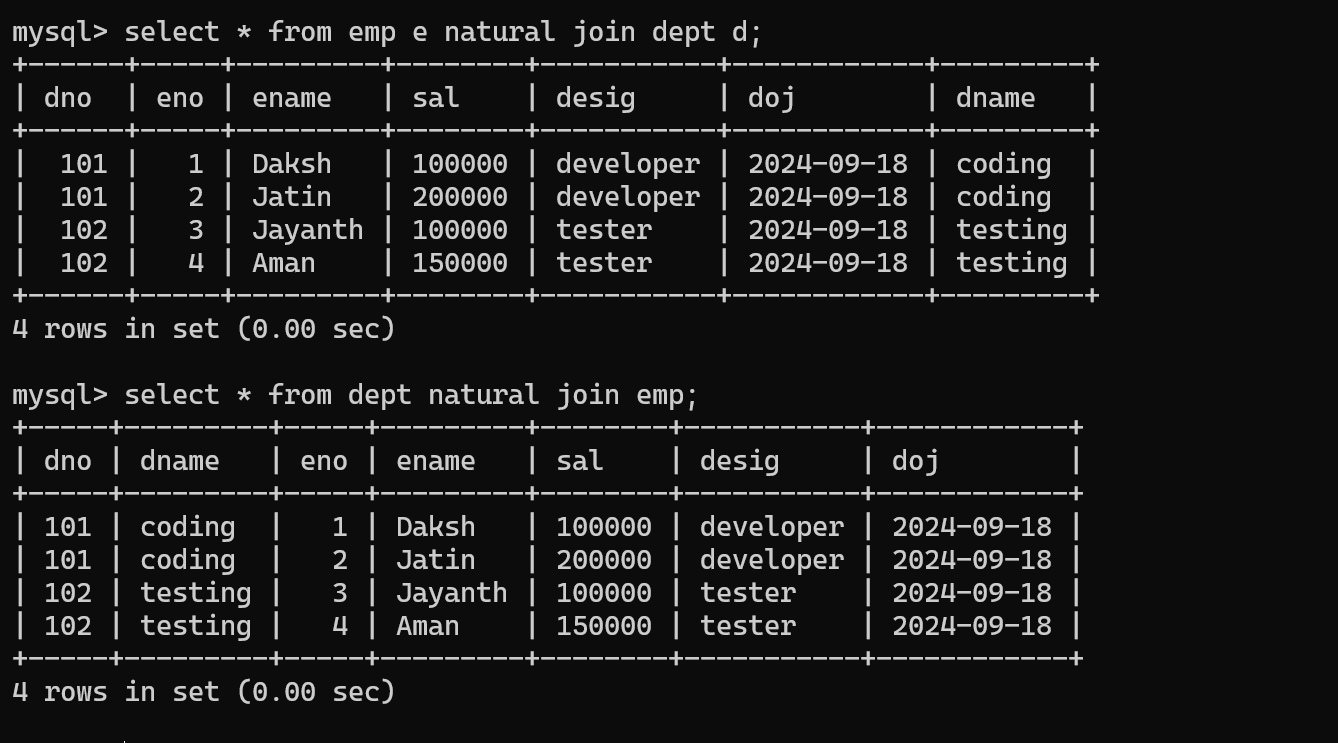
SQL Natural Join is a type of Inner join based on the condition that columns having the same name and datatype are present in both the tables to be joined.



Syntax

|  |
| --- |
| SELECT \* FROM  table-1 NATURAL JOIN table-2; |

**Example:**

****

Problem Statement

Write a query to find all details of customers who bought something from the store.

Query

|  |
| --- |
| SELECT \*  FROM Customers NATURAL JOIN Shopping\_Details; |

OUTER JOINS in SQL

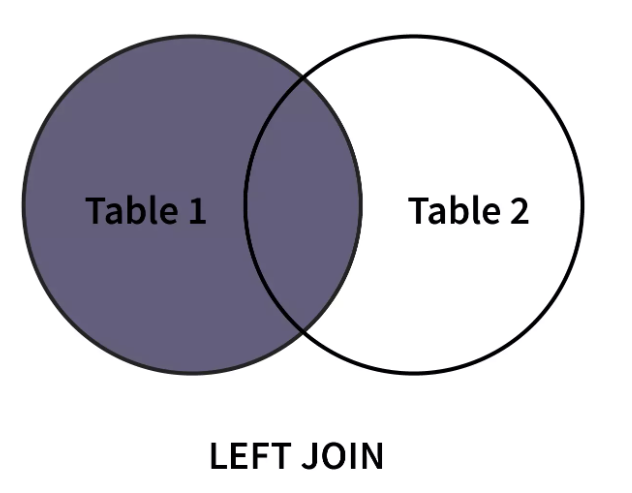
SQL Outer joins give both matched and unmatched rows of data depending on the type of outer joins. These types are outer joins are sub-divided into the following types:

* Left Outer Join
* Right Outer Join
* Full Outer Join

LEFT OUTER JOIN

In this join SQL Left Join, all the rows of the left-hand table, regardless of following the stated conditions are added to the output table. At the same time, only matching rows of the right-hand table are added.

Rows belonging to the left-hand table and not having values from the right-hand table are presented as NULL values in the resulting table.

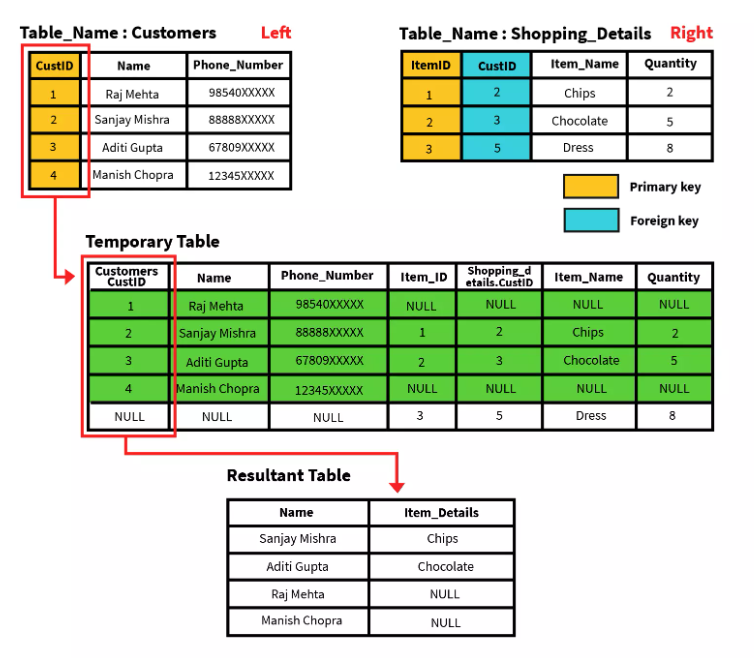


Syntax:

|  |
| --- |
| SELECT column-name(s)  FROM table1(left) LEFT OUTER JOIN table2(right)  ON table1.column-name = table2.column-name; |

Example

Let’s consider two tables of a supermarket set-up. The first table named Customers gives us information about different customers, i.e., their customer id, name, and phone number. Here, CustID is the primary key that uniquely identifies each row. The second table, named Shopping\_Details gives us information about items bought by customers, i.e., item id, customer id (referencing the customer that bought the item), item name, and quantity.

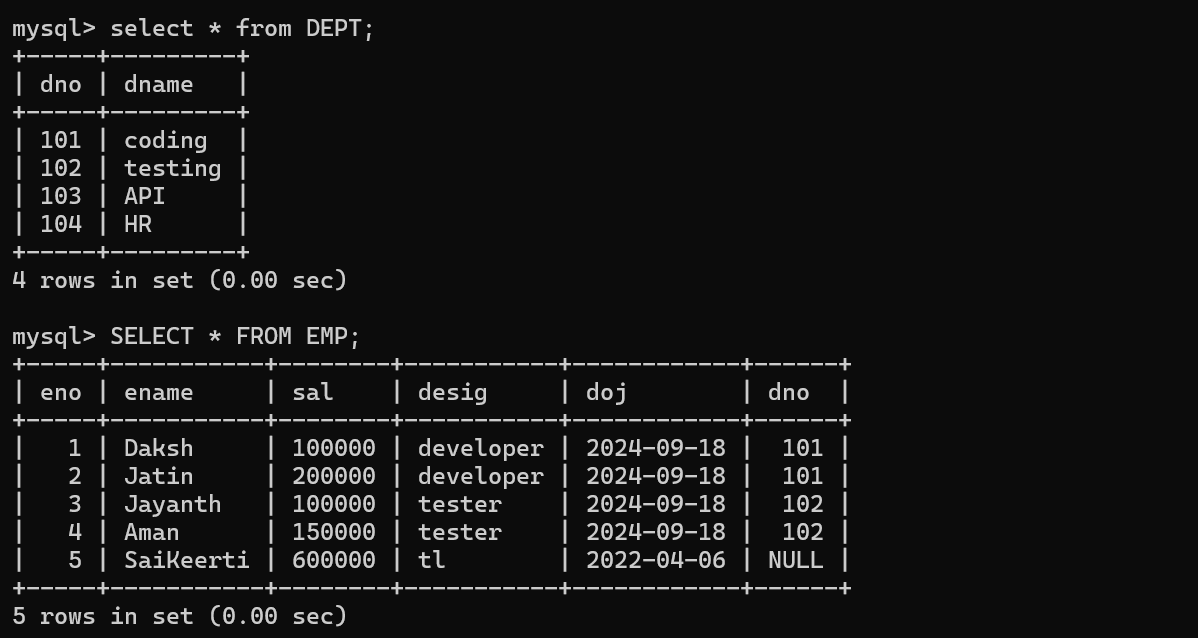


Problem Statement

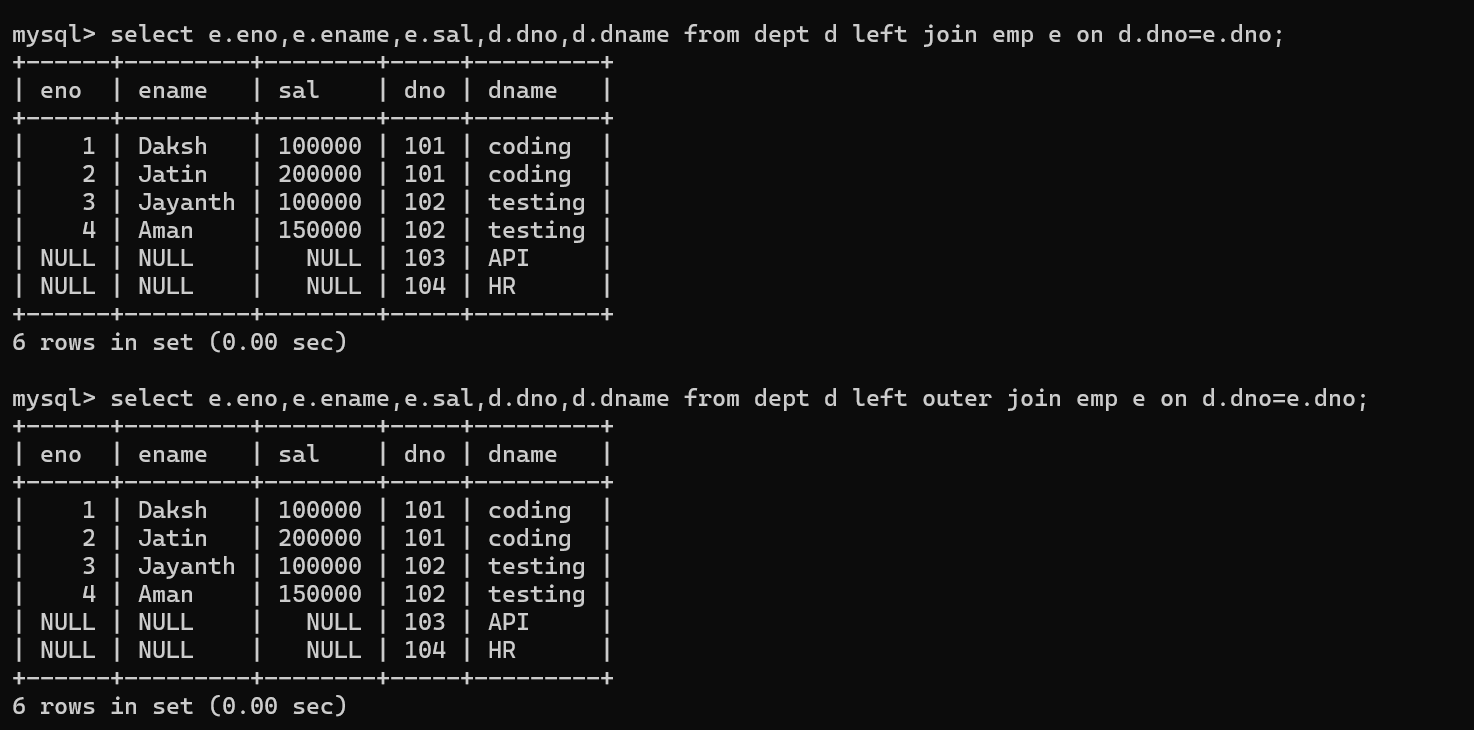
Write a query to display all customers irrespective of items bought or not. Display the name of the customer, and the item bought. If nothing is bought, display NULL.

Query

|  |
| --- |
| SELECT Customers. Name, Shopping\_Details.Item\_Name  FROM Customers LEFT OUTER JOIN Shopping\_Details;  ON Customers.ID = Shopping\_Details.ID; |

****

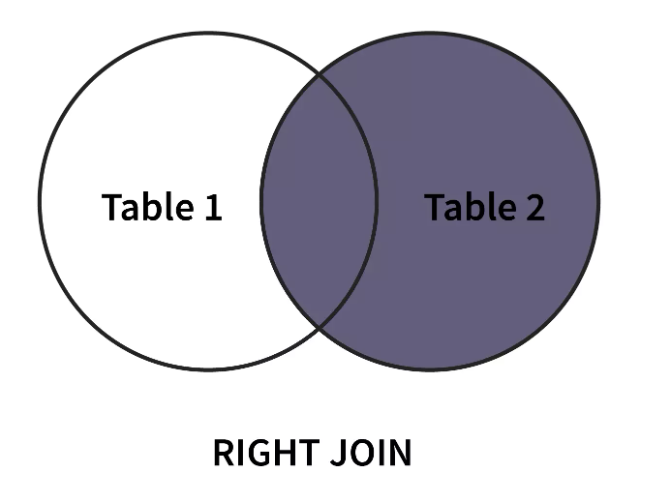
**Example**

****

RIGHT OUTER JOIN

Similar to the left outer join, in the case of the Right Outer Join, a.k.a. SQL Right Join, all the rows on the right-hand table, regardless of following the stated conditions, are added to the output table. At the same time, only matching rows of the left-hand table are added.

Rows belonging to the right-hand table and not having values from the left-hand table are presented as NULL values in the resulting table.

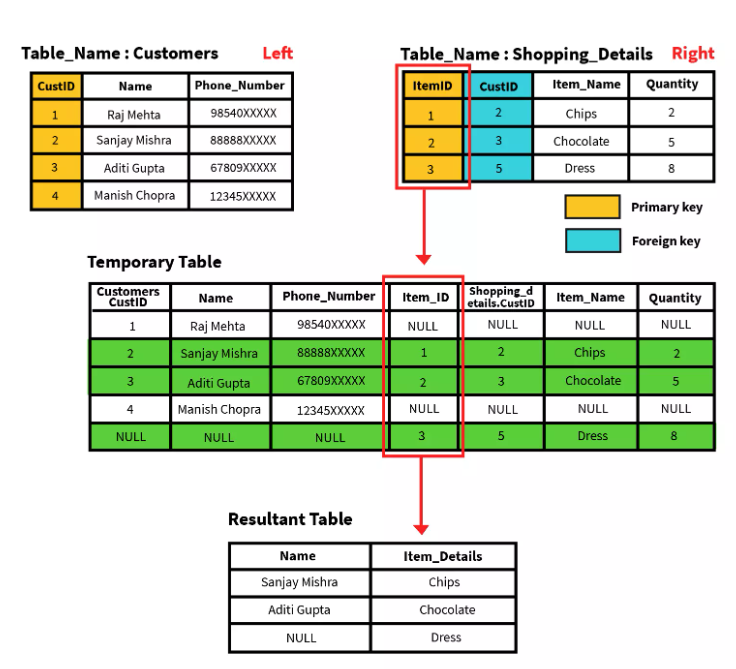


Syntax

|  |
| --- |
| SELECT column-name(s)  FROM table1 RIGHT OUTER JOIN table2  ON table1.column-name = table2.column-name; |

Example

Let’s consider two tables of a supermarket set-up. The first table named Customers gives us information about different customers, i.e., their customer id, name, and phone number. Here, CustID is the primary key that uniquely identifies each row. The second table, named Shopping\_Details gives us information about items bought by customers, i.e., item id, customer id (referencing the customer that bought the item), item name, and quantity.



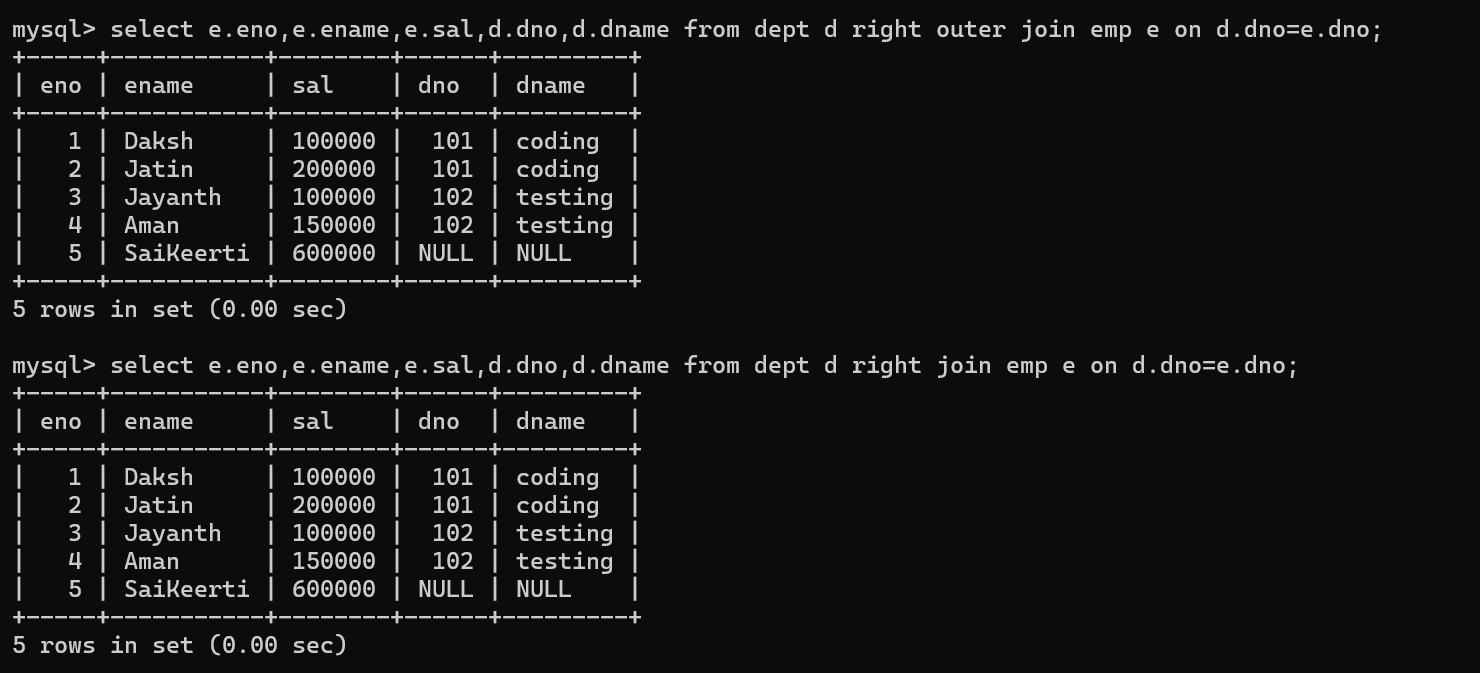
**Problem Statement**

Write a query to get all the items bought by customers, even if the customer does not exist in the Customer database. Display customer name and item name. If a customer doesn’t exist, display NULL.

**Query**

|  |
| --- |
| SELECT Customers.Name, Shopping\_Details.Item\_Name  FROM Customers RIGHT OUTER JOIN Shopping\_Details;  ON Customers.ID = Shopping\_Details.ID; |

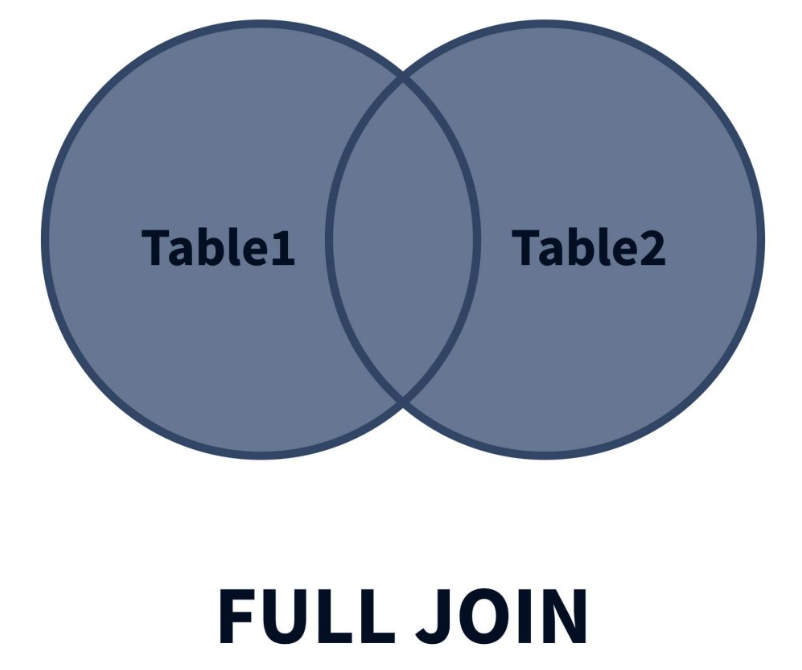
**Example**

****

**FULL OUTER JOIN SUPPORTS IN ORACLE, BUT NOT IN MYSQL**

The full outer join (a.k.a. SQL Full Join) first adds all the rows matching the stated condition in the query and then adds the remaining unmatched rows from both tables. We need two or more tables for the join.

After the matched rows are added to the output table, the unmatched rows of the left-hand table are added with subsequent NULL values, and then unmatched rows of the right-hand table are added with subsequent NULL values.

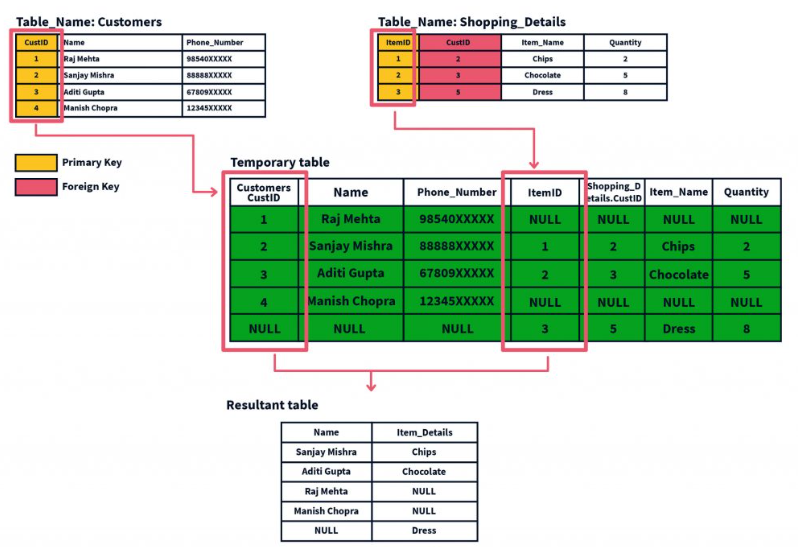


**Syntax:**

|  |
| --- |
| SELECT column-name(s)  FROM table1 FULL OUTER JOIN table2  ON table1.column-name = table2.column-name; |

Example

Let’s consider two tables of a supermarket set-up. The first table named Customers gives us information about different customers, i.e., their customer id, name, and phone number. Here, CustID is the primary key that uniquely identifies each row. The second table, named Shopping\_Details gives us information about items bought by customers, i.e., item id, customer id (referencing the customer that bought the item), item name, and quantity.



Problem Statement

Write a query to provide data for all customers and items ever bought from the store. Display the name of the customer and the item name. If either data does not exist, display NULL.

Query

|  |
| --- |
| SELECT Customers.Name, Shopping\_Details.Item\_Name  FROM Customers FULL OUTER JOIN Shopping\_Details  WHERE Customer.ID = Shopping\_Details.ID; |

How we can implement full outer join in mysql

|  |
| --- |
| By using union we can implement it see the below example |

**SELF JOIN in SQL**

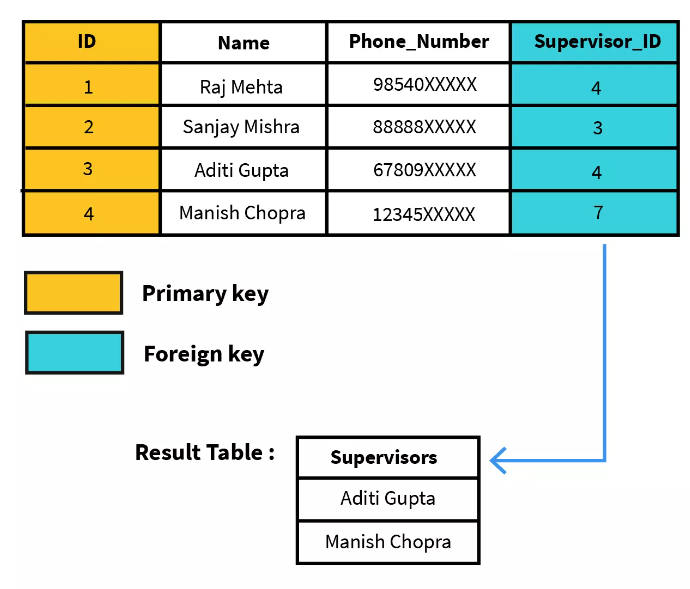
|  |
| --- |
| In SQL Self Join, a table is joined to itself. This means each row of the table is joined with itself and all other rows concerning stated conditions if any. In other words, we can say that it is a merge between two copies of the same table. This is extremely helpful when the foreign key references the primary key of the same table. |

Syntax:

|  |
| --- |
| SELECT a.column1 , b.column2  FROM table\_name a, table\_name b  WHERE some\_condition;  Here we reference the same table with different names, i.e., a and b. This signifies a SELF JOIN. |

Example

Let’s consider an employee table with the following details, i.e., employee id, name, phone number, and supervisor id. The supervisors are present at the employee table itself. Hence, the supervisor id acts like a foreign key which is also the primary key as it references the employee id.



**SELF JOIN in SQL**

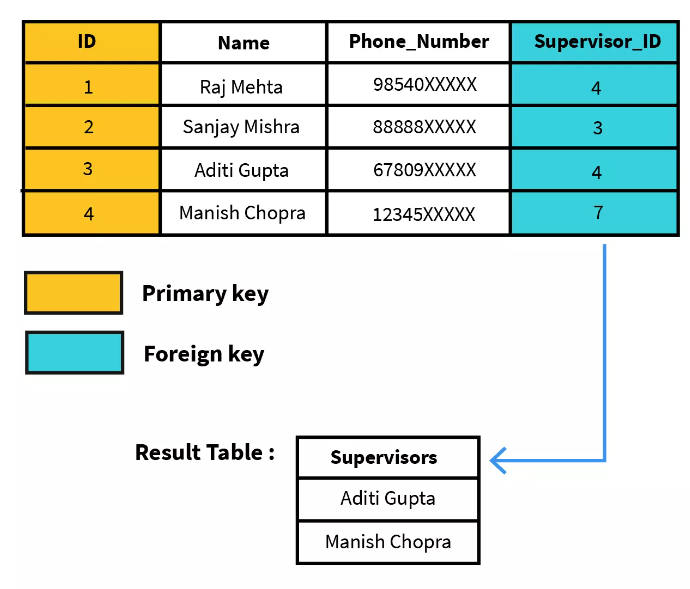
|  |
| --- |
| In SQL Self Join, a table is joined to itself. This means each row of the table is joined with itself and all other rows concerning stated conditions if any. In other words, we can say that it is a merge between two copies of the same table. This is extremely helpful when the foreign key references the primary key of the same table. |

Syntax:

|  |
| --- |
| SELECT a.column1 , b.column2  FROM table\_name a, table\_name b  WHERE some\_condition;  Here we reference the same table with different names, i.e., a and b. This signifies a SELF JOIN. |

Example

Let’s consider an employee table with the following details, i.e., employee id, name, phone number, and supervisor id. The supervisors are present at the employee table itself. Hence, the supervisor id acts like a foreign key which is also the primary key as it references the employee id.



**Create the table before you want to perform self\_join**

|  |  |
| --- | --- |
| create table emp\_supervisors(  eid int,  ename varchar(25)not null,  salary float check(salary>=20000 and salary<=500000),  phone varchar(12)unique,  email varchar(25)unique,  doj date not null,  supervisor\_id int,  primary key(eid),  foreign key(supervisor\_id) references emp\_supervisors(eid)  );    **Inserting data into table**  insert into emp\_supervisors values  (1,'praveen',25000,'8987787878','praveen@gmail.com','2024-12-01',null),  (2,'balakiran',100000,'4987787878','bala@gmail.com','2024-02-01',1),  (3,'kusuma',150000,'5987787878','kusuma@gmail.com','2024-02-01',1),  (4,'mahesh',100000,'6987787878','mahesh@gmail.com','2024-02-01',1),  (5,'gautham',190000,'1987787878','gautham@gmail.com','2024-02-01',4),  (6,'yashasree',90000,'8987787877','yashasree@gmail.com','2024-02-01',4),  (7,'samhith',200000,'8987787875','samhith@gmail.com','2024-02-01',5);    Task: Write a query to get all the employees who are working under Praveen    **Self join examples**      **How the below self join command works**   |  | | --- | | select t1.eid,t1.ename,t2.ename as supervisor\_name from emp\_supervisors t1 join emp\_supervisors t2 on t2.eid=t1.supervisor\_id;  Here in the above command each value of supervisor\_id column of t1 table tries to find matching values in eid column of t2 table if it finds then data will be displayed from both the tables. | |

**Command on usage of between, and operator**

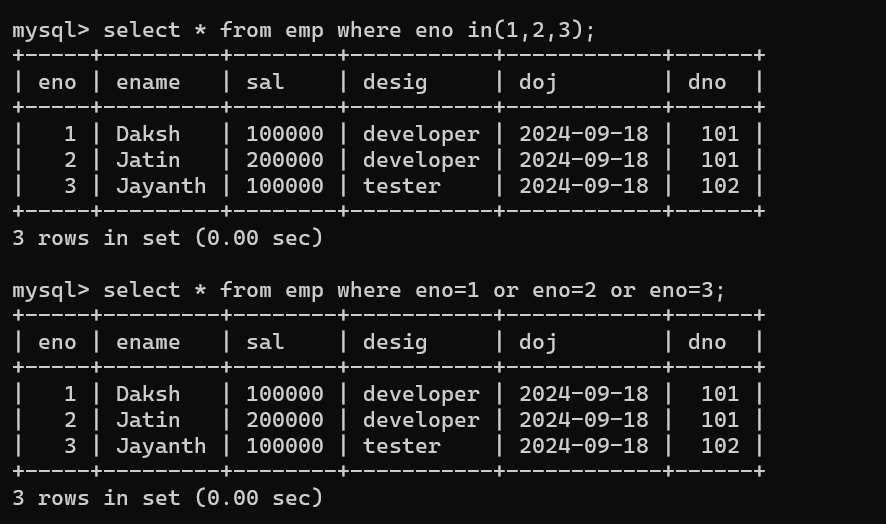
**How to find out the no.of orders which we got between 2023-07-09 to 2023-08-11**

|  |
| --- |
| **Example on < and > operators** |

**In operator**

|  |
| --- |
| 1. The IN operator allows you to specify multiple values in a WHERE clause. 2. The IN operator is a shorthand for multiple OR conditions. |

**Usage of in operator**

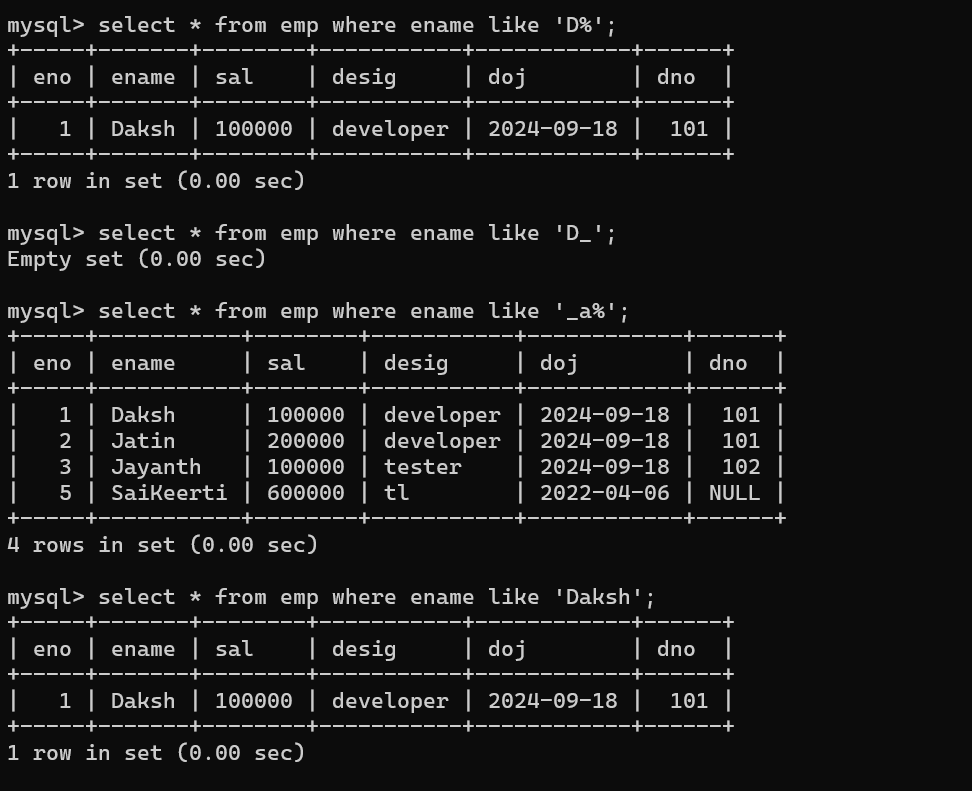
****

**Usage of not in operator**

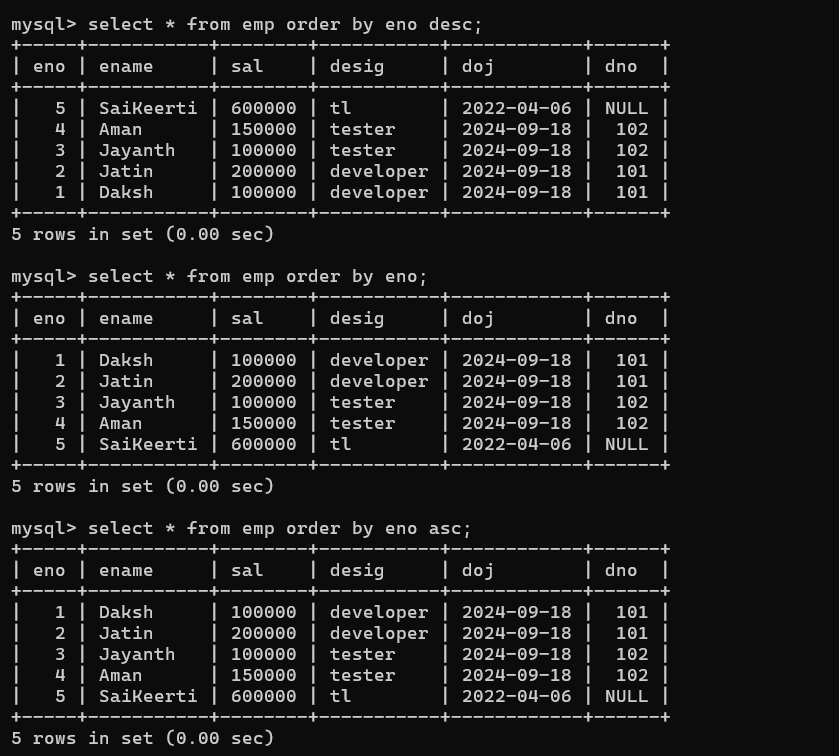
|  |
| --- |
|  |

**Like operator**

|  |
| --- |
| The LIKE operator is used in aWHERE clause to search for a specified pattern in a column.  Along with like operator we can use wild cards  **MySQL Wildcard Characters**   1. % (matches zero or more characters) 2. \_ (matches only one character) |

****

**Order by clause**

****

**Order by clause**

|  |
| --- |
| We use order by clause to display the rows in either ascending or descending order. |

|  |
| --- |
|  |

**Common SQL Aggregate Functions**

**Here are some of the most commonly used SQL aggregate functions:**

**Aggregate functions**

1. Sum()
2. Avg()
3. Min()
4. Max()
5. Count()

**Command to find total count of rows in a table**

|  |
| --- |
|  |

**Write a command to find count of employees who are having salary greater than 100000.**

|  |
| --- |
|  |

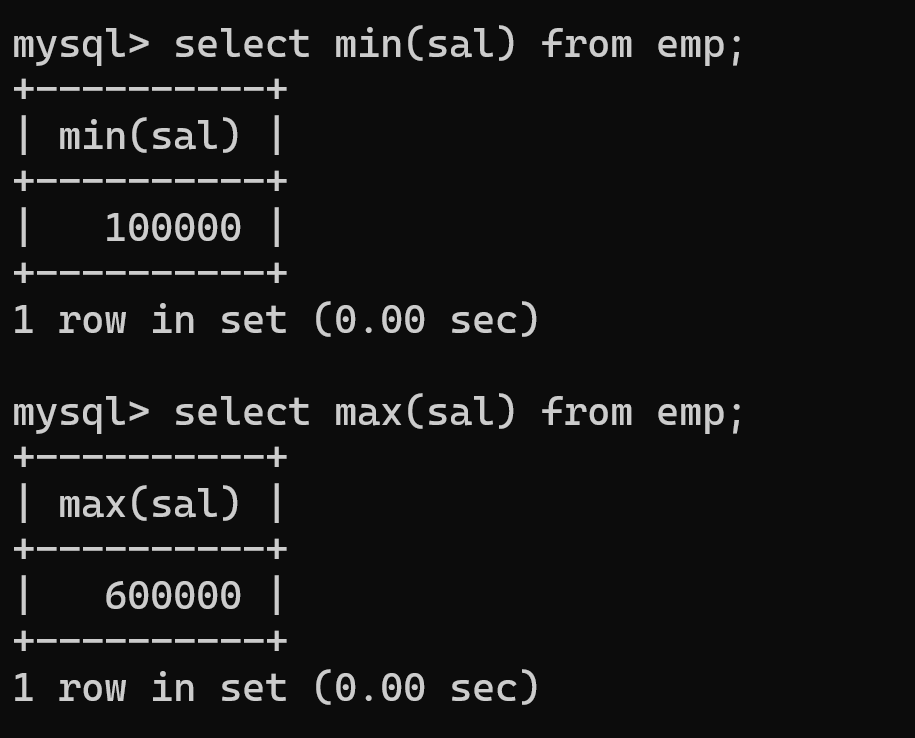
**Write a command to find total sum of salaries of employees**

|  |
| --- |
|  |

**Usage of avg() function**

|  |
| --- |
|  |

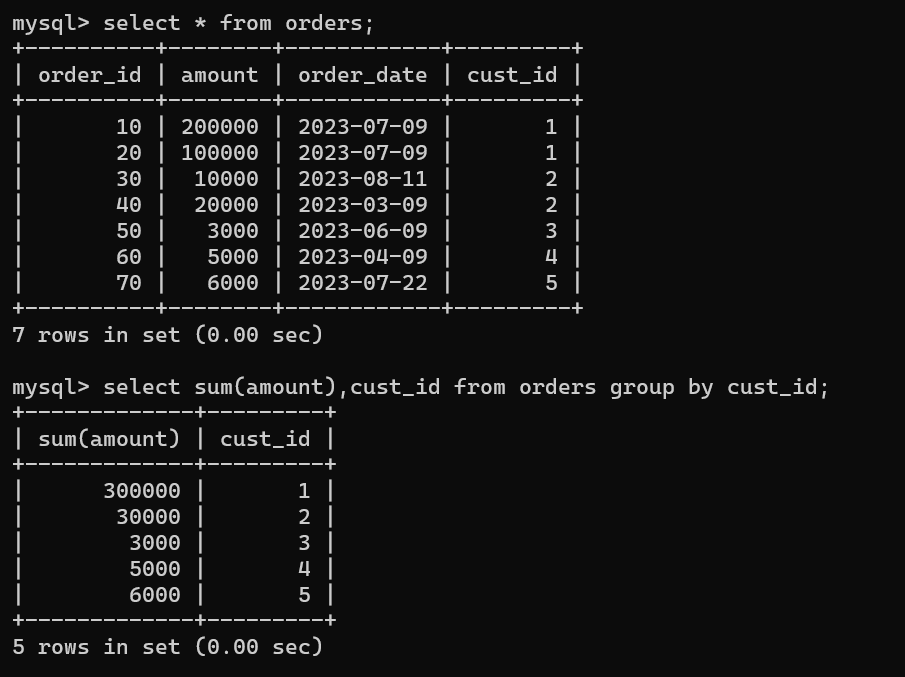
**Find min salary and max salary**

****

**group by clause**

|  |
| --- |
|  |

**Another Example on usage of group by**

****

**Example on usage of aggregate functions in joins**

|  |
| --- |
|  |

**Having clause**

|  |
| --- |
| The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.  **Syntax:**  SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s); |

**Example on having clause usage**

****

**Example on usage of having clause**

|  |
| --- |
|  |

**Another example**

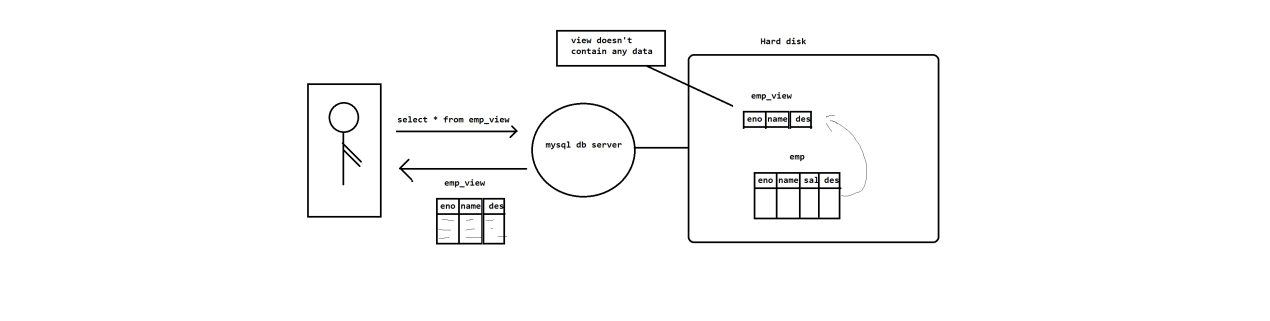
|  |
| --- |
| select ProductName,count(\*) as procount from orders group by ProductName having procount>=3;  select ProductName,count(\*) as procount from orders group by ProductName having productName in ('Product A','Product B'); |

**Views(Virtual Table)**

1. We can create a view by using already existed table
2. If you create a view, it doesn’t contain any separate copy of data
3. View is called as a virtual table which gets the data from the original table.
4. If you modify the data through view, then the modification will be reflected in the original table not in view.

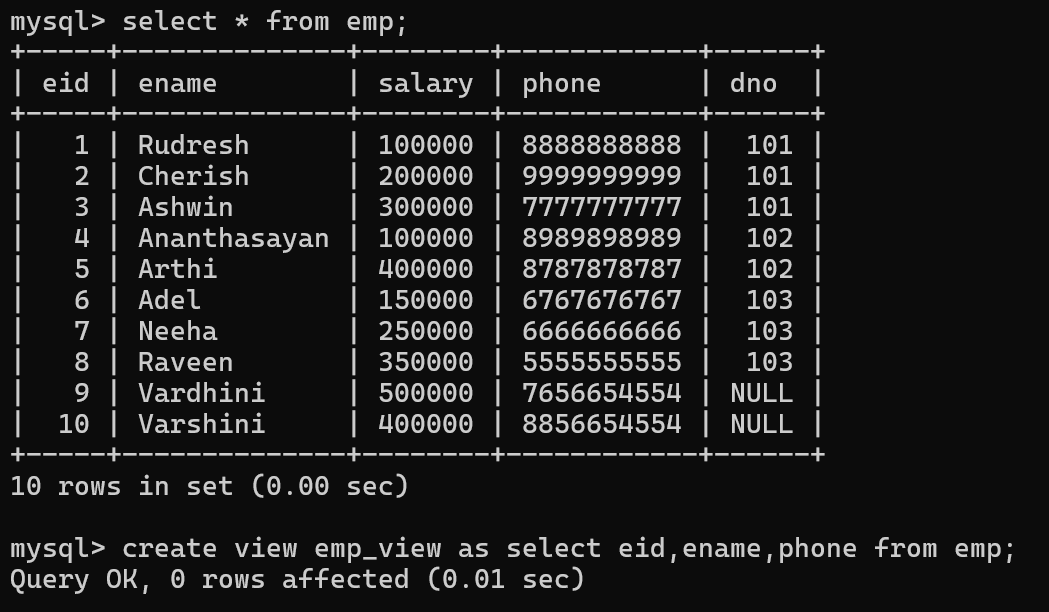
**Working with views**

A view is a database object that doesn’t have any data in it. Its contents are based on the table that serves as the foundation. It has the same rows and columns as a genuine table. In MySQL, a View is a virtual table that is produced by connecting one or more tables in a query. It works in the same way as the base table, but it doesn’t have any data of its own. The fundamental distinction between a view and a table is that views are definitions constructed on top of other tables (or views).

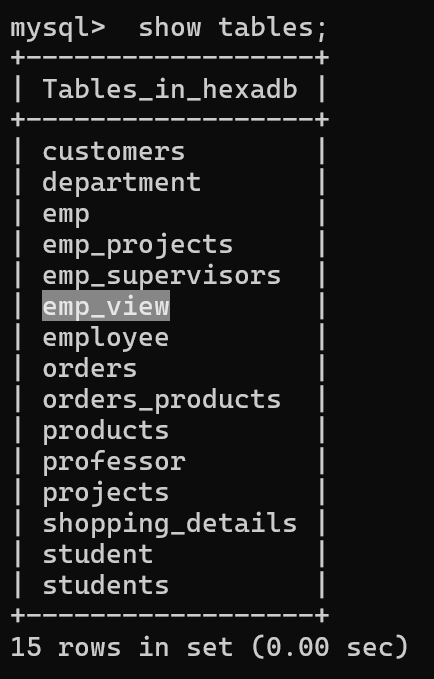


Example on usage of views

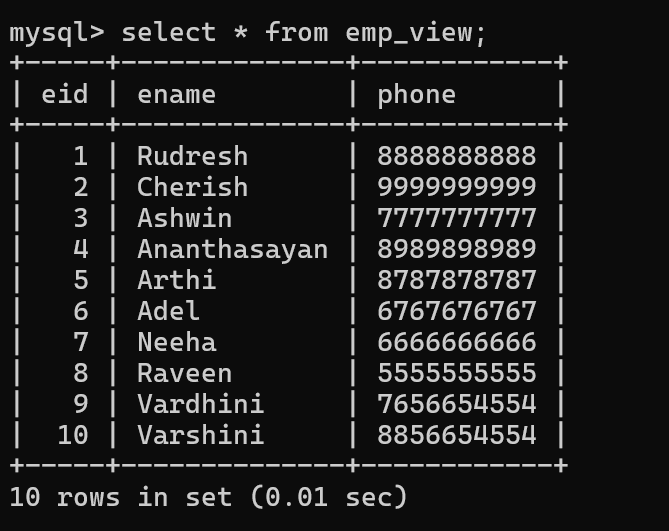
|  |
| --- |
| Let us say we are having two table emp and dept    Note: we know that to access the data from multiple tables we have to use joins  I wanted to get data from emp and dept table many no.of times. So I will create a view with the result of inner join, so that no need to write the same long command every time I wanted to access data from emp and dept table.  Example is |



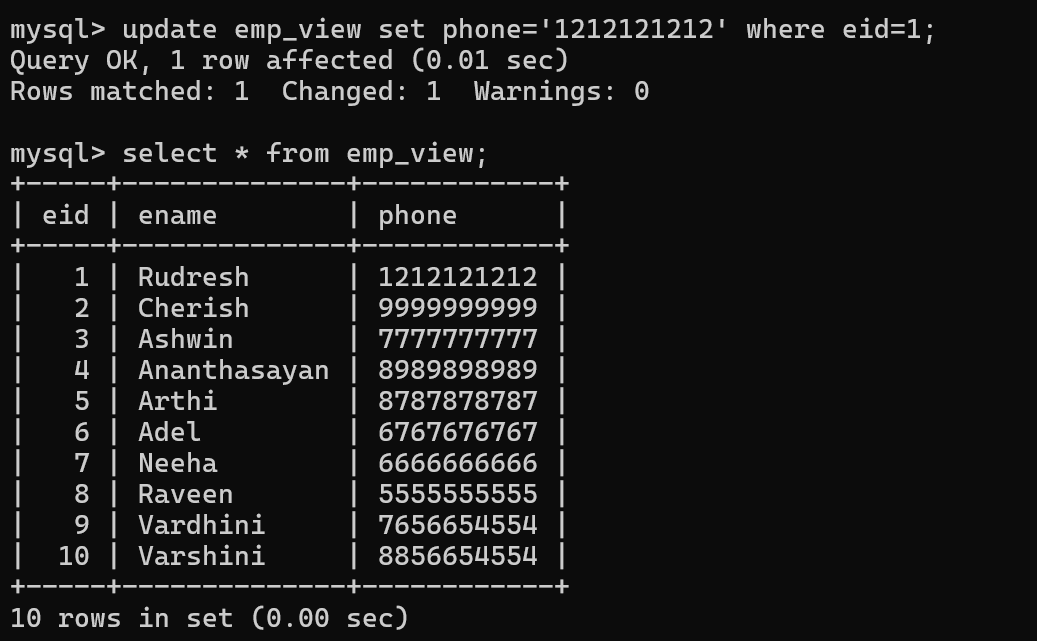
If you want to see the view



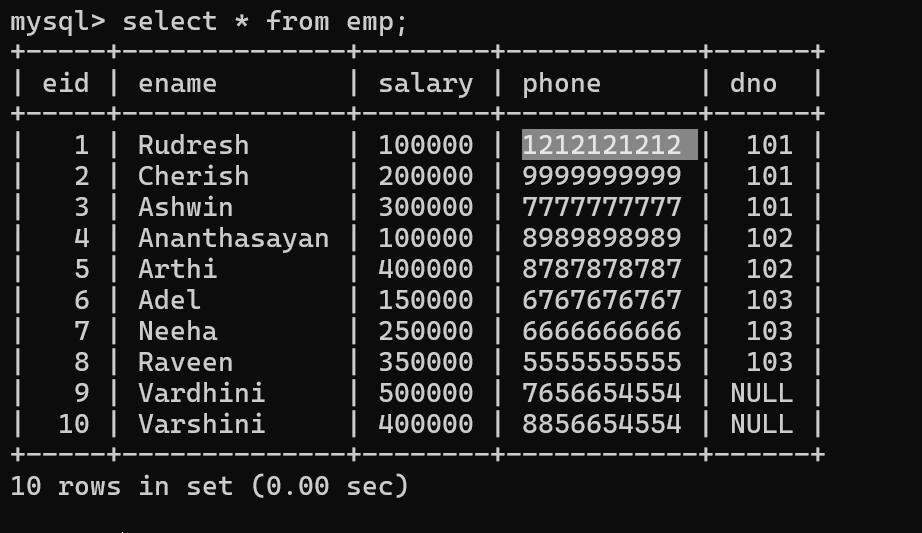
Getting the data of an employee table using emp\_view



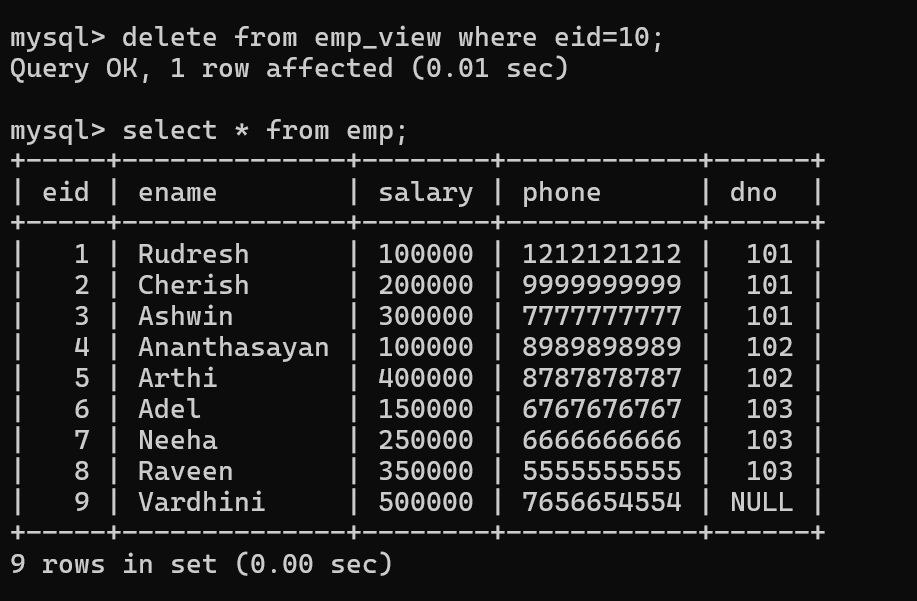
Updating the data through view

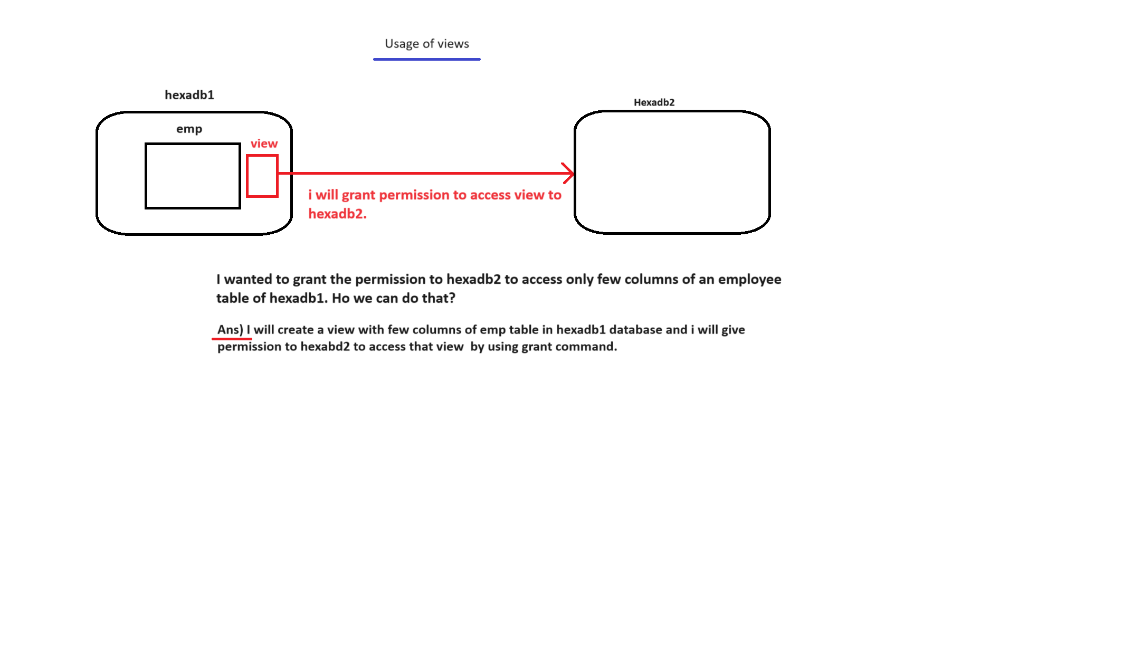


See the table emp the modification you have done are reflected in emp table



Deleting emp details whose id is 10 using view but reflected in emp table





Example-1 on creating view

|  |
| --- |
| mysql> select \* from emp;  +-----+-----------+--------+------------+---------------------+-----------+---------+  | eno | ename | salary | phone | email | desig | dept\_no |  +-----+-----------+--------+------------+---------------------+-----------+---------+  | 1 | samhith | 500000 | 1777777777 | sam@gmail.com | developer | 1 |  | 2 | praveen | 600000 | 2777777777 | praveen@gmail.com | developer | 1 |  | 3 | kusuma | 700000 | 3777777777 | kusuma@gmail.com | developer | 1 |  | 4 | mahesh | 800000 | 4777777777 | mahesh@gmail.com | developer | 1 |  | 5 | balakiran | 900000 | 5777777777 | balakiran@gmail.com | tester | 2 |  | 6 | gautham | 700000 | 6777777777 | gautham@gmail.com | tester | 2 |  | 7 | yashasree | 800000 | 7777777777 | yashasree@gmail.com | hr | 3 |  | 8 | madhu | 50000 | 5656565656 | madhu@gmail.com | NULL | NULL |  +-----+-----------+--------+------------+---------------------+-----------+---------+  8 rows in set (0.00 sec)  mysql> create view emp\_view as select eno,ename,desig from emp;  Query OK, 0 rows affected (0.01 sec)  mysql> select \* from emp\_view;  +-----+-----------+-----------+  | eno | ename | desig |  +-----+-----------+-----------+  | 1 | samhith | developer |  | 2 | praveen | developer |  | 3 | kusuma | developer |  | 4 | mahesh | developer |  | 5 | balakiran | tester |  | 6 | gautham | tester |  | 7 | yashasree | hr |  | 8 | madhu | NULL |  +-----+-----------+-----------+  8 rows in set (0.04 sec) |

Add constraint to column (using alter)

|  |
| --- |
| One-way  mysql> alter table emp modify dept\_no int check(dept\_no>=1 and dept\_no<=10);  Another way  mysql> alter table emp3 add constraint primary key(eno) ;  Query OK, 0 rows affected (0.11 sec)  Records: 0 Duplicates: 0 Warnings: 0  Adding Check constraint to a column  mysql> alter table emp3 add constraint check(salary>=20000 and salary<=50000);  Query OK, 0 rows affected (0.09 sec)  Records: 0 Duplicates: 0 Warnings: 0 |

How to delete a view from database

|  |
| --- |
| mysql> drop view emp\_view;  Query OK, 0 rows affected (0.02 sec) |

Drop multiple tables at a time

|  |
| --- |
| mysql> drop table emp2,emp3;  Query OK, 0 rows affected (0.03 sec) |

Creating a view using columns from both the tables

|  |
| --- |
| mysql> create view emp\_dept as select e.eno,e.ename,e.desig,e.dept\_no,d.dname from emp e join dept d on e.dept\_no=d.dept\_no;  Query OK, 0 rows affected (0.05 sec)  mysql> select \* from emp\_dept;  +-----+-----------+-----------+---------+---------+  | eno | ename | desig | dept\_no | dname |  +-----+-----------+-----------+---------+---------+  | 1 | samhith | developer | 1 | codding |  | 2 | praveen | developer | 1 | codding |  | 3 | kusuma | developer | 1 | codding |  | 4 | mahesh | developer | 1 | codding |  | 7 | yashasree | hr | 3 | hr |  | 5 | balakiran | tester | 2 | testing |  | 6 | gautham | tester | 2 | testing |  +-----+-----------+-----------+---------+---------+  7 rows in set (0.00 sec) |

Create a new view by following these steps:

CREATE VIEW [IF NOT EXISTS] view\_name

AS Select query;

|  |
| --- |
| **Example-1:**  create view emp\_view as select \* from employee;  **Example-2:**  create view cust\_orders\_view as (select o.customerid,o.OrderDate,o.ProductName,c.firstname,c.lastname from customer c inner join orders o on  c.customerid=o.customerid); |

Let’s try to understand it with an example. We are going to create a view from the Marks table mentioned above.

We can use SELECT to see the actual view, just like what we do with tables.

With the WITH CHECK OPTION, create a new view.

CREATE VIEW [IF NOT EXISTS] view\_name

AS select\_statement

WITH CHECK OPTION;

Make a new view or replace an existing one:

CREATE OR REPLACE view\_name

AS

select\_statement;

To drop multiple views:

DROP VIEW [IF EXISTS] view1, view2, …;

Rename a view:

RENAME TABLE view\_name

TO new\_view\_name;

Show views from a database:

SHOW FULL TABLES

[{FROM | IN } database\_name]

WHERE table\_type = ‘VIEW’;

Ex:

SHOW FULL TABLES

**DCL (Data Control Language)**

1. Grant
2. Revoke

**How to create New User**

1. **First, I am going to connect to root user because it is having admin privileges,**
2. **To create new user first we need to connect to a user who is having those privileges**
3. **Connect to root user**

|  |
| --- |
|  |

1. **Create a new user**

|  |
| --- |
| **Or** |

1. How to give permission to user ‘manager’ so that he can create and use new database.

|  |
| --- |
| (or) |

1. Connect to new user

|  |
| --- |
| (or) |

1. Manager user able access all the database of root user. Because he is having all permissions on all databases.

|  |
| --- |
|  |

1. **Tl user is able to access only one database called ‘hexawaredb2’**

|  |
| --- |
|  |

1. **Does tl able to create new database?**

|  |
| --- |
| No, because he is not having sufficient permissions. |

1. **Does ‘manager’ able to create new database?**

|  |
| --- |
| Yes, because he is having all permissions on all databases. |

1. **Tl user can connect to ‘hexawaredb2’**

|  |
| --- |
| **Show tables**    **Create table**    **Inserting data to a table**    **Tl user can’t create new user** |

1. **Manager is able to connect to new database**

|  |
| --- |
| **Show tables**    **Creating new table**    **Insert data to table**    **Manager is creating new user**    **Grant permission to user employee**    **Note:** if a new wants to give grant permissions then you have to create the user with grant option like below    **Revoke permissions from employee** |

**How to see the list of users existed in MySQL?**

|  |
| --- |
|  |

**How to grant only view permission a view to a user**

|  |
| --- |
|  |

How to see the no.of grants allotted to the user?

|  |
| --- |
|  |

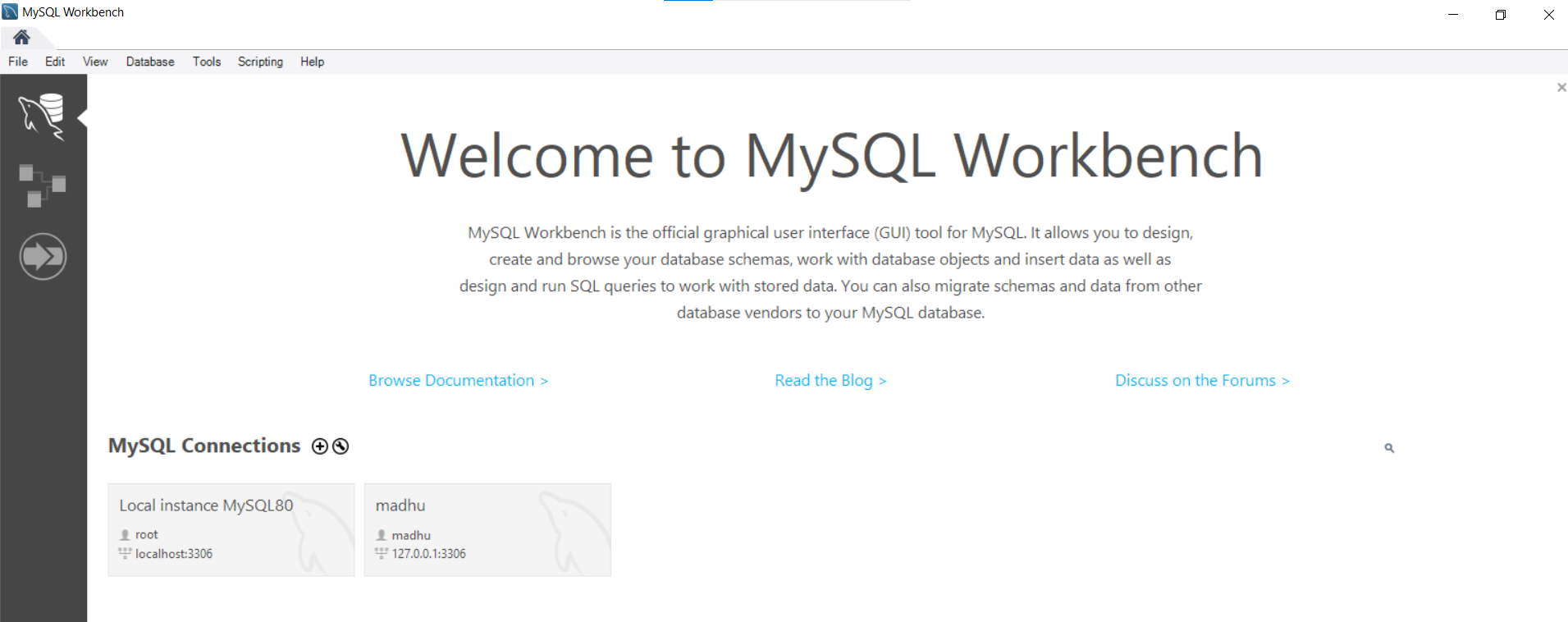
****

How to access the newly created user from workbench?

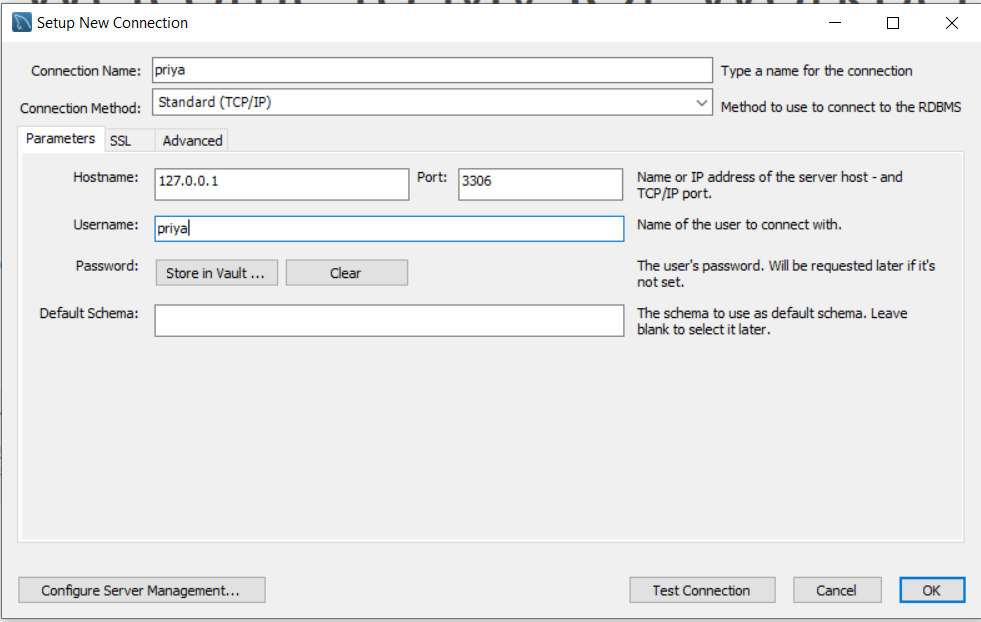
|  |
| --- |
| We have to create a profile through workbench. |

Create a profile for user priya like below

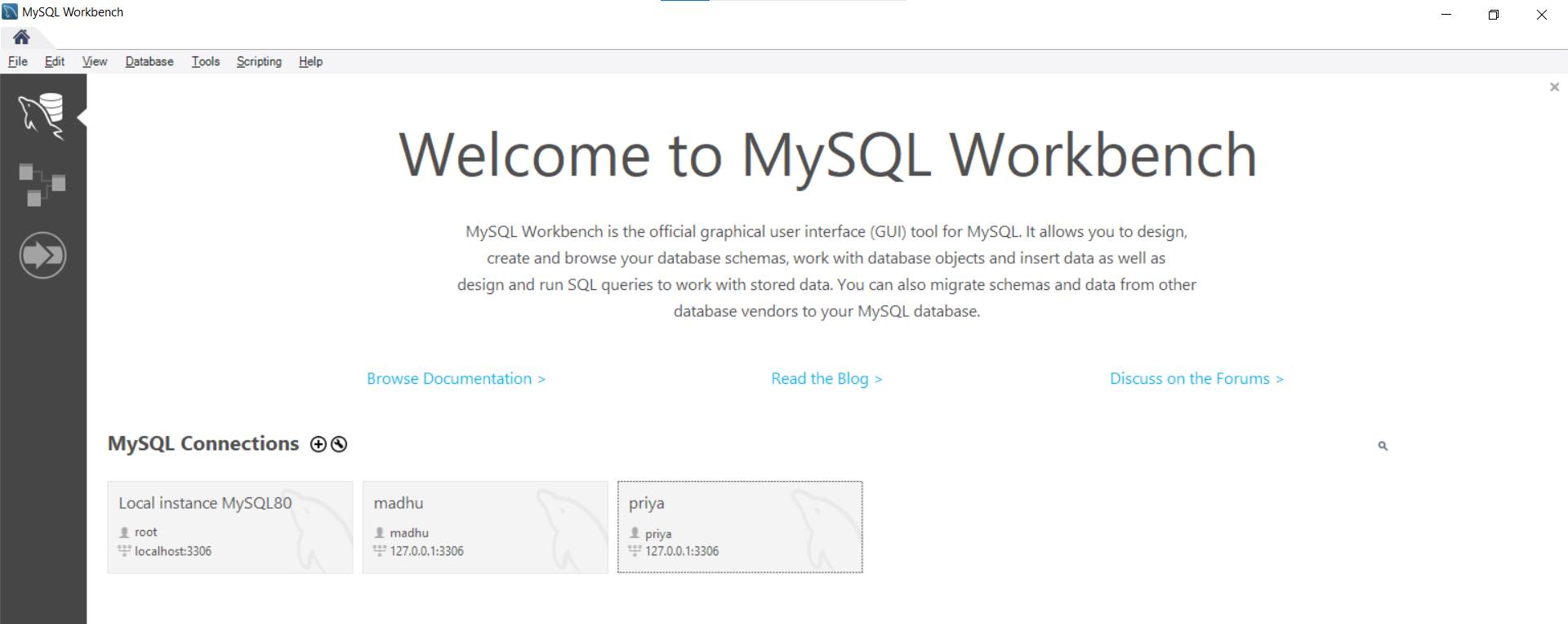
1. Open the workbench



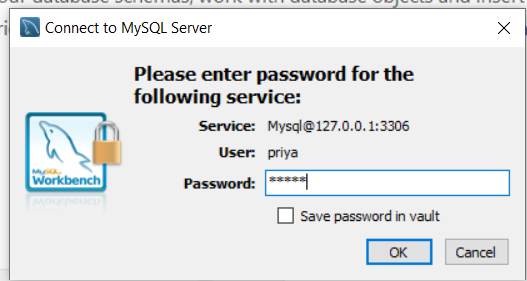
1. Click on + symbol (MYSQL Connections) and fill the username as priya and connection name priya and click on OK button



1. New profile is created



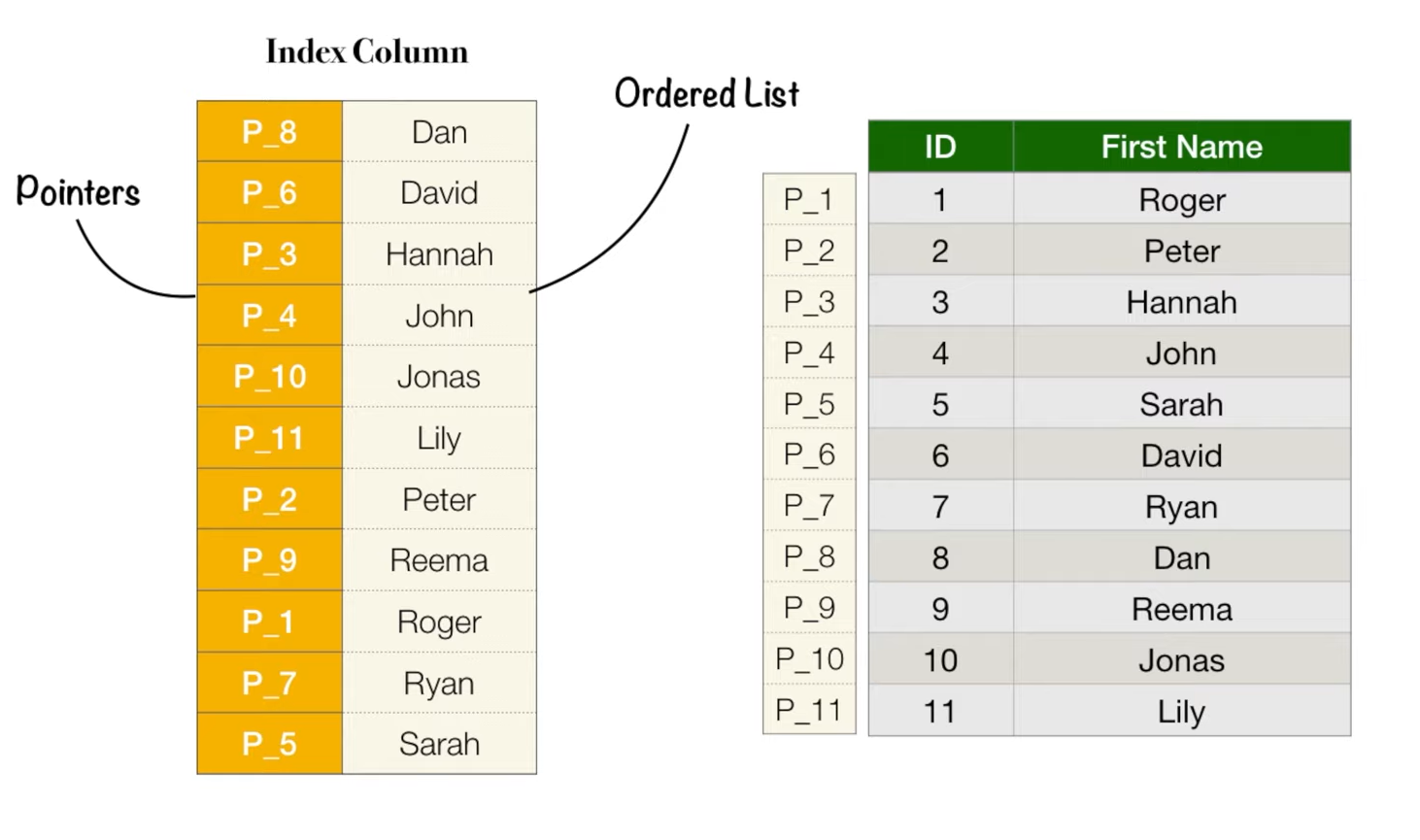
1. Now click on user priya and connect



1. Now we have connected to user priya and it is accessing all the tables from all databases.

**Index**

Assume we have a contact book with the users’ names and phone numbers. We’re looking for a phone number in this contact book. If the contact book is in an unordered manner, which means the names aren’t organized alphabetically, we’ll have to go through all the pages and read every name until we don’t locate the requested name. Sequential searching is the name for this form of search. While this approach is correct, applying this in a large database will consume a lot of time. In this situation, database indexing aids in the retrieval of the desired result and increases the query’s overall performance.

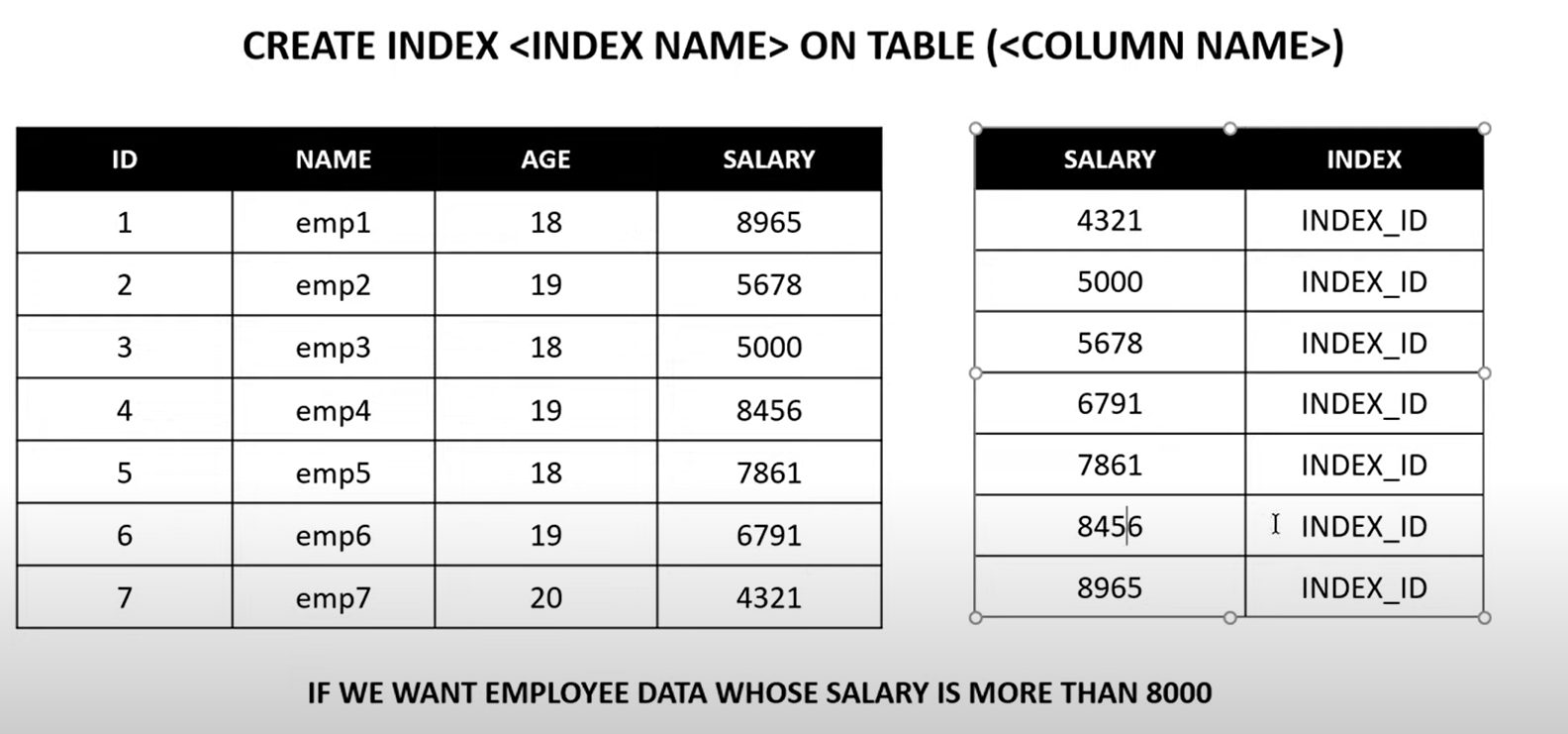


We use indexes to get (search and get) data Fastly from a table

|  |
| --- |
| It uses binary search to find a row |

Indexes are used to quickly locate records with certain column values. It is similar to the index given in a book; you can easily locate a chapter using the index of the book.

|  |
| --- |
| Index makes us to access the data faster.  **Example:**  create index salary\_index on employee (salary asc) |



Select \* from employee where id>=6000

|  |
| --- |
| drop index ename\_index on employee;  create index ename\_index on employee(ename);  select \* from employee where ename='Mohith';  create index enameindex on employee(ename);  create index salaryindex on employee(salary); |

A unique index ensures that the index key includes no duplicate values, ensuring that each entry in the table is distinct in some sense. To create a unique index, use the following syntax:

|  |
| --- |
| create unique index enoindex on employee(eno); |

Drawbacks of indexes

|  |
| --- |
| The insert and update statements take more on tables having indexes, whereas the select statements become fast on those tables. The reason is that while doing insert or update, a database needs to insert or update the index values as well |

**MYSQL functions**

Crete a table

|  |
| --- |
| CREATE TABLE Customers (  CustomerID INT PRIMARY KEY AUTO\_INCREMENT,  CustomerName VARCHAR(255),  Address VARCHAR(255),  PostalCode VARCHAR(20),  City VARCHAR(100)  );  INSERT INTO Customers (CustomerName, Address, PostalCode, City)  VALUES  ('Rajesh Kumar', '123 MG Road', '110001', 'New Delhi'),  ('Priya Sharma', '45 Aurobindo Marg', '400001', 'Mumbai'),  ('Anil Kapoor', '12 Brigade Road', '560001', 'Bangalore'),  ('Meena Gupta', '67 Anna Salai', '600002', 'Chennai'),  ('Suresh Reddy', '89 Banjara Hills', '500034', 'Hyderabad'); |

**String functions**

|  |
| --- |
| The below functions are used to perform string operations. |

What is the keyword I can use as print() function in mysql?

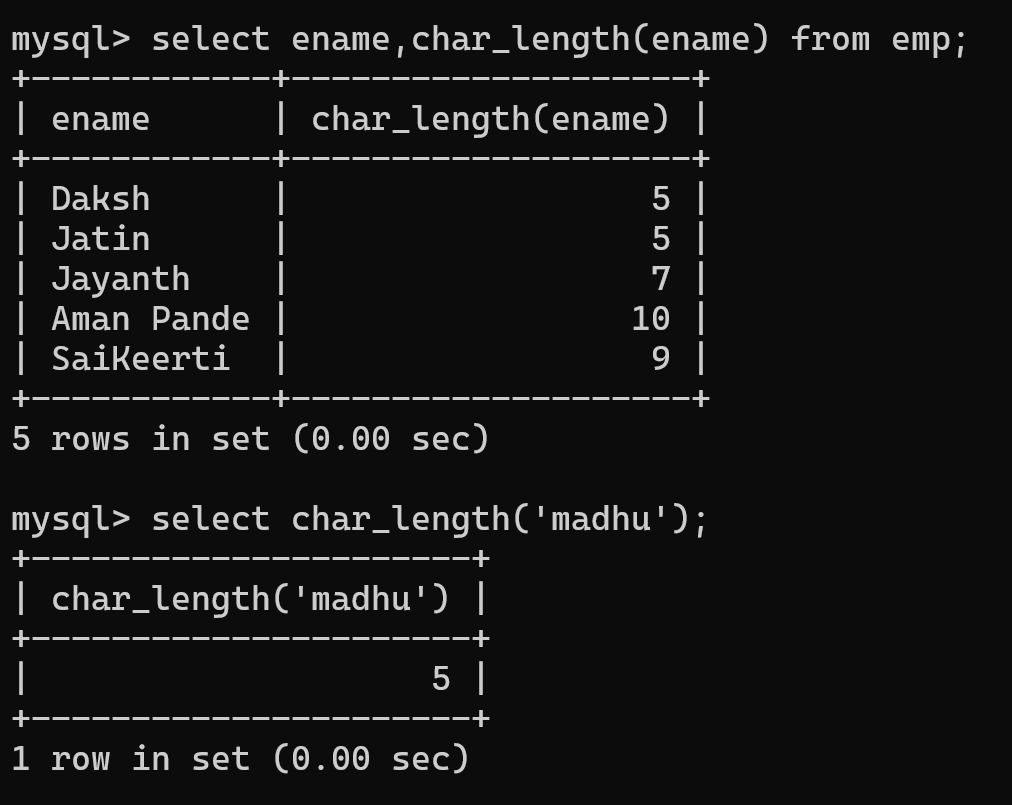
|  |
| --- |
| Select keyword |

1. ASCII() function

The ASCII() function returns the ASCII value of the specific character. If more than one character is entered, it will only return the value for the first character

|  |
| --- |
| SELECT ASCII(CName) AS NumCodeOfFirstChar FROM Customer; |

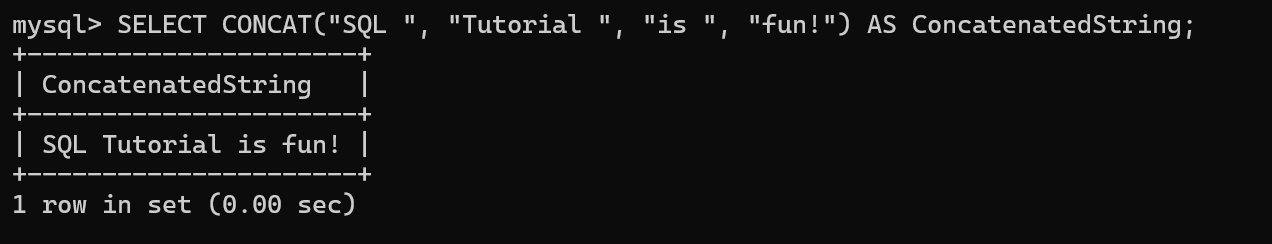
1. CHAR\_LENGTH() function

****

1. CONCAT() function

|  |
| --- |
|  |

Another example



1. CONCAT\_WS()

|  |
| --- |
| The CONCAT\_WS() function adds two or more expressions together with a separator.  mysql> SELECT CONCAT\_WS("-", "SQL", "Tutorial", "is", "fun!") AS ConcatenatedString; |

1. FIELD() function

|  |
| --- |
| * The FIELD() function returns the index position of a value in a list of values. * This function performs a case-insensitive search.   **Note:** If the specified value is not found in the list of values, this function will return 0. If value is NULL, this function will return 0. |

WAC to determine the position of certain cities in a predefined list of cities. Here's an example:

|  |
| --- |
| SELECT CustomerName, City, FIELD(City, 'Mumbai', 'New Delhi', 'Bangalore', 'Chennai', 'Hyderabad') AS CityRank FROM Customers; |

FIND\_IN\_SET() Function

|  |
| --- |
|  |

FORMAT() Function

|  |
| --- |
| Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places. |

INSERT() Function

|  |
| --- |
| Inserts a string within a string at the specified position and for a certain number of characters |

|  |
| --- |
|  |

[INSTR](https://www.w3schools.com/mysql/func_mysql_instr.asp)

|  |
| --- |
| Returns the position of the first occurrence of a string in another string |

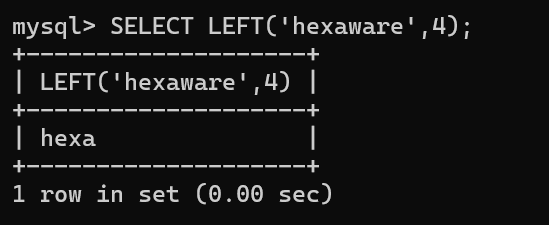
LCASE() and LOWER() function

|  |
| --- |
| 1. Converts a string to lower-case 2. It is used in some database systems like **MySQL**. 3. It performs the same function as LOWER() by converting all characters in a string to lowercase. |

|  |
| --- |
|  |

LEFT() Function

|  |
| --- |
| Extracts a number of characters from a string (starting from left) |

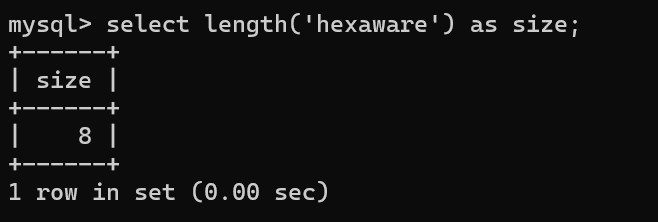


RIGHT() Function

|  |
| --- |
| Extracts a number of characters from a string (starting from right) |

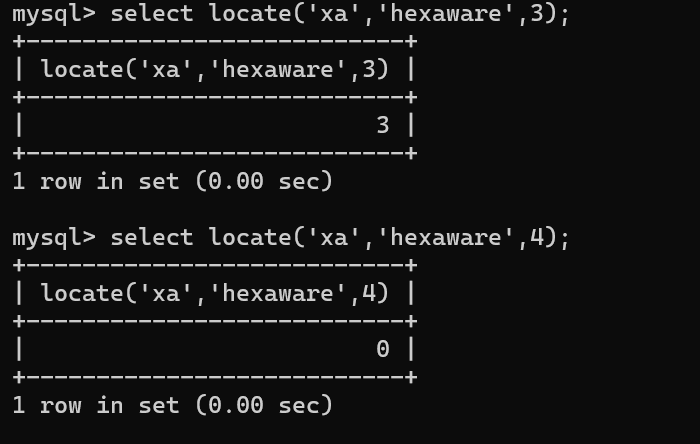
LENGTH() Function

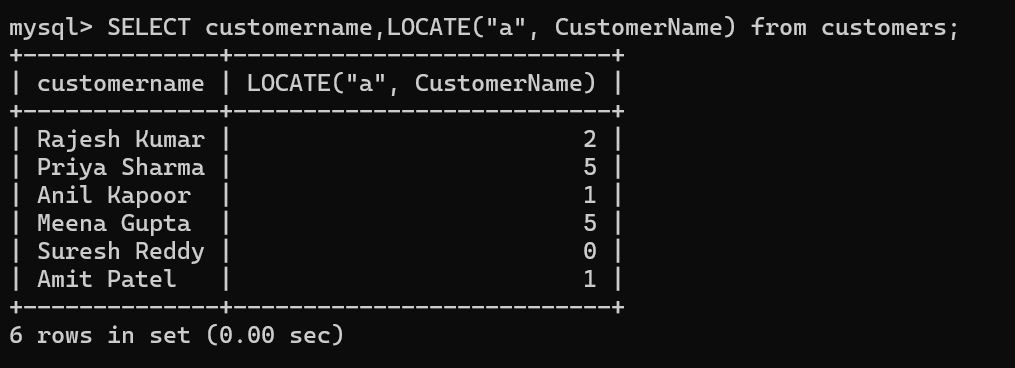
|  |
| --- |
| Returns the length of a string (in bytes) |



MySQL LOCATE () Function

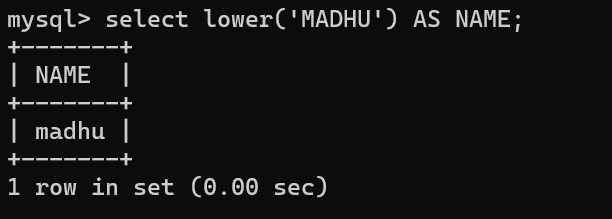
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. The LOCATE () function returns the position of the first occurrence of a substring in a string. 2. If the substring is not found within the original string, this function returns 0. 3. This function performs a case-insensitive search.   Parameter Values   |  |  | | --- | --- | | **Parameter** | **Description** | | *Substring* | Required. The substring to search for in *string* | | *String* | Required. The string that will be searched | | *Start* | Optional. The starting position for the search. Position 1 is default | |





LOWER() Function

|  |
| --- |
| 1. Convert the text to lower-case: 2. This is a more **standardized function** supported by most SQL-based databases like **MySQL, PostgreSQL, Oracle, SQL Server**, etc. |



UPPER() Function

|  |
| --- |
| Convert the text to upper-case: |

UCASE() Function

|  |
| --- |
| Convert the text to upper-case: |

TRIM() Function

|  |
| --- |
| Remove leading and trailing spaces from a string: |

STRCMP() Function

|  |
| --- |
| Compare two strings: |

SUBSTR() Function (OR) SUBSTRING() Function (or) MID() FUNCTION

|  |
| --- |
| SYNTAX:  SUBSTR(*string*, *start*, *length*)  Note:  **Note:** The [SUBSTR()](https://www.w3schools.com/mysql/func_mysql_substr.asp) and [MID()](https://www.w3schools.com/mysql/func_mysql_mid.asp) functions equals to the SUBSTRING() function. |

|  |
| --- |
|  |

LPAD() Function

|  |
| --- |
| Left-pads a string with another string, to a certain length  The below example Left-pad the string with "ABC", to a total length of 20:  It adds the specified characters to the actual string until it becomes 20. |

RPAD() Function

|  |
| --- |
| Right-pad the string with "ABC", to a total length of 20:  It adds the specified characters to the right side of the actual string until it becomes 20. |

MySQL LTRIM() Function

|  |
| --- |
| Remove leading spaces from a string: |

RTRIM() Function

|  |
| --- |
| Remove trailing spaces from a string: |

POSITION() Function (OR) The [LOCATE()](https://www.w3schools.com/mysql/func_mysql_locate.asp) function

|  |
| --- |
| 1. The POSITION() function returns the position of the first occurrence of a substring in a string. 2. If the substring is not found within the original string, this function returns 0. 3. This function performs a case-insensitive search.   Note: The [LOCATE()](https://www.w3schools.com/mysql/func_mysql_locate.asp) function is equal to the POSITION() function.  Example:    Locate function usage |

REPEAT() Function

|  |
| --- |
| Repeats a string as many times as specified |

REPLACE() Function

|  |
| --- |
| The REPLACE() function replaces all occurrences of a substring within a string, with a new substring.  **Note:** This function performs a case-sensitive replacement. |

REVERSE() Function

Reverses a string and returns the result

|  |
| --- |
|  |

SPACE() Function

|  |
| --- |
| Returns a string of the specified number of space characters |

**MySQL Numeric Functions**

ABS() Function: Return the absolute value of a number:

|  |
| --- |
|  |

AVG() Function: Return the average value

|  |
| --- |
|  |

CEIL() (or) CEILING() Function: Returns the smallest integer value that is >= to a number

|  |
| --- |
| (Or) |

FLOOR() Function: Returns the largest integer value that is <= to a number

|  |
| --- |
|  |

COUNT() Function: Return the number of products in the "Products" table:

|  |
| --- |
|  |

DIV Function: Used for integer division

|  |
| --- |
| **Syntax:** *x* DIV *y* |

GREATEST() Function: Return the greatest value of the list of arguments:

|  |
| --- |
|  |

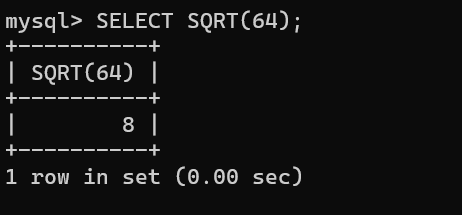
LEAST() Function: Return the smallest value of the list of arguments:

|  |
| --- |
|  |

PI() Function: Return the value of PI:

|  |
| --- |
|  |

SQRT() Function



ROUND() Function

|  |
| --- |
|  |

RAND() Function

|  |
| --- |
|  |

POW() or POWER() Function: Returns the value of a number raised to the power of another number

|  |
| --- |
|  |

MOD() Function

|  |
| --- |
|  |

sum() Function

|  |
| --- |
|  |

min()

|  |
| --- |
|  |

max()

|  |
| --- |
|  |

Example on creating a table using auto\_increment column

|  |
| --- |
|  |

**Date functions**

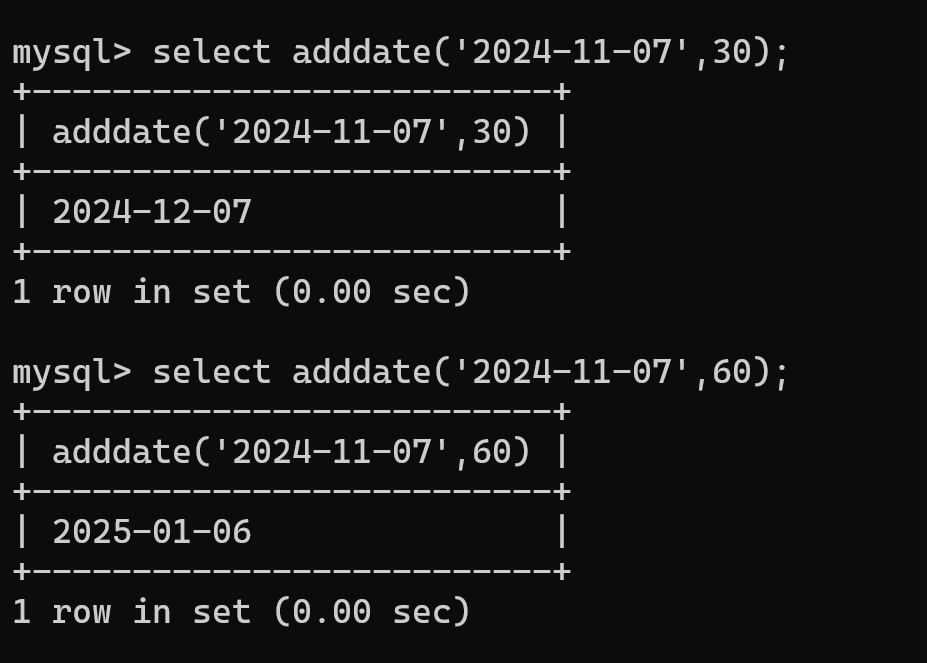
ADDDATE() Function

|  |
| --- |
| The ADDDATE() function adds a time/date interval to a date and then returns the date.  Syntax  ADDDATE(*date*, INTERVAL *value addunit*)  **OR:**  ADDDATE(*date*, *days*) |

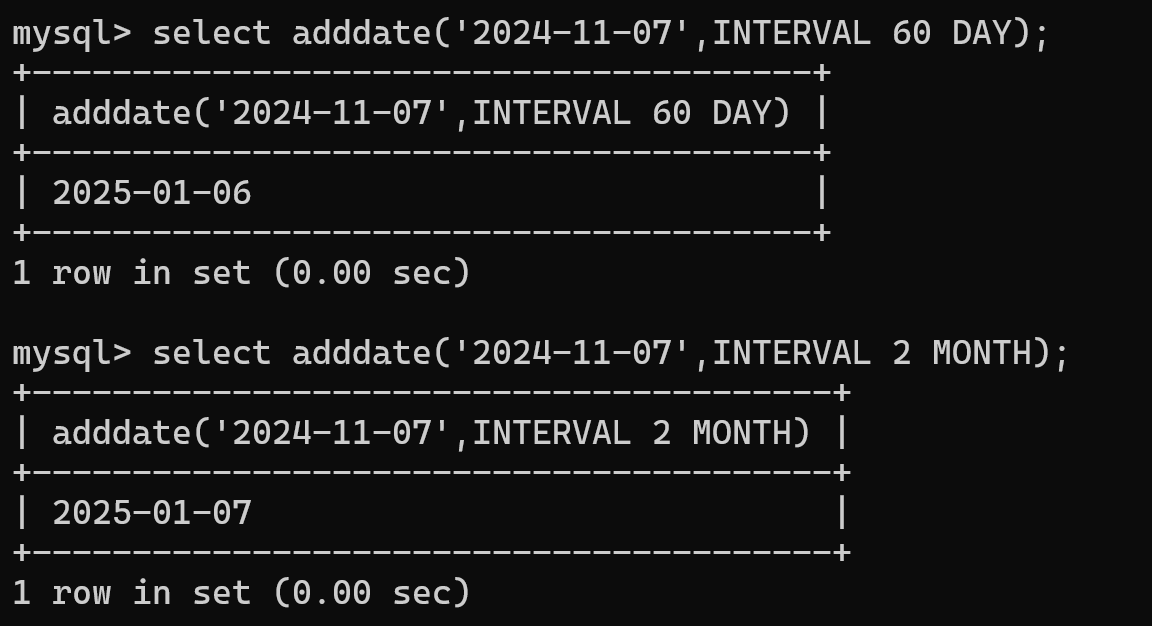
Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *Date* | Required.  The date to be modified |
| *Days* | Required. The number of days to add to *date* |
| *Value* | Required. The value of the time/date interval to add. Both positive and negative values are allowed |
| *Addunit* | Required. The type of interval to add. Can be one of the following values:   * MICROSECOND * SECOND * MINUTE * HOUR * DAY * WEEK * MONTH * QUARTER * YEAR * SECOND\_MICROSECOND * MINUTE\_MICROSECOND * MINUTE\_SECOND * HOUR\_MICROSECOND * HOUR\_SECOND * HOUR\_MINUTE * DAY\_MICROSECOND * DAY\_SECOND * DAY\_MINUTE * DAY\_HOUR * YEAR\_MONTH |

**Example-1**

****

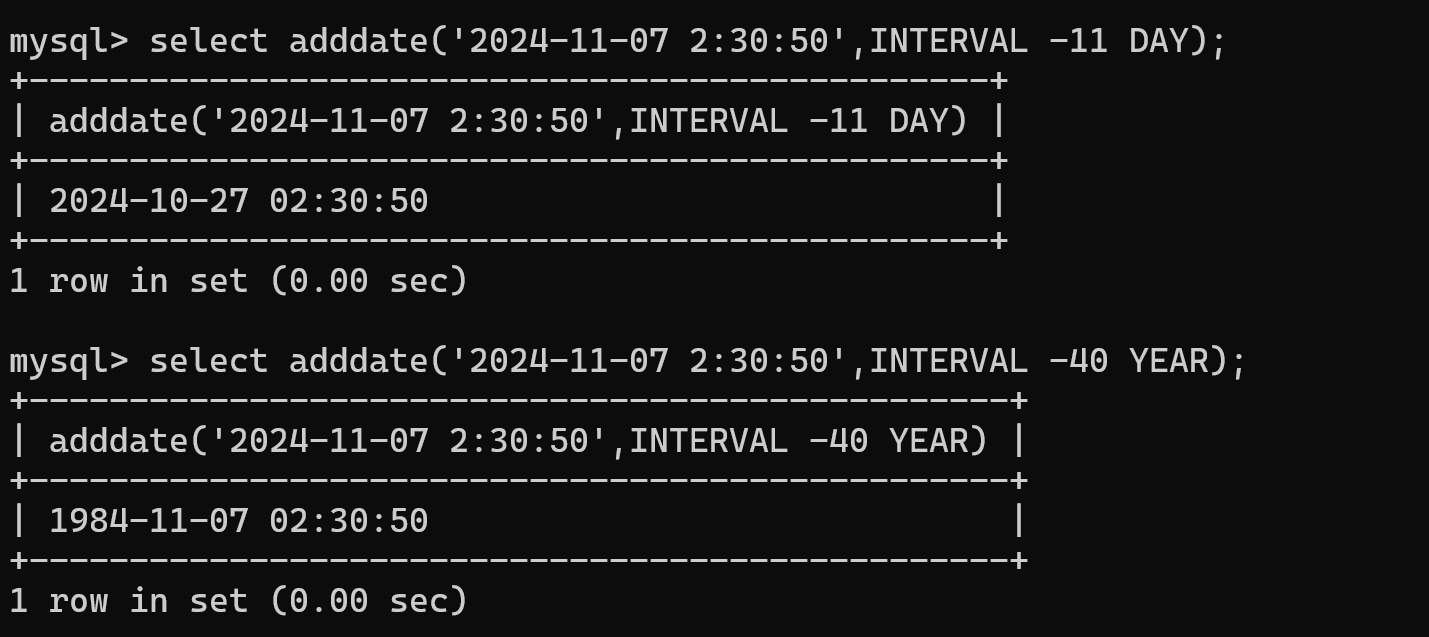
**Example-2**

****

**Example-3**

****

**Example**

****

|  |
| --- |
| SELECT ADDDATE("2017-06-15", INTERVAL 10 DAY);  SELECT ADDDATE("2017-06-15 09:34:21", INTERVAL 15 MINUTE);  SELECT ADDDATE("2017-06-15 09:34:21", INTERVAL -3 HOUR);  SELECT ADDDATE("2017-06-15", INTERVAL -2 MONTH); |

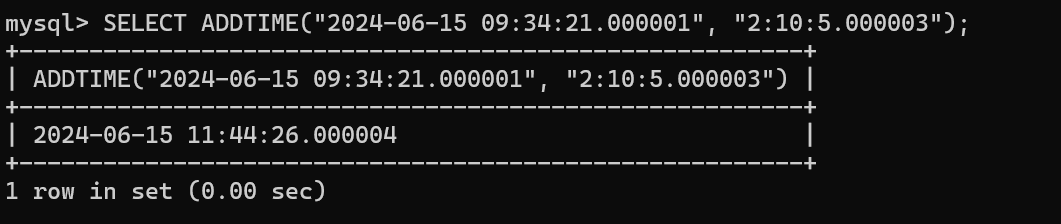
ADDTIME() Function

|  |
| --- |
| SELECT ADDTIME("2017-06-15 09:34:21", "2");  Note: here 2 represents 2 seconds    Another example    Another Example |

Another example

|  |
| --- |
| Adding seconds and microseconds |

Another example



# What is the use of exists and any?

|  |
| --- |
| 1. In MySQL, both **EXISTS** and **ANY** are used in queries involving subqueries, but they are employed for different purposes and have distinct meanings. |

Exists

|  |
| --- |
| 1. **Purpose**: The EXISTS keyword is used to check if a subquery returns any rows. It is often used to test for the existence of records in a related table. 2. **usage**: It returns TRUE if the subquery returns one or more rows, otherwise, it returns FALSE. Typically used in WHERE clauses.   **Syntax:**  SELECT column\_name(s) FROM table\_name WHERE EXISTS (subquery); |

Let's say we have two tables: **Employees** and **Departments**.

**Employees** Table:

|  |  |  |
| --- | --- | --- |
| EmployeeID | EmployeeName | DepartmentID |
| 101 | Alice | 1 |
| 102 | Bob | 2 |

**Departments** Table:

|  |  |
| --- | --- |
| DepartmentID | DepartmentName |
| 1 | HR |
| 2 | IT |
| 3 | Sales |

**Example:**

|  |
| --- |
| SELECT DepartmentName  FROM Departments d  WHERE EXISTS (  SELECT 1  FROM Employees e  WHERE e.DepartmentID = d.DepartmentID  );  The above query returns all departments that have at least one employee. The EXISTS clause checks if the subquery returns any rows (i.e., if there are any employees in the department). |

Another Example on usage of exists()

|  |
| --- |
| Continue |

**2. ANY:**

* **Purpose**: The ANY keyword is used to compare a value to any value in a list or subquery. It is typically used with comparison operators like =, >, <, >=, <=, etc.
* **Usage**: The condition will be true if it matches **any** value in the result set of the subquery.

Syntax:

|  |
| --- |
| SELECT column\_name(s)  FROM table\_name  WHERE column\_name operator ANY (subquery); |

**Example:** Let's say we want to select employees whose salary is greater than the salary of **any** employee in department 2.

|  |
| --- |
| **SELECT EmployeeName, Salary FROM Employees WHERE Salary > ANY ( SELECT Salary FROM Employees WHERE DepartmentID = 2 );** |

**Employees** Table (with Salary):

|  |  |  |  |
| --- | --- | --- | --- |
| EmployeeID | EmployeeName | DepartmentID | Salary |
| 101 | Alice | 1 | 5000 |
| 102 | Bob | 2 | 6000 |
| 103 | Carol | 2 | 7000 |
| 104 | Dave | 1 | 5500 |

Conclusion:

* **EXISTS**: Checks if a subquery returns **any rows** (focus on existence).
* **ANY**: Compares a value to **any value** returned by a subquery (focus on comparison with multiple values).

These are typically used when filtering data based on subqueries that refer to related data in other tables.

**Another Scenario**

**Customers Table:**

|  |  |
| --- | --- |
| CustomerID | CustomerName |
| 1 | **John** |
| 2 | **Jane** |
| 3 | **Mike** |

**Orders Table:**

|  |  |  |
| --- | --- | --- |
| OrderID | CustomerID | Amount |
| 101 | **1** | **500** |
| 102 | **2** | **1500** |
| 103 | **2** | **700** |
| 104 | **3** | **200** |
| 105 | **3** | **600** |

|  |
| --- |
| **Task:** You want to find all customers who have placed an order. In other words, you want to check if any orders exist for each customer.  SELECT CustomerName  FROM Customers c  WHERE EXISTS (  SELECT 1  FROM Orders o  WHERE o.CustomerID = c.CustomerID  );  **Explanation:**   * The subquery checks whether there is at least one order (EXISTS) for each customer. * If there is an order, the main query returns the customer. * In this case, **John**, **Jane**, and **Mike** will be returned because they all have orders in the **Orders** table.   **2. Using ANY:**  **Task**: You want to find all customers who have placed an order with an amount greater than **any order amount** of customer 3 (Mike).  SELECT CustomerName  FROM Customers c  WHERE CustomerID != 3  AND 500 > ANY (  SELECT Amount  FROM Orders  WHERE CustomerID = 3  );  **Explanation:**   * The subquery retrieves all the order amounts for **CustomerID = 3** (Mike), which are 200 (from the **Orders** table). * 500 > ANY means it will check if 500 is greater than any of those amounts (in this case, it is greater than 200). * The query will return **John** and **Jane**, since they are not customer 3, and the condition is met for the amounts.   **Simplified Understanding:**   * **EXISTS**: Is there any record? (Used to check **existence** of rows.) * **ANY**: Compare with **any** value in a list (Used for **comparison** with multiple values from a subquery). |

What are **correlated subqueries?**

|  |
| --- |
| A **correlated subquery** is an SQL query that uses values from the outer query to evaluate its conditions.  Unlike a regular subquery, which is independent and can be executed once before the main query, a correlated subquery is executed repeatedly, once for each row processed by the outer query. This relationship between the inner and outer queries is what gives the correlated subquery its name.  The below example is on regular subquery |

**Example on correlated sub-Query**

**Students** Table:

|  |  |  |  |
| --- | --- | --- | --- |
| StudentID | StudentName | ClassID | Marks |
| 1 | John | 101 | 85 |
| 2 | Jane | 102 | 92 |
| 3 | Bob | 101 | 75 |
| 4 | Alice | 102 | 80 |

**Classes Table:**

|  |  |
| --- | --- |
| ClassID | ClassName |
| 101 | Maths |
| 102 | Science |

**Commands to create the above tables**

|  |
| --- |
| create table classes(  class\_id int,  class\_name varchar(15)unique not null,  constraint pk\_class\_id primary key(class\_id)  );  **Inserting data into table**  insert into classes values(101,'Maths'),  (102,'Science');  create table students(  student\_id int,  student\_name varchar(25)not null,  class\_id int,  marks int,  constraint pk\_student\_id primary key(student\_id),  constraint fk\_class\_id foreign key(class\_id) references classes(class\_id)  );  **Inserting data to students table**  insert into students values(1,'John',101,85),  (2,'Jane',102,92),  (3,'Bob',101,75),  (4,'Alice',102,80); |

**Task:**

|  |
| --- |
| We want to find students who have marks **higher than the average marks of their class**. |

**Correlated Subquery Example**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SELECT StudentName, Marks  FROM Students s  WHERE Marks > (  SELECT AVG(Marks)  FROM Students  WHERE ClassID = s.ClassID  );  **Explanation:**   1. **Outer Query**:    * The outer query is selecting the StudentName and Marks from the **Students** table. 2. **Subquery** (Correlated Subquery):    * The subquery calculates the **average marks** for each class. The important part here is that the subquery refers to the **ClassID** of the current student (s.ClassID) from the outer query.    * This means that for each row in the **outer query** (for each student), the subquery will calculate the average marks of the class that student belongs to. 3. **Row-by-row Evaluation**:  * For **John** (Class 101): The subquery will calculate the average marks in Class 101 as (85 + 75) / 2 = 80. Since John’s marks (85) are greater than 80, he will be included in the result. * For **Jane** (Class 102): The subquery will calculate the average marks in Class 102 as (92 + 80) / 2 = 86. Jane's marks (92) are higher than 86, so she will be included. * For **Bob** (Class 101): The subquery calculates the same average for Class 101 (80). Since Bob's marks (75) are less than 80, he will **not** be included. * For **Alice** (Class 102): Alice's marks (80) are less than the average of her class (86), so she will **not** be included.   **Result:**   | **StudentName** | **Marks** | | --- | --- | | John | 85 | | Jane | 92 |   **In Simple Terms:**   * The **outer query** runs to fetch students' names and marks. * For each student, the **correlated subquery** calculates the average marks for that student’s class. * Then the outer query checks if the student’s marks are higher than the average marks of their class.   This way, the subquery "correlates" with each row in the outer query, making it a **correlated subquery**. |

# Introduction of ER Model

The Entity Relational Model is a model for identifying entities to be represented in the database and representation of how those entities are related.

The Entity Relationship Diagram explains the relationship among the entities present in the database.

## Symbols Used in ER Model

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Rectangles** |  | Rectangles represent Entities in the ER Model. |
| **Double Rectangle** |  | Double Rectangle represents a Weak Entity. |
| **Ellipses** |  | Ellipses represent Attributes in the ER Model. |
| **Dotted ellipse** |  | **Derived Attribute** |
| **Ellipse with sub ellipse** |  | **Composite Attribute** |
| **Double Ellipse** |  | Double Ellipses represent Multi-Valued Attributes. |
| **Diamond** |  | Diamonds represent Relationships among Entities. |
| **Double diamond** |  | Weak Relationship |
| **Lines** |  | Lines represent attributes to entities and entity sets with other relationship types |

Examples of derived attributes

|  |
| --- |
| 1. Age : we can get it by using dob 2. Discount amount : we can get it by using mrp and discount percentage 3. Distance we can get It by using speed and time 4. Experience: we can get it by using doj |

## Components of ER Diagram

## 

**What is weak entity?**

|  |
| --- |
| A weak entity in a relational database is an entity that cannot be uniquely identified by its own attributes alone. It depends on a "strong entity" (or parent entity) and uses a foreign key relationship combined with some partial key to form a primary key. |

**Let's take an example where we have two entities:**

1. Department (a strong entity)
2. Employee(a strong entity)
3. EmployeeDependents (a weak entity).

**Example:**

1. **Department**: This is a strong entity where each department has a unique DepartmentID.
2. **EmployeeDependents**: This is a weak entity because a dependent (like a child or spouse) cannot be identified without knowing which employee they are related to
3. Thus, we need both the EmployeeID from the employee and a dependent's information (like DependentName).

**Department Table (Strong Entity)**

|  |  |
| --- | --- |
| DepartmentID | DepartmentName |
| 1 | **HR** |
| 2 | **IT** |
| 3 | **Sales** |

**Employee Table (Strong Entity)**

|  |  |  |
| --- | --- | --- |
| EmployeeID | EmployeeName | DepartmentID |
| 101 | **Alice** | **1** |
| 102 | **Bob** | **2** |
| 103 | **Carol** | **3** |

**EmployeeDependents Table (Weak Entity)**

|  |  |  |  |
| --- | --- | --- | --- |
| EmployeeID | DependentName | Relation | DependentID (PK) |
| 101 | **Sam** | **Son** | **1** |
| 101 | **Jane** | **Daughter** | **2** |
| 102 | **Tom** | **Spouse** | **1** |

**Explanation:**

* Department: Each department has a unique DepartmentID.
* Employee: Each employee belongs to a department and is identified by a unique EmployeeID.
* EmployeeDependents: This weak entity cannot be uniquely identified by just the DependentName. It needs to combine the EmployeeID (from the strong entity Employee) and an additional attribute (like DependentID or DependentName) to form a unique identifier for each dependent.

**In the EmployeeDependents table:**

* EmployeeID is a foreign key linking to the Employee table.
* The primary key is a combination of EmployeeID and DependentID, where DependentID is a partial key of the weak entity.

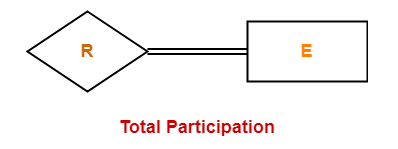
**Thus, the EmployeeDependents entity is weak because its existence is dependent on the Employee entity.**

**Tables**

**How we can identify total participation in e-r diagram?**

**1. Total Participation-**

1. It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set.That is why, it is also called as mandatory participation.
2. Total participation is represented using a double line between the entity set and relationship set.



Example-

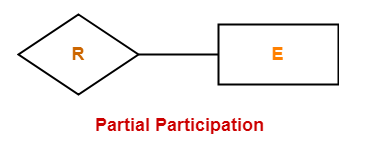


Here,

Double line between the entity set “Student” and relationship set “Enrolled in” signifies total participation. It specifies that each student must be enrolled in at least one course.

**2. Partial Participation-**

1. It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set. That is why, it is also called as optional participation.
2. Partial participation is represented using a single line between the entity set and relationship set.



Example-



Here,

Single line between the entity set “Course” and relationship set “Enrolled in” signifies partial participation. It specifies that there might exist some courses for which no enrollments are made.

