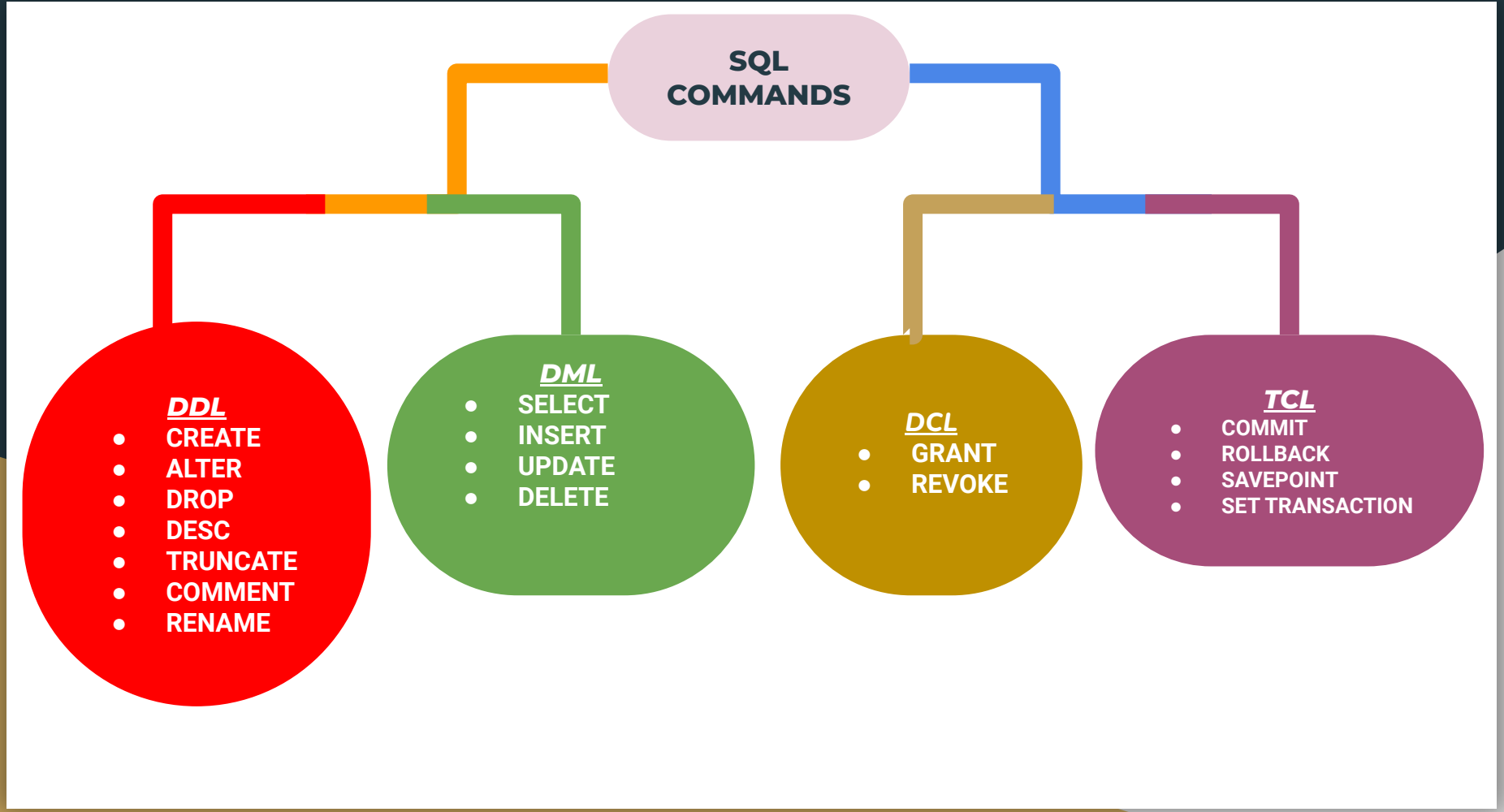
RAILWAY MANAGEMENT SYSTEM

Submitted To:-
DR.Mamta Juneja
Associate Professor ,CSE Department
UIET, Panjab University Chandigarh (160014)

Submitted By:-

Archit	UE203020
Arpit	UE203022
Aryan	UE203025
Ashish	UE203026
Davy	UE203033
Gajendra	UE203038
Gurdeep	UE203042
Karan	UE203054


SQL COMMANDS
DDL
CREATE
ALTER
DROP
DESC
TRUNCATE
COMMENT
RENAME
DML
SELECT
INSERT
UPDATE
DELETE
DCL
GRANT
REVOKE
TCL
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

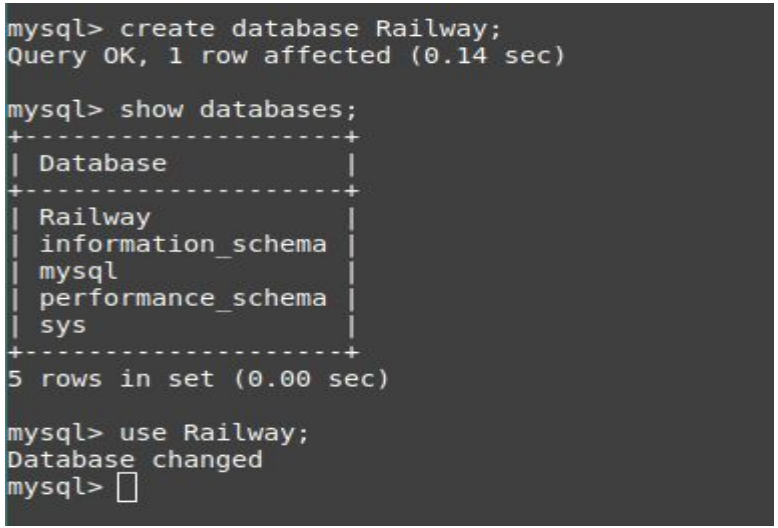
DDL(DATA DEFINITION LANGUAGE)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.
- It deals with database schemas and descriptions, of how the data should reside in the database.

CREATE COMMAND

To create a database and its objects like (table, index, views, store procedure, function, and triggers).

1. **To create new database –**



2. To create new table in a database

```
CREATE TABLE Passenger(
    P_ID INT PRIMARY KEY,
    P_NAME VARCHAR(50),
    SEAT_NO INT,
    GENDER VARCHAR(1),
    PH_NUMBER INT
);
```

Introducing our all tables

Passenger_Table

```
mysql> DESC PASSENGER;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| P_ID      | int         | NO   | PRI | NULL    |                |
| P_NAME    | varchar(50) | YES  |     | NULL    |                |
| SEAT_NO   | int         | YES  |     | NULL    |                |
| GENDER    | varchar(1)  | YES  |     | NULL    |                |
| PH_NUMBER | int         | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

Station_Table

```
mysql> DESC Station;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| STATION_ID      | int         | NO   | PRI | NULL    |       |
| STATION_NAME    | varchar(20) | NO   |     | NULL    |       |
| NO_OF_LINES     | int         | NO   |     | NULL    |       |
| NO_OF_PLATFORMS | int         | NO   |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
```

Train_Table

```
mysql> DESC TRAIN;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| TRAIN_ID   | int         | NO   | PRI | NULL    |       |
| TRAIN_NAME | varchar(20) | NO   | UNI | NULL    |       |
| STATION_ID | int         | NO   | UNI | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

Ticket_Table

```
mysql> DESC TICKET;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Ticket_No   | int         | NO   | PRI | NULL    |       |
| P_ID        | int         | YES  | MUL | NULL    |       |
| SOURCE      | varchar(20) | NO   |     | NULL    |       |
| DESTINATION | varchar(20) | NO   |     | NULL    |       |
| TRAIN_ID    | int         | YES  | MUL | NULL    |       |
| RES_STATUS  | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

FARE_TABLE

```
mysql> DESC FARE_TABLE;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| CLASS     | varchar(20) | YES  |     | NULL    |       |
| Fare      | int         | YES  |     | NULL    |       |
| Ticket_No | int         | NO   | PRI | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

RESERVE TABLE

Reserve Table: It used to sketches all reserved ticket with their required attributes like status of booked ticket , date of booking , no of seats, address of passenger, and their contact details.

ATTRIBUTE

1. PNR_No : Foreign Key from Reserve Relation , referencing Passenger Relation.
2. Journey_date
3. No_of_seats: Tells the no of seats booked by Passenger
4. Address
5. Contact_No
6. Status : Shows status whether ticket is reserved or not.

3.. **Reserve** : PNR_No (PK & FK -> Passenger), Journey_date,No_of_seats, Address, Contact_No, Status

```
CREATE TABLE Reserve(
PNR_No NUMERIC(9),
foreign key(PNR_NO) references passenger(PNR_NO),
Journey_date date,
No_of_seats Numeric(8),
Address Varchar(40),
Contact_No NUMERIC(10),
Status Char(2));
```

Cancellation: PNR_No(Foreign Key), Journey_date, No_of_seats, Address, Contact_No,Status

```
CREATE TABLE Cancellation(
PNR_No NUMERIC(9), foreign
key(PNR_NO) references passenger(PNR_NO),
Journey_date date,No_of_seats
NUMERIC(8),
Address Varchar(40),
Contact_No NUMERIC(10),
Status char(2));
```

CANCELLATION TABLE

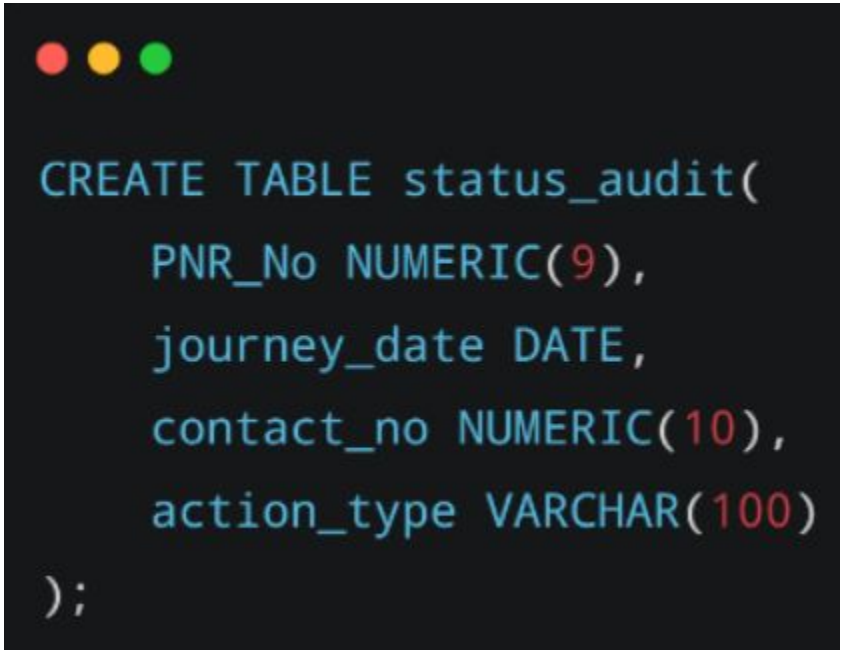
Cancellation Table: It used to sketches all reserved ticket with their required attributes like status of booked ticket , date of booking , no of seats, and their contact details.

ATTRIBUTE

1. PNR_NO : Passenger Name Record. FK references Passenger Table
2. Journey_Date :
3. No_of_seats : Explain This
4. Contact_No :
5. Status : Cancellation status of ticket

Status_Audit Table

It keeps track status of tickets more briefly what happens to tickets and such stuffs and helped us in **Triggers** part of this project.



```
CREATE TABLE status_audit(
    PNR_No NUMERIC(9),
    journey_date DATE,
    contact_no NUMERIC(10),
    action_type VARCHAR(100)
);
```

RENAME COMMAND

Sometimes we may want to rename our table to give it a more relevant name.

Syntax –

rename table <current_table_name> to <new_table_name>;

```
RENAME TABLE Passenger to Passenger_Table;
RENAME TABLE Passenger_Table to Passenger
```

DESCRIBE COMMAND

DESCRIBE is used to describe something. Since in database we have tables, that's why we use DESCRIBE or DESC(both are same) command to describe the structure of a table(like column headings, datatypes).

```
DESC PASSENGER;
```

ALTER COMMAND

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

The `ALTER TABLE` statement is also used to add and drop various constraints on an existing table.

1. **ALTER TABLE - ADD Column**

   ADD is used to add columns into the existing table. Sometimes we may require to add additional information, in that case we do not require to create the whole database again, ADD comes to our rescue.

```
mysql> ALTER TABLE Passenger  ADD New_column INT;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> DESC PASSENGER;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| P_ID       | int         | NO   | PRI | NULL    | auto_increment |
| P_NAME     | varchar(50) | YES  |     | NULL    |                |
| SEAT_NO    | int         | YES  |     | NULL    |                |
| GENDER     | varchar(1)  | YES  |     | NULL    |                |
| PH_NUMBER  | int         | YES  |     | NULL    |                |
| New_column | int         | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

As we can see in output new column added in table

2. ALTER TABLE - MODIFY COLUMN

It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

```
mysql> ALTER TABLE Passenger MODIFY COLUMN New_column VARCHAR(20);
Query OK, 17 rows affected (0.08 sec)
Records: 17  Duplicates: 0  Warnings: 0
```

```
mysql> DESC PASSENGER;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| P_ID       | int         | NO   | PRI | NULL    | auto_increment |
| P_NAME     | varchar(50) | YES  |     | NULL    |                |
| SEAT_NO    | int         | YES  |     | NULL    |                |
| GENDER     | varchar(1)  | YES  |     | NULL    |                |
| PH_NUMBER  | int         | YES  |     | NULL    |                |
| New_column | varchar(20) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

See finally we have changed datatype of New_column

3. ALTER TABLE - DROP COLUMN

DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

```
mysql> ALTER TABLE Passenger DROP New_column;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

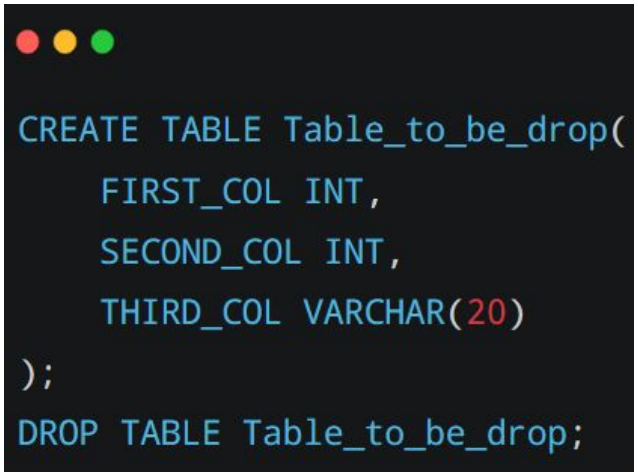
```
mysql> DESC PASSENGER;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| P_ID      | int         | NO   | PRI | NULL    | auto_increment |
| P_NAME    | varchar(50) | YES  |     | NULL    |                |
| SEAT_NO   | int         | YES  |     | NULL    |                |
| GENDER    | varchar(1)  | YES  |     | NULL    |                |
| PH_NUMBER | int         | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

As we can see in table our previous col New_column deleted successfully

DROP COMMAND

DROP is used to delete a whole database or just a table.The DROP statement destroys the objects like an existing database, table, index, or view.

A DROP statement in SQL removes a component from a relational database management system (RDBMS).



```
CREATE TABLE Table_to_be_drop(
    FIRST_COL INT,
    SECOND_COL INT,
    THIRD_COL VARCHAR(20)
);
DROP TABLE Table_to_be_drop;
```

TRUNCATE COMMAND

TRUNCATE statement is a Data Definition Language (DDL) operation that is used to mark the extents of a table for deallocation (empty for reuse). The result of this operation quickly removes all data from a table, typically bypassing a number of integrity enforcing mechanisms.

```
CREATE TABLE Table_to_be_drop(
    FIRST_COL INT,
    SECOND_COL INT,
    THIRD_COL VARCHAR(20)
);
TRUNCATE Table_to_be_drop;
```

DROP VS TRUNCATE COMMAND

- Truncate is normally ultra-fast and its ideal for deleting data from a temporary table.
- Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.
- Table or Database deletion using DROP statement cannot be rolled back, so it must be used wisely.

DML (DATA MANIPULATION LANGUAGE)

The DML commands change the data present in the SQL database. We can easily access, store, modify, update and delete the existing records from the database using DML commands.

Following are the four main DML commands in SQL:

1. **SELECT Command**
2. **INSERT Command**
3. **UPDATE Command**
4. **DELETE Command**

Insert command

```
-- INSERTION commands to query to add passengers into passenger table

INSERT INTO Passenger VALUES('Camila
Cabello',8,'F',600034342);
```

Insert Data Only in Specified Columns

```
-- inserting to only specific column
INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER) VALUES('Jennifer Lawrence',9,'F');
```

Insert Multiple Rows

```
-- INSERTING MULTIPLE COLUMNS
INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER,PH_NUMBER)
VALUES('Jackline fernandez',20,'F',848777888),
('Amar Pal',7,'M',117567354),
('Johnson KUMAR',19,'M',777567354),
('Archit Sharma',61,'M',848567341),
('David Bansal',41,'M',644534342),
('GB Singh',51,'M',983135565);
```

Create table from another table and limit clause

```
mysql> create table Passenger_copy as  select* from Passenger;
Query OK, 30 rows affected (1.18 sec)
Records: 30  Duplicates: 0  Warnings: 0

mysql> select* from Passenger_copy LIMIT 5;
+------+------------------+---------+--------+-----------+
| P_ID | P_NAME           | SEAT_NO | GENDER | PH_NUMBER |
+------+------------------+---------+--------+-----------+
|    1 | Camila Cabello   |       8 | F      | 600034342 |
|    2 | Jennifer Lawrence|       9 | F      | 113135265 |
|    3 | Jennifer Lopez   |      10 | F      | 848567888 |
|    4 | Charlie Puth     |      11 | M      | 777567354 |
|    5 | Davy Bansal      |       4 | M      | 644534342 |
+------+------------------+---------+--------+-----------+
5 rows in set (0.00 sec)
```

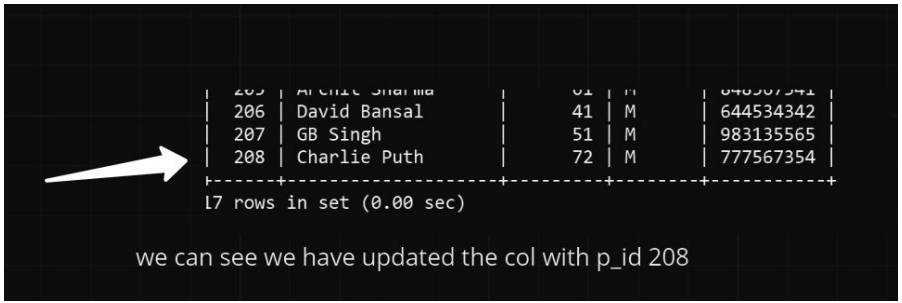
UPDATE COMMAND

The `UPDATE` statement is used to modify the existing records in a table.

The `WHERE` clause specifies which record(s) that should be updated. If we omit the `WHERE` clause, all records in the table will be updated!

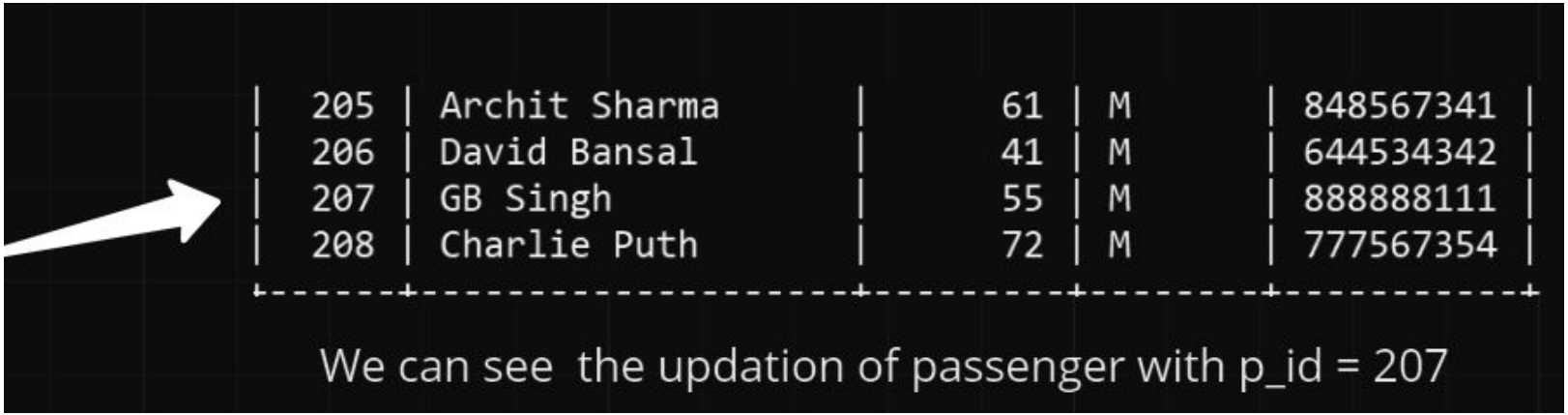
1. Updating single column

```
mysql> UPDATE PASSENGER SET SEAT_NO = 72 WHERE P_ID = 208;
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```



2. **Updating multiple columns**

```
mysql> UPDATE PASSENGER SET PH_NUMBER = 888888111, SEAT_NO = 55 WHERE P_ID = 207;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```



205	Archit Sharma	61	M	848567341
206	David Bansal	41	M	644534342
207	GB Singh	55	M	888888111
208	Charlie Puth	72	M	777567354

We can see the updation of passenger with p_id = 207

Select commands for all rows and all cols

```
mysql> SELECT * FROM PASSENGER
    -> ;
+------+--------------------+---------+--------+-----------+
| P_ID | P_NAME             | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------------+---------+--------+-----------+
|    1 | Camila Cabello     |       8 | F      | 600034342 |
|    2 | Jennifer Lawrence  |       9 | F      | 113135265 |
|    3 | Jennifer Lopez     |      10 | F      | 848567888 |
|    4 | Charlie Puth       |      11 | M      | 777567354 |
|    5 | Davy Bansal        |       4 | M      | 644534342 |
|    6 | Gurdeep Singh      |       5 | M      | 983135265 |
|    7 | Aryan Sharma       |       6 | M      | 848567341 |
|    8 | Gajendra Pal       |       7 | M      | 747567354 |
|  200 | Shabby Cabello     |      81 | F      | 600011139 |
|  201 | Jennifer Lawrence  |       9 | F      |      NULL |
|  202 | Jackline fernandez |      20 | F      | 848777888 |
```

Select commands for all rows and specific cols

```
SELECT P_ID AS ID , P_NAME AS NAME, SEAT_NO AS seat FROM Passenger;
```

```
mysql> SELECT P_ID AS ID , P_NAME AS NAME, SEAT_NO AS seat FROM Passenger;
+-----+--------------------+------+
| ID  | NAME               | seat |
+-----+--------------------+------+
|   1 | Camila Cabello     |    8 |
|   2 | Jennifer Lawrence  |    9 |
|   3 | Jennifer Lopez     |   10 |
|   4 | Charlie Puth       |   11 |
|   5 | Davy Bansal        |    4 |
|   6 | Gurdeep Singh      |    5 |
|   7 | Aryan Sharma       |    6 |
|   8 | Gajendra Pal       |    7 |
| 200 | Shabby Cabello     |   81 |
```

Select command for all rows and single col

```
mysql> SELECT P_NAME AS name_of_passenger FROM PASSENGER;
+--------------------+
| name_of_passenger  |
+--------------------+
| Camila Cabello     |
| Jennifer Lawrence  |
| Jennifer Lopez     |
| Charlie Puth       |
| Davy Bansal        |
| Gurdeep Singh      |
| Aryan Sharma       |
| Gajendra Pal       |
| Shabby Cabello     |
| Jennifer Lawrence  |
| Jackline fernandez |
| Amar Pal           |
```

Select Commands with where clause

```
mysql> SELECT P_NAME AS name FROM PASSENGER  WHERE GENDER = 'F';
+--------------------+
| name               |
+--------------------+
| Camila Cabello     |
| Jennifer Lawrence  |
| Jennifer Lopez     |
| Shabby Cabello     |
| Jennifer Lawrence  |
| Jackline fernandez |
+--------------------+
```

Select Commands With All Columns And Specific Rows

```
mysql> SELECT *
    -> FROM station
    -> WHERE NO_OF_PLATFORMS>4;
+------------+--------------+-------------+-----------------+
| STATION_ID | STATION_NAME | NO_OF_LINES | NO_OF_PLATFORMS |
+------------+--------------+-------------+-----------------+
|      10045 | Pune         |          30 |               7 |
|      11898 | Thane        |          35 |               8 |
|      19996 | Chennai      |          32 |               6 |
+------------+--------------+-------------+-----------------+
```

Select Commands With DISTINCT keyword

With this DISTINCT keyword we can select only distinct rows from a particular column.

```
SELECT DISTINCT Gender FROM Passenger;
```

Success
(2 rows) 0.2 s

Gender
M
F

DCL(DATA CONTROL LANGUAGE)

- **DCL COMMANDS ARE USED TO CONTROL PRIVILEGE IN THE DATABASE.**
- **DCL COMMANDS ARE USED TO PERFORM WORK RELATED TO RIGHTS , PERMISSIONS IN THE DATABASE.**
- **THE TWO MOST IMPORTANT DCL COMMANDS ARE:**
- **1) GRANT -This command is used to grant permission to the user to perform a particular operation on a particular object.**
- **2)REVOKE - This command is used to take permission/access back from the user. If you want to return permission from the database that you have granted to the users at that time you need to run REVOKE command.**

ADVANTAGES OF DCL COMMANDS

1) It allows to restrict the unauthorized users from accessing data in database.
2) It ensures security in database when the data is exposed to multiple users.
3) It is the wholesome responsibility of the data owner or data administrator to maintain the authority of grant and revoke privileges to the users preventing any threat to data.
4) It prevents other users to make changes in database who have no access to Database

GRANT

➢ **It is employed to grant a privilege to a user. GRANT command allows specified users to perform specified tasks**

Syntax

```
GRANT privilege_name on objectname to user;
```

Here,

- privilege names are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL
- objectname is table name
- user is the name of the user to whom we grant privileges

CREATING USERS IN MYSQL

- **Different Users with different permissions can be created in the database to enhance security and effectiveness within the database.**

Syntax

```
create user USER_NAME identified by PASSWORD;
```

```
#Creating ADMIN ROLE IN THE DATABASE;
```

```
(NOTE: admin role is not DBA here)
```

```
mysql> create user admin identified by 'admin@123';
Query OK, 0 rows affected (0.10 sec)

mysql> show grants for admin;
+-----------------------------------+
| Grants for admin@%                |
+-----------------------------------+
| GRANT USAGE ON *.* TO `admin`@`%` |
+-----------------------------------+
1 row in set (0.00 sec)
```

```
mysql> grant select on Passenger to 'admin';
Query OK, 0 rows affected (0.10 sec)

mysql> ^DBye
root@error01800180-Laptop:/home/error01800180/Downloads# mysql -u admin  -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> status;
--------------
mysql  Ver 8.0.27-0ubuntu0.20.04.1 for Linux on x86_64 ((Ubuntu))

Connection id:          16
Current database:
Current user:           admin@localhost
SSL:                    Not in use
Current pager:          stdout
Using outfile:          ''
Using delimiter:        ;
Server version:         8.0.27-0ubuntu0.20.04.1 (Ubuntu)
Protocol version:       10
Connection:             Localhost via UNIX socket
Server characterset:    utf8mb4
Db     characterset:    utf8mb4
Client characterset:    utf8mb4
Conn.  characterset:    utf8mb4
UNIX socket:            /var/run/mysqld/mysqld.sock
Binary data as:         Hexadecimal
Uptime:                 3 hours 14 min 43 sec

Threads: 2  Questions: 171  Slow queries: 0  Opens: 295  Flush tables: 3  Open tables: 209  Queries per second avg: 0.014
--------------
```

granting select privilege to admin on passenger table

Switched to user admin

Current User is Admin

`mysql>Show grants for admin;` ->Shows all grants for the user admin.

We can notice that till this stage admin only has access to select statement.

```
mysql> show grants for admin;
+-----------------------------------------------------+
| Grants for admin@%                                  |
+-----------------------------------------------------+
| GRANT USAGE ON *.* TO `admin`@`%`                   |
| GRANT SELECT ON `Railway`.`Passenger` TO `admin`@`%` |
+-----------------------------------------------------+
2 rows in set (0.01 sec)

mysql> use Railway;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from Passenger;
+------+---------------+---------+--------+-----------+
| P_ID | P_NAME        | SEAT_NO | GENDER | PH_NUMBER |
+------+---------------+---------+--------+-----------+
|    1 | Davy Bansal   |       4 | M      | 644534342 |
|    2 | Gurdeep Singh |       5 | M      | 983135265 |
|    3 | Aryan Sharma  |       6 | M      | 848567341 |
|    4 | Gajendra Pal  |       7 | M      | 747567354 |
+------+---------------+---------+--------+-----------+
4 rows in set (0.00 sec)

mysql> INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES('Aman',8,'M',747567254);
ERROR 1142 (42000): INSERT command denied to user 'admin'@'localhost' for table 'Passenger'
mysql>
```

Granting insert,alter,delete,update permissions on table Passenger to admin role.

```
root@error01800180-Laptop:/home/error01800180/Downloads# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> grant select,insert,alter,delete,update  on Passenger To admin;
ERROR 1046 (3D000): No database selected
mysql> use Railway;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> grant select,insert,alter,delete,update  on Passenger To admin;
Query OK, 0 rows affected (0.11 sec)
```

Grants given to admin And now admin can insert into the Passenger table;

```
root@error01800180-Laptop:/home/error01800180/Downloads# mysql -u admin  -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show grants
    -> ;
+-----------------------------------------------------------------------------------------+
| Grants for admin@%                                                                      |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `admin`@`%`                                                       |
| GRANT SELECT, INSERT, UPDATE, DELETE, ALTER ON `Railway`.`Passenger` TO `admin`@`%` |
+-----------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES('Aman',8,'M',747567254);
ERROR 1046 (3D000): No database selected
mysql> use Railway;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES('Aman',8,'M',747567254);
Query OK, 1 row affected (0.15 sec)

mysql> select * from Passenger;
+------+---------------+---------+--------+-----------+
| P_ID | P_NAME        | SEAT_NO | GENDER | PH_NUMBER |
+------+---------------+---------+--------+-----------+
|    1 | Davy Bansal   |       4 | M      | 644534342 |
|    2 | Gurdeep Singh |       5 | M      | 983135265 |
|    3 | Aryan Sharma  |       6 | M      | 848567341 |
|    4 | Gajendra Pal  |       7 | M      | 747567354 |
|    5 | Aman          |       8 | M      | 747567254 |
+------+---------------+---------+--------+-----------+
```

Granting insert,alter,delete,update permissions on different tables to admin role.

```
mysql> grant select,insert,alter,delete,update  on Fare_table To admin;
Query OK, 0 rows affected (0.09 sec)

mysql> grant select,insert,alter,delete,update  on Station To admin;
Query OK, 0 rows affected (0.14 sec)

mysql> grant select,insert,alter,delete,update  on Train To admin;
Query OK, 0 rows affected (0.24 sec)

mysql> grant select,insert,alter,delete,update  on Route  To admin;
Query OK, 0 rows affected (0.10 sec)

mysql> grant select,insert,alter,delete,update  on Ticket To admin;
Query OK, 0 rows affected (0.10 sec)
```

Revoking Permissions from users

```
mysql> savepoint user_created;
Query OK, 0 rows affected (0.00 sec)

mysql> create user user1 identified by 'user1@123';
Query OK, 0 rows affected (0.09 sec)

mysql> use Railway;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> grant select on Fare_table to user1;
Query OK, 0 rows affected (0.16 sec)

mysql> grant select on Station to user1;
Query OK, 0 rows affected (0.12 sec)

mysql> grant select on Train  to user1;
Query OK, 0 rows affected (0.11 sec)
```

```
mysql> grant select on Passenger to user1;
Query OK, 0 rows affected (0.14 sec)

mysql> Revoke select on Passenger from  user1;
Query OK, 0 rows affected (0.08 sec)
```

Revoke:

Syntax:

`REVOKE privilege_name on objectname from user;`

user1 further creating another user2

We can achieve this by creating a user with the same privilege as the root user.

```
mysql> grant all on *.*  to 'user1';
Query OK, 0 rows affected (0.03 sec)

mysql> ^DBye
root@error01800180-Laptop:/home/error01800180# mysql -u user1 -p Railway
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 29
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create user user2 identified by 'user2@123';
Query OK, 0 rows affected (0.08 sec)
```

TCL(TRANSACTION CONTROL LANGUAGE)

- **TCL manages transactions within the database.**
- **SOME OF THE TCL COMMANDS ARE:**
- **1)SAVEPOINT - The changes done till savpoint will be unchanged and all the transactions after savepoint will be rolled back.**
- **2)ROLLBACK - It is used to restore the database to that state which was last committed.**
- **3) COMMIT-It is used to save the transactions in the database.**

SAVPOINT AND ROLLBACK

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select* from Passenger;
+------+---------------+---------+--------+-----------+
| P_ID | P_NAME        | SEAT_NO | GENDER | PH_NUMBER |
+------+---------------+---------+--------+-----------+
|    1 | Davy Bansal   |       4 | M      | 644534342 |
|    2 | Gurdeep Singh |       5 | M      | 983135265 |
|    3 | Aryan Sharma  |       6 | M      | 848567341 |
|    4 | Gajendra Pal  |       7 | M      | 747567354 |
|    5 | Aman          |       8 | M      | 747567254 |
+------+---------------+---------+--------+-----------+
5 rows in set (0.00 sec)

mysql> savepoint s1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES('Samar Singh',9,'M',983131265);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select* from Passenger;
+------+--------------+---------+--------+-----------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------+---------+--------+-----------+
|    1 | Davy Bansal  |       4 | M      | 644534342 |
|    2 | Gurdeep Singh|       5 | M      | 983135265 |
|    3 | Aryan Sharma |       6 | M      | 848567341 |
|    4 | Gajendra Pal |       7 | M      | 747567354 |
|    5 | Aman         |       8 | M      | 747567254 |
|    6 | Samar Singh  |       9 | M      | 983131265 |
+------+--------------+---------+--------+-----------+
6 rows in set (0.00 sec)

mysql> rollback to s1;
Query OK, 0 rows affected (0.02 sec)

mysql> select* from Passenger;
+------+--------------+---------+--------+-----------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------+---------+--------+-----------+
|    1 | Davy Bansal  |       4 | M      | 644534342 |
|    2 | Gurdeep Singh|       5 | M      | 983135265 |
|    3 | Aryan Sharma |       6 | M      | 848567341 |
|    4 | Gajendra Pal |       7 | M      | 747567354 |
|    5 | Aman         |       8 | M      | 747567254 |
+------+--------------+---------+--------+-----------+
5 rows in set (0.00 sec)
```

Rolling back to the savepoint.

```
mysql> INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES('Samar Singh',9,'M',983131265);
Query OK, 1 row affected (0.38 sec)

mysql> select* from Passenger;
+------+--------------+---------+--------+-----------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------+---------+--------+-----------+
|    1 | Davy Bansal  |       4 | M      | 644534342 |
|    2 | Gurdeep Singh|       5 | M      | 983135265 |
|    3 | Aryan Sharma |       6 | M      | 848567341 |
|    4 | Gajendra Pal |       7 | M      | 747567354 |
|    5 | Aman         |       8 | M      | 747567254 |
|    6 | Samar Singh  |       9 | M      | 983131265 |
+------+--------------+---------+--------+-----------+

mysql> commit;
Query OK, 0 rows affected (0.19 sec)

mysql> rollback to s1;
ERROR 1305 (42000): SAVEPOINT s1 does not exist
mysql> INSERT INTO Passenger(P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES('SAar Singh',10,'M',973131265);
Query OK, 1 row affected (0.01 sec)

mysql> rollback;
Query OK, 0 rows affected (0.04 sec)

mysql> select* from Passenger;
+------+--------------+---------+--------+-----------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------+---------+--------+-----------+
|    1 | Davy Bansal  |       4 | M      | 644534342 |
|    2 | Gurdeep Singh|       5 | M      | 983135265 |
|    3 | Aryan Sharma |       6 | M      | 848567341 |
|    4 | Gajendra Pal |       7 | M      | 747567354 |
|    5 | Aman         |       8 | M      | 747567254 |
|    6 | Samar Singh  |       9 | M      | 983131265 |
+------+--------------+---------+--------+-----------+
mysql>
```

Committing to database so now changes are permanently made to database;

Rollback rolls back to the last made commit to the database.

DATA CONSTRAINTS

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the one or more columns in a table.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

DATA CONSTRAINTS

A.Naming of a constraint and why it's important

#A general convention used for naming constraints is

<table name>_<column name>_<constraint type>

(1) If a query (insert, update, delete) violates a constraint, SQL will generate an error message that will contain the constraint name. If the constraint name is clear and descriptive, the error message will be easier to understand; if the constraint name is a random, it's less clear.

(2) If a constraint needs to be modified in the future , it's very hard to do if you don't know what it's named. (ALTER TABLE MyTable drop CONSTRAINT um...)

TYPES OF CONSTRAINTS

a)Column Level Constraints: A column level constraint references a single column and is defined along with the definition of the column.Any constraint can be defined at the column level except for a COMPOSITE primary key constraint.

Syntax:

Column name datatype[CONSTRAINT constraint_name] constraint_type;

b)Table Level Constraints: A table level constraint references one or more columns and is defined separately from the definitions of the columns.Normally, it is written after all the columns are defined.

Syntax:

[CONSTRAINT constraint_name] constraint_type(Column names);

SQL - Operators (tutorialspoint.com)

Iss link par jo tables hai na unko slides par lana hai

Some constraints available in SQL

- NOT NULL Constraint – Ensures that a column cannot have NULL value.
- DEFAULT Constraint – Provides a default value for a column when none is specified.
- UNIQUE Constraint – Ensures that all values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key – Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.

Some Constraints at Glance...

1)Not Null Constraint:

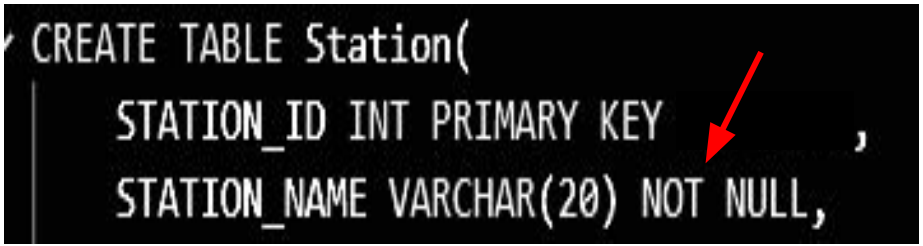
By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

Syntax for column level:

Attribute name datatype(size) CONSTRAINT constraint_name NOT NULL;



2) Default Constraint:

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

Syntax for column level:

Attribute name datatype(size) CONSTRAINT constraint_name DEFAULT <default value>,

3) Unique Constraint:

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Syntax for column level:

Attribute name datatype(size) CONSTRAINT constraint_name **UNIQUE;**

Syntax for table level:

CONSTRAINT constraint name UNIQUE(column names),

4) Primary Key Constraint:

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

Syntax for column level:

Attribute name datatype(size) CONSTRAINT

constraint_name *PRIMARY KEY;*

```
CREATE TABLE Passenger(
    P_ID INT PRIMARY KEY,
    P_NAME VARCHAR(50),
    SEAT_NO INT,
    GENDER VARCHAR(1),
    PH_NUMBER INT
);
```

Syntax for Table level:

CONSTRAINT constraint name PRIMARY KEY(column names);

```
+-----------+-------------+------+-----+
| Field     | Type        | Null | Key |
+-----------+-------------+------+-----+
| P_ID      | int         | NO   | PRI |
```

5) Foreign Key Constraint:

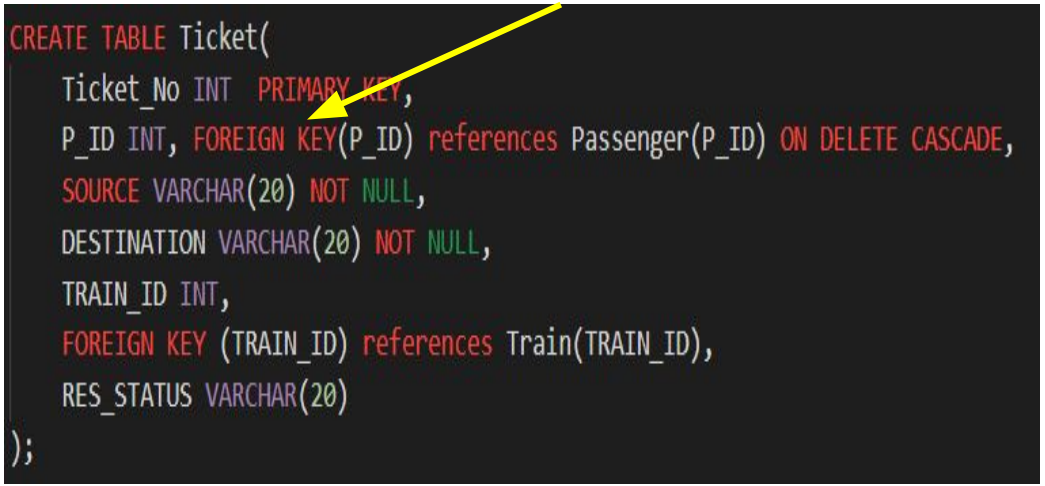
The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY of another table or of the same table.

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.

Syntax :

```
[CONSTRAINT [symbol]] FOREIGN KEY
[index_name] (col_name, ...)
REFERENCES tbl_name (col_name,...)
[ON DELETE reference_option]
[ON UPDATE reference_option]
```



```
CREATE TABLE Ticket(
    Ticket_No INT  PRIMARY KEY,
    P_ID INT, FOREIGN KEY(P_ID) references Passenger(P_ID) ON DELETE CASCADE,
    SOURCE VARCHAR(20) NOT NULL,
    DESTINATION VARCHAR(20) NOT NULL,
    TRAIN_ID INT,
    FOREIGN KEY (TRAIN_ID) references Train(TRAIN_ID),
    RES_STATUS VARCHAR(20)
);
```

6) Check Constraint:

The CHECK constraint is used to limit the value range that can be placed in a column or the value must satisfies some condition.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns.

Syntax for column level:

Attribute name datatype(size) CONSTRAINT constraint_name **CHECK(condition);**

Syntax for table level:

CONSTRAINT constraint name

CHECK (column names);

```
ALTER TABLE Passenger
ADD CONSTRAINT myCheckConstraint
CHECK (GENDER='M' OR GENDER='F' OR GENDER='T');
```

What is a foreign key with ON DELETE CASCADE & ON UPDATE CASCADE

A foreign key with **ON DELETE CASCADE** means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted.

—A foreign key with cascade delete can be created using either a CREATE TABLE statement or with an ALTER TABLE statement if we forgot to add it while creating table.

A foreign key using **ON UPDATE CASCADE** is also useful as the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key.

—A foreign key with cascade update can also be created in the same way as cascade delete using either a CREATE TABLE statement or with an ALTER TABLE statement if we forgot to add it while creating table.

We have created Ticket <->Fare_table and Passenger<->Ticket Tables with this property.

What is a foreign key with ON DELETE SET NULL & ON UPDATE SET NULL

A foreign key with **ON DELETE SET NULL** means that if a record in the parent table is being deleted, then the foreign key attribute value in corresponding record in the child table will set to NULL if it is allowed.

—A foreign key with on delete set NULL can be created using either a CREATE TABLE statement or with an ALTER TABLE statement if we forgot to add it while creating table.

A foreign key using **ON UPDATE SET NULL** is also useful as the foreign key attribute value in corresponding record in the child table will set to NULL if it is allowed when the referenced row is updated in the parent table which has a primary key.

—A foreign key with on update set can also be created in the same way as on delete set NULL using either a CREATE TABLE statement or with an ALTER TABLE statement if we forgot to add it while creating table.

SQL OPERATORS

OPERATORS: An operator is a reserved word or a character used primarily in an SQL statement WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators

ARITHMETIC OPERATORS

Arithmetic operators can perform arithmetical operations on numeric operands involved. These operators are used to manipulate mathematical calibrations like addition, multiplication, division, subtraction and other modulus numeric values in the SQL query. Arithmetic operators are addition(+), subtraction(-), multiplication(*), division(/) and modulo(%).

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

Used Arithmetic operators

```
mysql> select 4+5 AS 'add',5*6 AS 'multiplication',4/3 as 'divide', 3-4 as 'substraction' from dual;
+-----+----------------+--------+--------------+
| add | multiplication | divide | substraction |
+-----+----------------+--------+--------------+
|   9 |             30 | 1.3333 |           -1 |
+-----+----------------+--------+--------------+
1 row in set (0.05 sec)
```

We can also add integer and strings like below:

If we mix data types such as INT and VARCHAR, SQL Server will always attempt to convert everything to the one that has the highest precedence.

```
mysql> select (5+'20'),(5+'2ab') from dual;
+----------+----------+
| (5+'20') | (5+'2ab') |
+----------+----------+
|       25 |        7 |
+----------+----------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> select ('a5'+'20a'),('5a'+'a22b'),('5ab'+'22ab') from dual;
+--------------+--------------+---------------+
| ('a5'+'20a') | ('5a'+'a22b') | ('5ab'+'22ab') |
+--------------+--------------+---------------+
|           20 |            5 |            27 |
+--------------+--------------+---------------+
1 row in set, 6 warnings (0.00 sec)
```

SQL Comparison Operators

A comparison (or relational) operator is a mathematical symbol which is used to compare two values.

Comparison operators are used in conditions that compares one expression with another. The result of a comparison can be TRUE, FALSE

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

1. EQUAL TO OPERATOR(=)

```
mysql> SELECT * FROM PASSENGER WHERE SEAT_NO=6;
+------+--------------+---------+--------+------------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER  |
+------+--------------+---------+--------+------------+
|    3 | Aryan Sharma |       6 | M      | 848567341  |
+------+--------------+---------+--------+------------+
1 row in set (0.00 sec)
```

2. GREATER THAN(>)

```
mysql> select * from passenger
    -> where (seat_no)>6;
+------+----------------+---------+--------+------------+
| P_ID | P_NAME         | SEAT_NO | GENDER | PH_NUMBER  |
+------+----------------+---------+--------+------------+
|    4 | Gajendra Pal   |       7 | M      | 747567354  |
|    5 | Gajendra Sharma|       8 | M      | 988008004  |
+------+----------------+---------+--------+------------+
2 rows in set (0.00 sec)
```

3. LESS THAN(<)

```
mysql> select * from passenger
    -> where (seat_no)<6;
+------+--------------+---------+--------+------------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER  |
+------+--------------+---------+--------+------------+
|    1 | Davy Bansal  |       4 | M      | 644534342  |
|    2 | Gurdeep Singh|       5 | M      | 983135265  |
+------+--------------+---------+--------+------------+
2 rows in set (0.00 sec)
```

4.GREATER THAN OR EQUAL(>=)

```
mysql> SELECT * FROM PASSENGER WHERE SEAT_NO>=8;
+------+-----------------+---------+--------+------------+
| P_ID | P_NAME          | SEAT_NO | GENDER | PH_NUMBER  |
+------+-----------------+---------+--------+------------+
|    5 | Gajendra Sharma |       8 | M      |  988008004 |
+------+-----------------+---------+--------+------------+
1 row in set (0.00 sec)
```

5. LESS THAN OR EQUAL TO(<=)

```
mysql> select * from passenger
    -> where (seat_no<=6);
+------+--------------+---------+--------+------------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER  |
+------+--------------+---------+--------+------------+
|    1 | Davy Bansal  |       4 | M      | 644534342  |
|    2 | Gurdeep Singh|       5 | M      | 983135265  |
|    3 | Aryan Sharma |       6 | M      | 848567341  |
+------+--------------+---------+--------+------------+
3 rows in set (0.00 sec)
```

6. NOT EQUAL TO(<>)

```
mysql> SELECT * FROM PASSENGER WHERE SEAT_NO<>6;
+------+-----------------+---------+--------+------------+
| P_ID | P_NAME          | SEAT_NO | GENDER | PH_NUMBER  |
+------+-----------------+---------+--------+------------+
|    1 | Davy Bansal     |       4 | M      | 644534342  |
|    2 | Gurdeep Singh   |       5 | M      | 983135265  |
|    4 | Gajendra Pal    |       7 | M      | 747567354  |
|    5 | Gajendra Sharma |       8 | M      | 988008004  |
+------+-----------------+---------+--------+------------+
4 rows in set (0.00 sec)
```

SQL LOGICAL OPERATOR

Here is a list of all the logical operators available in SQL.

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE

1. AND OPERATOR - The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

```
mysql> select P_ID, P_Name from passenger
    -> where (P_Name='Gajendra Pal' && Seat_no=6);
Empty set, 1
```

2. OR OPERATOR - The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

```
mysql> select P_ID, P_Name from passenger
    -> where (P_Name='Gajendra Pal' || Seat_no=6);
+------+---------------+
| P_ID | P_Name        |
+------+---------------+
|    3 | Aryan Sharma  |
|    4 | Gajendra Pal  |
+------+---------------+
2 rows in set,
```

3. NOT OPERATOR - The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT BETWEEN, NOT IN, etc.

```
mysql> select * from passenger
    -> where !(seat_no<6);
+------+--------------+---------+--------+-----------+
| P_ID | P_NAME       | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------+---------+--------+-----------+
|    3 | Aryan Sharma |       6 | M      | 848567341 |
|    4 | Gajendra Pal |       7 | M      | 747567354 |
|    5 | Gajendra Sharma |    8 | M      | 988008004 |
+------+--------------+---------+--------+-----------+
3 rows in set
```

4.. ALL OPERATOR - The ALL operator is used to compare a value to all values in another value set.

```
mysql> SELECT *
    -> FROM fare_table
    -> WHERE Fare >= ALL( SELECT Fare from fare_table);
+-------+------+-----------+
| CLASS | Fare | Ticket_No |
+-------+------+-----------+
| THIRD |  500 |     20004 |
+-------+------+-----------+
1 row in set (0.01 sec)
```

5. LIKE OPERATOR - The LIKE operator is used to compare a value to similar values using wildcard operators.

- **Match pattern and filter the result as per pattern which satisfies the result.**
  String pattern:-
- **% indicates any number of characters whereas _ indicates single character**
- **['b%' -> starts with b or B]**
- **['%b%' -> b somewhere in string]**
- **['%y' -> ends with y]**
-
- **[_y -> 2 character string ending with y]**
- **[b__y -> string having 4 characters which starts with b and ends with y]**

```
mysql> select p_name from passenger
    -> where p_name like 'G%';
+-----------------+
| p_name          |
+-----------------+
| Gurdeep Singh   |
| Gajendra Pal    |
| Gajendra Sharma |
+-----------------+
3 rows in set (0.04 sec)
```

```
mysql> select p_name from passenger
    -> where p_name NOT like '_a%';
+---------------+
| p_name        |
+---------------+
| Gurdeep Singh |
| Aryan Sharma  |
+---------------+
2 rows in set (0.00 sec)
```

6. IS NULL OPERATOR - The NULL operator is used to compare a value with a NULL value.

```
mysql> SELECT *
    -> FROM passenger p
    -> WHERE p.PH_NUMBER IS NULL;
+------+------------------+---------+--------+-----------+
| P_ID | P_NAME           | SEAT_NO | GENDER | PH_NUMBER |
+------+------------------+---------+--------+-----------+
|  201 | Jennifer Lawrence |       9 | F      |      NULL |
+------+------------------+---------+--------+-----------+
```

```
mysql> SELECT *
    -> FROM passenger
    -> WHERE PH_NUMBER IS NOT NULL;
+------+--------------------+---------+--------+-----------+
| P_ID | P_NAME             | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------------+---------+--------+-----------+
|    1 | Camila Cabello     |       8 | F      | 600034342 |
|    2 | Jennifer Lawrence  |       9 | F      | 113135265 |
|    3 | Jennifer Lopez     |      10 | F      | 848567888 |
|    4 | Charlie Puth       |      11 | M      | 777567354 |
|    5 | Davy Bansal        |       4 | M      | 644534342 |
|    6 | Gurdeep Singh      |       5 | M      | 983135265 |
|    7 | Aryan Sharma       |       6 | M      | 848567341 |
|    8 | Gajendra Pal       |       7 | M      | 747567354 |
|  200 | Shabby Cabello     |      81 | F      | 600011139 |
|  202 | Jackline fernandez |      20 | F      | 848777888 |
|  203 | Amar Pal           |       7 | M      | 117567354 |
|  204 | Johnson KUMAR      |      19 | M      | 777567354 |
|  205 | Archit Sharma      |      61 | M      | 848567341 |
|  206 | David Bansal       |      41 | M      | 644534342 |
|  207 | GB Singh           |      55 | M      | 888888111 |
|  208 | Charlie Puth       |      72 | M      | 777567354 |
+------+--------------------+---------+--------+-----------+
```

7. ANY OPERATOR - The ANY operator is used to compare a value to any applicable value in the list as per the condition.

```
mysql> SELECT *
    -> FROM ticket t
    -> WHERE t.P_ID=ANY(
    -> SELECT p.P_ID
    -> FROM passenger p
    -> WHERE p.PH_NUMBER IS NULL
    -> );
+-----------+------+--------+-------------+----------+------------+
| Ticket_No | P_ID | SOURCE | DESTINATION | TRAIN_ID | RES_STATUS |
+-----------+------+--------+-------------+----------+------------+
|      2000 |  201 | MUMBAI | CHENNNAI    |    11899 | WAITING    |
+-----------+------+--------+-------------+----------+------------+
```

8. BETWEEN OPERATOR - The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

```
mysql> select * from passenger
    -> where seat_no Between 5 and 6;
+------+---------------+---------+--------+-----------+
| P_ID | P_NAME        | SEAT_NO | GENDER | PH_NUMBER |
+------+---------------+---------+--------+-----------+
|    2 | Gurdeep Singh |       5 | M      | 983135265 |
|    3 | Aryan Sharma  |       6 | M      | 848567341 |
+------+---------------+---------+--------+-----------+
2 rows in set (0.00 sec)
```

```
mysql> select * from passenger
    -> where seat_no>5 and seat_no<6;
Empty set (0.00 sec)
```

9. IN OPERATOR - The IN operator is used to compare a value to a list of literal values that have been specified.

```
mysql> SELECT * FROM PASSENGER WHERE SEAT_NO IN(4,8);
+------+-----------------+---------+--------+-----------+
| P_ID | P_NAME          | SEAT_NO | GENDER | PH_NUMBER |
+------+-----------------+---------+--------+-----------+
|    1 | Davy Bansal     |       4 | M      | 644534342 |
|    5 | Gajendra Sharma |       8 | M      | 988008004 |
+------+-----------------+---------+--------+-----------+
2 rows in set (0.04 sec)
```

FUNCTIONS

MySQL comes bundled with a number of built in functions. Built in functions are simply functions come already implemented in the SQL. These functions allow us to perform different types of manipulations on the data.

NUMERIC FUNCTIONS

1. | ROUND | Rounds a number to a specified number of decimal places |

```
mysql> SELECT ROUND(1.23),ROUND(1.26,1);
+-------------+---------------+
| ROUND(1.23) | ROUND(1.26,1) |
+-------------+---------------+
|           1 |           1.3 |
+-------------+---------------+
1 row in set (0.04 sec)
```

2.	CEILING	Returns the smallest integer value that is >= a number
3.	FLOOR	Returns the largest integer value that is <= to a number

```
nysql> select ceiling(2.43),floor(2.43) from dual;
+--------------+------------+
| ceiling(2.43) | floor(2.43) |
+--------------+------------+
|            3 |          2 |
+--------------+------------+
1 row in set (0.00 sec)
```

```
mysql> select ceiling(2.83),floor(2.83) from dual;
+--------------+------------+
| ceiling(2.83) | floor(2.83) |
+--------------+------------+
|            3 |          2 |
+--------------+------------+
1 row in set (0.00 sec)
```

4.	ABS	Returns the absolute value of a number
5.	RAND	Returns a random number

```
mysql> select abs(-2.83),rand() from dual;
+------------+--------------------+
| abs(-2.83) | rand()             |
+------------+--------------------+
|       2.83 | 0.5261048986819619 |
+------------+--------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT RAND()*(10-5)+5 from dual;
+-------------------+
| RAND()*(10-5)+5   |
+-------------------+
| 5.722220754923504 |
+-------------------+
1 row in set (0.01 sec)

mysql> SELECT RAND()*(10-5)+5 from dual;
+-------------------+
| RAND()*(10-5)+5   |
+-------------------+
| 5.71953044805632  |
+-------------------+
1 row in set (0.00 sec)
```

6.

TRUNCATE	Truncates a number to the specified number of decimal places

```
mysql> select truncate(2.56546,5),truncate(0.000005555,10),truncate(21345.121,1) from dual;
+---------------------+--------------------------+-----------------------+
| truncate(2.56546,5) | truncate(0.000005555,10) | truncate(21345.121,1) |
+---------------------+--------------------------+-----------------------+
|             2.56546 |              0.000005555 |               21345.1 |
+---------------------+--------------------------+-----------------------+
1 row in set (0.00 sec)
```

7.

POWER	Returns the value of a number raised to the power of another number

```
mysql> SELECT POW(2,3), POW(2,-2);
+----------+-----------+
| POW(2,3) | POW(2,-2) |
+----------+-----------+
|        8 |      0.25 |
+----------+-----------+
1 row in set (0.04 sec)
```

STRING FUNCTIONS

1. LENGTH — Returns the length of a string (in bytes)

```
mysql> select P_name,LENGTH(P_NAME) FROM PASSENGER;
+-----------------+----------------+
| P_name          | LENGTH(P_NAME) |
+-----------------+----------------+
| Davy Bansal     |             11 |
| Gurdeep Singh   |             13 |
| Aryan Sharma    |             12 |
| Gajendra Pal    |             12 |
| Gajendra Sharma |             15 |
+-----------------+----------------+
5 rows in set (0.04 sec)
```

2.	LOWER	Converts a string to lower-case
3.	UPPER	Converts a string to upper-case

```
mysql> SELECT LOWER(P_NAME) AS LOWERCASE,UPPER(P_NAME) AS UPPERCASE FROM PASSENGER;
+-----------------+-----------------+
| LOWERCASE       | UPPERCASE       |
+-----------------+-----------------+
| davy bansal     | DAVY BANSAL     |
| gurdeep singh   | GURDEEP SINGH   |
| aryan sharma    | ARYAN SHARMA    |
| gajendra pal    | GAJENDRA PAL    |
| gajendra sharma | GAJENDRA SHARMA |
+-----------------+-----------------+
5 rows in set (0.00 sec)
```

4	LTRIM	Removes leading spaces from a string
5	RTRIM	Removes trailing spaces from a string

```
mysql> SELECT LTRIM('   LEFTTRIM');
+----------------------+
| LTRIM('   LEFTTRIM') |
+----------------------+
| LEFTTRIM             |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT RTRIM('RIGHTTRIM   ') AS DEMO;
+-----------+
| DEMO      |
+-----------+
| RIGHTTRIM |
+-----------+
1 row in set (0.00 sec)
```

6.

TRIM	Removes leading and trailing spaces from a string

```
mysql> SELECT TRIM('    THIS IS  OUR STRING FUNCTIONS   ');
+-----------------------------------------------+
| TRIM('    THIS IS  OUR STRING FUNCTIONS   ') |
+-----------------------------------------------+
| THIS IS  OUR STRING FUNCTIONS                 |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

7.

LEFT	Extracts a number of characters from a string (starting from left)

8.

RIGHT	Extracts a number of characters from a string (starting from right)

```
mysql> SELECT LEFT(P_NAME,4),RIGHT(P_NAME,4) FROM PASSENGER;
+-----------------+------------------+
| LEFT(P_NAME,4) | RIGHT(P_NAME,4) |
+-----------------+------------------+
| Davy            | nsal             |
| Gurd            | ingh             |
| Arya            | arma             |
| Gaje            |  Pal             |
| Gaje            | arma             |
+-----------------+------------------+
5 rows in set (0.00 sec)
```

9. SUBSTR — Extracts a substring from a string (starting at any position)

```
mysql> SELECT SUBSTR(P_NAME,6,4) FROM PASSENGER;
+----------------------+
| SUBSTR(P_NAME,6,4) |
+----------------------+
| Bans                 |
| ep S                 |
|  Sha                 |
| dra                  |
| dra                  |
+----------------------+
5 rows in set (0.00 sec)
```

10. LOCATE — Returns the position of the first occurrence of a substring in a string

- If the substring is not found within the original string, this function returns 0.
- This function performs a case-insensitive search.

```
mysql> SELECT LOCATE('SHA',P_NAME) FROM PASSENGER;
+-------------------------+
| LOCATE('SHA',P_NAME) |
+-------------------------+
|                       0 |
|                       0 |
|                       7 |
|                       0 |
|                      10 |
+-------------------------+
5 rows in set (0.00 sec)
```

11.

INSTR	Returns the position of the first occurrence of a string in another string

```
mysql> SELECT INSTR(P_NAME,'A') FROM PASSENGER;
+---------------------+
| INSTR(P_NAME,'A') |
+---------------------+
|                   2 |
|                   0 |
|                   1 |
|                   2 |
|                   2 |
+---------------------+
5 rows in set (0.04 sec)
```

12.

CONCAT	Adds two or more expressions together

```
mysql> SELECT CONCAT(P_NAME,SEAT_NO) FROM PASSENGER;
+-----------------------+
| CONCAT(P_NAME,SEAT_NO) |
+-----------------------+
| Davy Bansal4          |
| Gurdeep Singh5        |
| Aryan Sharma6         |
| Gajendra Pal7         |
| Gajendra Sharma8      |
+-----------------------+
5 rows in set (0.00 sec)
```

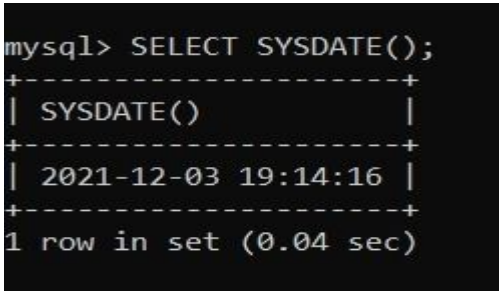
|| Operator to Concatenate Strings

Setting *sql_mode* to PIPES_AS_CONCAT or ANSI enables || to concatenate strings:

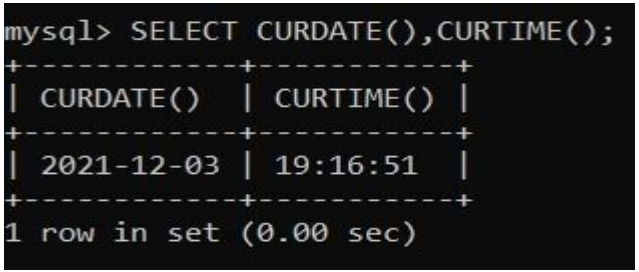
```
+---------------------------------------------+
| @@SQL_MODE                                  |
+---------------------------------------------+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+---------------------------------------------+
1 row in set (0.00 sec)

mysql> SET sql_mode='STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION,PIPES_AS_CONCAT';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> select p_name || ' Contact Number:- '|| Ph_number AS 'DETAILS' from passenger;
+-----------------------------------------+
| DETAILS                                 |
+-----------------------------------------+
| Davy Bansal Contact Number:- 644534342   |
| Gurdeep Singh Contact Number:- 983135265 |
| Aryan Sharma Contact Number:- 848567341  |
| Gajendra Pal Contact Number:- 747567354  |
| Gajendra Sharma Contact Number:- 8008004 |
+-----------------------------------------+
5 rows in set (0.04 sec)
```

DATE FUNCTIONS:-

1. SYSDATE() FUNCTION: This function returns the time at which the function executes.



2. CURDATE() AND CURTIME() FUNCTION - CURDATE function returns the current date and CURTIME function returns the current time.



1. DATE() FUNCTION - Extracts the date part of the date.
2. MONTH() FUNCTION- Returns the month for date.
3. MONTHNAME() FUNCTION- Returns the name of month for date.
4. YEAR() FUNCTION- Returns the year for date.
5. DAYNAME()- Returns the name of the weekday for date.
6. DAYOFMONTH()- Returns the day of the month for date.
7. DAYOFWEEK()-Returns the day weekday index for date.
8. DAYOFYEAR()-Returns the day of the year for date.

```
mysql> SELECT DATE('2021-12-03 19:14:16'),MONTH('2021-12-03'),MONTHNAME('2021-12-03'),YEAR('2021-12-03'),DAYNAME('2021-12-03'),DAYOFMONTH('2021-12-03'),DAYOFWEEK('2021-12-03'),DAYOFYEAR('2021-12-03');
+-----------------------------+-----------------------+-------------------------+----------------------+-------------------------+-----------------------------+---------------------------+---------------------------+
| DATE('2021-12-03 19:14:16') | MONTH('2021-12-03') | MONTHNAME('2021-12-03') | YEAR('2021-12-03') | DAYNAME('2021-12-03') | DAYOFMONTH('2021-12-03') | DAYOFWEEK('2021-12-03') | DAYOFYEAR('2021-12-03') |
+-----------------------------+-----------------------+-------------------------+----------------------+-------------------------+-----------------------------+---------------------------+---------------------------+
| 2021-12-03                  |                    12 | December                |                 2021 | Friday                  |                           3 |                         6 |                       337 |
+-----------------------------+-----------------------+-------------------------+----------------------+-------------------------+-----------------------------+---------------------------+---------------------------+
1 row in set (0.04 sec)
```

Difference between SYSDATE() and NOW():-

***NOW()* function - Returns a steady time that indicates the time at which the particular statement began to execute.**

***SYSDATE()* function- Returns the accurate time at which the statement executes.**

```
mysql> Select NOW(), SLEEP(5), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(5) | NOW()               |
+---------------------+----------+---------------------+
| 2021-12-04 17:19:30 |        0 | 2021-12-04 17:19:30 |
+---------------------+----------+---------------------+
1 row in set (5.01 sec)
```

```
mysql> Select SYSDATE(), SLEEP(5), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(5) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2021-12-04 17:20:02 |        0 | 2021-12-04 17:20:07 |
+---------------------+----------+---------------------+
1 row in set (5.00 sec)
```

Aggregate Function

SQL aggregation is the task of collecting a set of values to return a single value. It is done with the help of aggregate functions, such as SUM, COUNT, and AVG. For example, in a database of products, you might want to calculate the average price of the whole inventory.

```
mysql> SELECT * FROM TICKET JOIN FARE_TABLE USING(TICKET_NO);
+-----------+------+----------+-------------+----------+------------+--------+------+
| Ticket_No | P_ID | SOURCE   | DESTINATION | TRAIN_ID | RES_STATUS | CLASS  | Fare |
+-----------+------+----------+-------------+----------+------------+--------+------+
|     20002 |    2 | Pune     | Thane       |    12138 | CONFIRMED  | SECOND |  100 |
|     20003 |    3 | Chennai  | Darbangha   |    11862 | CONFIRMED  | FIRST  |  250 |
|     20004 |    4 | Bathinda | Thane       |    12138 | CONFIRMED  | THIRD  |  500 |
|     20005 |  202 | Chennai  | Bathinda    |    34521 | CONFIRMED  | SECOND |  150 |
+-----------+------+----------+-------------+----------+------------+--------+------+
4 rows in set (0.00 sec)

mysql> SELECT SUM(FARE) FROM TICKET JOIN FARE_TABLE USING(TICKET_NO);
+-----------+
| SUM(FARE) |
+-----------+
|      1000 |
+-----------+
```

SUM()

Aggregate (count())

- COUNT: It is used to count the number of rows in a set. the COUNT function includes rows with NULL values.

```
mysql>  SELECT * FROM TICKET JOIN FARE_TABLE USING(TICKET_NO);
+-----------+------+----------+-------------+----------+------------+--------+------+
| Ticket_No | P_ID | SOURCE   | DESTINATION | TRAIN_ID | RES_STATUS | CLASS  | Fare |
+-----------+------+----------+-------------+----------+------------+--------+------+
|     20002 |    2 | Pune     | Thane       |    12138 | CONFIRMED  | SECOND |  100 |
|     20003 |    3 | Chennai  | Darbangha   |    11862 | CONFIRMED  | FIRST  |  250 |
|     20004 |    4 | Bathinda | Thane       |    12138 | CONFIRMED  | THIRD  |  500 |
|     20005 |  202 | Chennai  | Bathinda    |    34521 | CONFIRMED  | SECOND |  150 |
+-----------+------+----------+-------------+----------+------------+--------+------+
4 rows in set (0.00 sec)

mysql> SELECT COUNT(*) AS total_bookings FROM TICKET JOIN  FARE_TABLE USING(TICKET_NO);
+----------------+
| total_bookings |
+----------------+
|              4 |
+----------------+
```

Aggregate (AVG)

```
mysql>  SELECT * FROM TICKET JOIN FARE_TABLE USING(TICKET_NO);
+-----------+------+----------+-------------+----------+------------+--------+------+
| Ticket_No | P_ID | SOURCE   | DESTINATION | TRAIN_ID | RES_STATUS | CLASS  | Fare |
+-----------+------+----------+-------------+----------+------------+--------+------+
|     20002 |    2 | Pune     | Thane       |    12138 | CONFIRMED  | SECOND |  100 |
|     20003 |    3 | Chennai  | Darbangha   |    11862 | CONFIRMED  | FIRST  |  250 |
|     20004 |    4 | Bathinda | Thane       |    12138 | CONFIRMED  | THIRD  |  500 |
|     20005 |  202 | Chennai  | Bathinda    |    34521 | CONFIRMED  | SECOND |  150 |
+-----------+------+----------+-------------+----------+------------+--------+------+
```

```
mysql> SELECT AVG(FARE_TABLE.FARE) FROM TICKET JOIN FARE_TABLE USING(TICKET_NO);
+-----------------------+
| AVG(FARE_TABLE.FARE) |
+-----------------------+
|              250.0000 |
+-----------------------+
1 row in set (0.00 sec)
```

- AVG: This calculates the average of all values in a group.

Aggregate(Max() , Min())

```
mysql> SELECT * FROM FARE_TABLE;
+--------+------+-----------+
| CLASS  | Fare | Ticket_No |
+--------+------+-----------+
| SECOND |  100 |     20002 |
| FIRST  |  250 |     20003 |
| THIRD  |  500 |     20004 |
| SECOND |  150 |     20005 |
+--------+------+-----------+
```

MAX: MAX aggregate function in SQL returns the largest value in a group

```
mysql> SELECT MAX(FARE) , MIN(FARE) FROM FARE_TABLE;
+-----------+-----------+
| MAX(FARE) | MIN(FARE) |
+-----------+-----------+
|       500 |       100 |
+-----------+-----------+
1 row in set (0.00 sec)
```

- MIN: It returns the lowest value in a group.

ORDER BY(Increasing(Default))

The ORDER BY keyword is used to sort the result-set in ascending or descending order.
The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
mysql> SELECT TICKET_NO,RES_STATUS FROM TICKET ORDER BY TICKET_NO;
+-----------+------------+
| TICKET_NO | RES_STATUS |
+-----------+------------+
|     20001 | CONFIRMED  |
|     20002 | CONFIRMED  |
|     20003 | WAITING    |
|     20004 | CONFIRMED  |
|     20005 | WAITING    |
+-----------+------------+
```

Order by (Descending)

```
mysql> SELECT TICKET_NO,RES_STATUS FROM TICKET ORDER BY TICKET_NO DESC;
+-----------+------------+
| TICKET_NO | RES_STATUS |
+-----------+------------+
|     20005 | WAITING    |
|     20004 | CONFIRMED  |
|     20003 | WAITING    |
|     20002 | CONFIRMED  |
|     20001 | CONFIRMED  |
+-----------+------------+
```

Order by (Combined)

```
mysql> SELECT * FROM PASSENGER
    -> ORDER BY GENDER DESC,SEAT_NO ASC;
+------+-----------------+---------+--------+------------+
| P_ID | P_NAME          | SEAT_NO | GENDER | PH_NUMBER  |
+------+-----------------+---------+--------+------------+
|    1 | Davy Bansal     |       4 | M      | 644534342  |
|    2 | Gurdeep Singh   |       5 | M      | 983135265  |
|    3 | Aryan Sharma    |       6 | M      | 848567341  |
|    4 | Gajendra Pal    |       7 | M      | 747567354  |
|    5 | Gajendra Sharma |       8 | M      | 988008004  |
+------+-----------------+---------+--------+------------+
5 rows in set (0.00 sec)
```

Order by (column no)

```
mysql>  SELECT * FROM PASSENGER
    -> ORDER BY 3 DESC;
+------+------------------+----------+--------+------------+
| P_ID | P_NAME           | SEAT_NO  | GENDER | PH_NUMBER  |
+------+------------------+----------+--------+------------+
|    5 | Gajendra Sharma  |        8 | M      |    8008004 |
|    4 | Gajendra Pal     |        7 | M      | 747567354  |
|    3 | Aryan Sharma     |        6 | M      | 848567341  |
|    2 | Gurdeep Singh    |        5 | M      | 983135265  |
|    1 | Davy Bansal      |        4 | M      | 644534342  |
+------+------------------+----------+--------+------------+
5 rows in set (0.00 sec)
```

Group By

The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

```
mysql> SELECT *
    -> FROM passenger
    -> GROUP BY GENDER,P_ID
    -> ORDER BY GENDER,P_ID;
+------+--------------------+---------+--------+-----------+
| P_ID | P_NAME             | SEAT_NO | GENDER | PH_NUMBER |
+------+--------------------+---------+--------+-----------+
|    1 | Camila Cabello     |       8 | F      | 600034342 |
|    2 | Jennifer Lawrence  |       9 | F      | 113135265 |
|    3 | Jennifer Lopez     |      10 | F      | 848567888 |
|  200 | Shabby Cabello     |      81 | F      | 600011139 |
|  201 | Jennifer Lawrence  |       9 | F      |      NULL |
|  202 | Jackline fernandez |      20 | F      | 848777888 |
```

4	Charlie Puth	11	M	777567354
5	Davy Bansal	4	M	644534342
6	Gurdeep Singh	5	M	983135265
7	Aryan Sharma	6	M	848567341
8	Gajendra Pal	7	M	747567354
203	Amar Pal	7	M	117567354
204	Johnson KUMAR	19	M	777567354
205	Archit Sharma	61	M	848567341
206	David Bansal	41	M	644534342
207	GB Singh	55	M	888888111
208	Charlie Puth	72	M	777567354

Having By

The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified condition. If you omit the GROUP BY clause, the HAVING clause behaves like the WHERE clause.

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition
GROUP BY
    group_by_expression
HAVING
    group_condition;
```

```
mysql> SELECT
    -> CLASS,
    ->     SUM(FARE) AS TOTAL_FARE_IN_CLASS
    -> FROM fare_table
    -> GROUP BY CLASS
    -> HAVING TOTAL_FARE_IN_CLASS >=500
    -> ORDER BY CLASS;
+-------+----------------------+
| CLASS | TOTAL_FARE_IN_CLASS |
+-------+----------------------+
| FIRST |                  910 |
| THIRD |                  500 |
+-------+----------------------+
```

SUBQUERIES

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Syntax:

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

Single Row,Multiple Row,Multiple Column Subqueries

Multiple Row & Multiple Column

```
mysql> SELECT *
    -> FROM PASSENGER p
    -> WHERE p.P_ID IN(
    -> SELECT t.P_ID
    -> FROM TICKET t
    -> WHERE t.RES_STATUS='CONFIRMED'
    -> );
+------+-------------------+---------+--------+-----------+
| P_ID | P_NAME            | SEAT_NO | GENDER | PH_NUMBER |
+------+-------------------+---------+--------+-----------+
|    1 | Camila Cabello    |       8 | F      | 600034342 |
|    2 | Jennifer Lawrence |       9 | F      | 113135265 |
|    4 | Charlie Puth      |      11 | M      | 777567354 |
+------+-------------------+---------+--------+-----------+
```

Subqueries With Insert Statement

INSERT statement can be used with subqueries.

```
INSERT INTO ticket VALUES
(20006,5,'PUNE','THANE',
(SELECT t.TRAIN_ID FROM train t WHERE t.TRAIN_NAME='PUNJAB MAIL'),
'WAITING')
```

AFTER INSERTION USING SUBQUERY

```
mysql> SELECT * FROM TICKET;
+-----------+------+-----------+-------------+----------+------------+
| Ticket_No | P_ID | SOURCE    | DESTINATION | TRAIN_ID | RES_STATUS |
+-----------+------+-----------+-------------+----------+------------+
|     20001 |    1 | Darbangha | Thane       |    11899 | CONFIRMED  |
|     20002 |    2 | Pune      | Thane       |    12138 | CONFIRMED  |
|     20003 |    3 | Chennai   | Darbangha   |    11862 | WAITING    |
|     20004 |    4 | Bathinda  | Thane       |    12138 | CONFIRMED  |
|     20005 |  202 | Chennai   | Bathinda    |    34521 | WAITING    |
|     20006 |    5 | PUNE      | THANE       |    12138 | WAITING    |
+-----------+------+-----------+-------------+----------+------------+
```

Subqueries With Update Statement

```
mysql>  UPDATE ticket t
    -> SET t.RES_STATUS='CONFIRMED'
    -> WHERE t.P_ID IN(
    -> SELECT p.P_ID FROM PASSENGER p WHERE p.GENDER='F'
    -> )
    -> ;
Query OK, 2 rows affected (0.01 sec)
Rows matched: 4  Changed: 2  Warnings: 0

mysql> SELECT * FROM TICKET;
+-----------+------+-----------+-------------+----------+------------+
| Ticket_No | P_ID | SOURCE    | DESTINATION | TRAIN_ID | RES_STATUS |
+-----------+------+-----------+-------------+----------+------------+
|     20001 |    1 | Darbangha | Thane       |    11899 | CONFIRMED  |
|     20002 |    2 | Pune      | Thane       |    12138 | CONFIRMED  |
|     20003 |    3 | Chennai   | Darbangha   |    11862 | CONFIRMED  |
|     20004 |    4 | Bathinda  | Thane       |    12138 | CONFIRMED  |
|     20005 |  202 | Chennai   | Bathinda    |    34521 | CONFIRMED  |
|     20006 |    5 | PUNE      | THANE       |    12138 | WAITING    |
+-----------+------+-----------+-------------+----------+------------+
```

Subqueries With Delete Statement

```
mysql> DELETE FROM ticket tk WHERE tk.TRAIN_ID=(SELECT t.TRAIN_ID FROM train t WHERE TRAIN_NAME='DEHRADUN MAIL');
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM TICKET;
+-----------+------+---------+-------------+----------+------------+
| Ticket_No | P_ID | SOURCE  | DESTINATION | TRAIN_ID | RES_STATUS |
+-----------+------+---------+-------------+----------+------------+
|     20002 |    2 | Pune    | Thane       |    12138 | CONFIRMED  |
|     20003 |    3 | Chennai | Darbangha   |    11862 | CONFIRMED  |
|     20004 |    4 | Bathinda| Thane       |    12138 | CONFIRMED  |
|     20005 |  202 | Chennai | Bathinda    |    34521 | CONFIRMED  |
|     20006 |    5 | PUNE    | THANE       |    12138 | WAITING    |
+-----------+------+---------+-------------+----------+------------+
5 rows in set (0.00 sec)
```

Subqueries in a Select Clause

```
mysql> SELECT P_ID,
    ->        P_NAME,
    ->        SEAT_NO,
    ->        GENDER,
    ->        (SELECT AVG(FARE)
    ->              FROM Fare_table) AS Avg_Fare
    -> from Passenger;
```

P_ID	P_NAME	SEAT_NO	GENDER	Avg_Fare
1	Camila Cabello	8	F	250.0000
2	Jennifer Lawrence	9	F	250.0000
3	Jennifer Lopez	10	F	250.0000
4	Charlie Puth	11	M	250.0000
5	Davy Bansal	4	M	250.0000
6	Gurdeep Singh	5	M	250.0000
7	Aryan Sharma	6	M	250.0000
8	Gajendra Pal	7	M	250.0000
200	Shabby Cabello	81	F	250.0000
201	Jennifer Lawrence	9	F	250.0000
202	Jackline fernandez	20	F	250.0000
203	Amar Pal	7	M	250.0000
204	Johnson KUMAR	19	M	250.0000
205	Archit Sharma	61	M	250.0000
206	David Bansal	41	M	250.0000
207	GB Singh	55	M	250.0000
208	Charlie Puth	72	M	250.0000

Subqueries in a From Clause

We Can place a subquery in the FROM clause of an outer query.

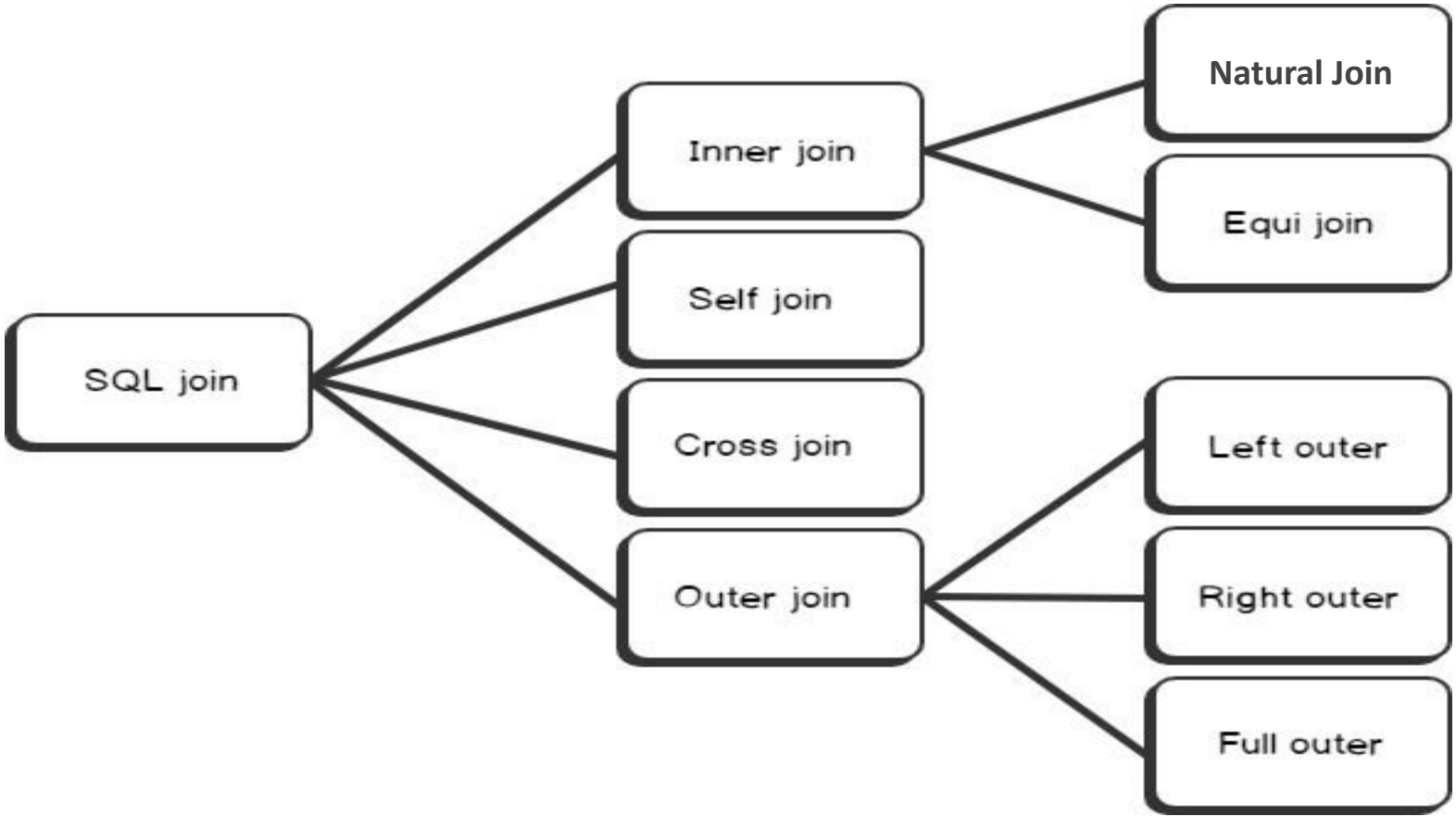
```
mysql> Select Ticket_No
    -> From(Select *
    -> From Fare_table
    -> Where Fare<400) as Fare_Filtered;
+-----------+
| Ticket_No |
+-----------+
|     20002 |
|     20003 |
|     20005 |
+-----------+
```

JOINS

A SQL Join statement is used to combine data or rows from two or more tables based on a particular column to get a required result.

Different types of Joins are as follows:

1. **INNER JOIN**
2. **CROSS JOIN**
3. **SELF JOIN**
4. **OUTER JOIN**


SQL join
Inner join
Self join
Cross join
Outer join
Natural Join
Equi join
Left outer
Right outer
Full outer

CROSS JOIN

It is a join which returns the Cartesian Product of rows from the tables in the Join.In other words, it combines each row from the first table with each row from second table.

```
98
99  SELECT * FROM Passenger CROSS JOIN Ticket;
```

9:11:34 PM × | 8:59:16 PM × | 8:58:01 PM × | 8:56:06 PM × | 8:55:50 PM × | 8:52:28 PM × | 8:50:44 PM × | 8:49:50 PM × | 8:49:28 PM × | 8:49:12 P

Success
(36 rows) 0.6 s

Explore | SQL | Data | Chart | Export | Search results...

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER	Ticket_No	P_ID	SOURCE	DESTINATION	TRAIN_ID	RES_STATU
1	Ashish Kumar	2	M	999999111	20001	1	Darbangha	Thane	11899	C
1	Tara Sutaria	1	F	999292939	20001	1	Darbangha	Thane	11899	C
1	Gajendra Pal	7	M	747567354	20001	1	Darbangha	Thane	11899	C
1	Aryan Sharma	6	M	848567341	20001	1	Darbangha	Thane	11899	C
1	Gurdeep Singh	5	M	983135265	20001	1	Darbangha	Thane	11899	C
1	Davy Bansal	4	M	644534342	20001	1	Darbangha	Thane	11899	C
2	Ashish Kumar	2	M	999999111	20002	2	Pune	Thane	12138	C
2	Tara Sutaria	1	F	999292939	20002	2	Pune	Thane	12138	C
2	Gajendra Pal	7	M	747567354	20002	2	Pune	Thane	12138	C
2	Aryan Sharma	6	M	848567341	20002	2	Pune	Thane	12138	C
2	Gurdeep Singh	5	M	983135265	20002	2	Pune	Thane	12138	C
2	Davy Bansal	4	M	644534342	20002	2	Pune	Thane	12138	C

INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies.Inner Join mainly results the Intersection of tables.
Types of Inner join:

1.Natural join

2.Equi Join

NATURAL JOIN

The NATURAL JOIN is an operation that creates join on the basis of common columns in a table and the value of that common attribute must be equal.To perform Natural Join there must be atleast one common attribute between tables to join.

```
53
54    SELECT * FROM Passenger NATURAL JOIN Ticket;
```

4:42:37 PM × | 4:42:24 PM × | 4:40:22 PM × | 4:39:49 PM × | 4:39:43 PM × | 4:38:24 PM × | 4:38:07 PM × | 4:30:43 PM × | 4:30:39 PM × | 4:[illegible]:00

Success
(6 rows) 0.2 s

Explore | SQL | Data | Chart | Export | Search results...

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Davy Bansal	4	M	644534342
2	Gurdeep Singh	5	M	983135265
3	Aryan Sharma	6	M	848567341
4	Gajendra Pal	7	M	747567354
106	Tara Sutaria	1	F	999292939
107	Ashish Kumar	2	M	999999111

EQUI JOIN

An equi-join is a join based on equality or matching column values. This equality is indicated with an equal sign (=) as the comparison operator in the WHERE clause.It is same as NATURAL JOIN but the only difference is that in EQUI JOIN we can equate any column from both the tables

```
73
74  SELECT * FROM Passenger,Ticket WHERE Passenger.P_ID=Ticket.P_ID;
```

6:11:36 PM | 6:10:53 PM | 6:10:17 PM | 6:09:51 PM | 6:09:40 PM | 6:08:55 PM | 6:07:54 PM | 6:07:42 PM | 6:06:52 PM

Success
(6 rows) 0.5 s

Explore | SQL | Data | Chart | Export

Search results...

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER	Ticket_No	P_ID	SOURCE	DESTINATION	TRAIN_ID	RES_ST
1	Davy Bansal	4	M	644534342	20001	1	Darbangha	Thane	11899	C
2	Gurdeep Singh	5	M	983135265	20002	2	Pune	Thane	12138	C
3	Aryan Sharma	6	M	848567341	20003	3	Chennai	Darbangha	11862	C
4	Gajendra Pal	7	M	747567354	20004	4	Bathinda	Thane	12138	C
106	Tara Sutaria	1	F	999292939	20005	106	Chennai	Thane	34521	C
107	Ashish Kumar	2	M	999999111	20006	107	Chennai	Bathinda	34521	C

SELF JOIN

In Self-Join a table joined with itself. That is, each row of the table is joined with itself depending on some conditions. Mainly it is a join between two copies of the same table.

```
SELECT R1.Class,R1.Train_ID,R1.Fare FROM Fare_Table AS R1,Fare_Table AS R2 WHERE R1.Receipt_No=R2.Receipt_No ORDER BY R1.Class;
```

9:19:02 PM × | 9:11:34 PM × | 8:59:16 PM × | 8:58:01 PM × | 8:56:06 PM × | 8:55:50 PM × | 8:52:28 PM ×

Success
(6 rows) 0.5 s

Explore | SQL | Data | Chart | Export | 🔗

Class	Train_ID	Fare
FIRST CLASS	11899	1700
FIRST CLASS	11862	1300
SECOND CLASS	11899	1000
SECOND CLASS	11862	700
SECOND CLASS	11862	900
THIRD CLASS	11899	700

OUTER JOIN

The OUTER JOIN returns matching rows as well as unmatched rows from one or both the tables.Outer Join mainly results the Union of tables.

Types of Outer join:

1.Left Outer Join

2.Right Outer Join

3.Full Outer Join

LEFT OUTER JOIN

This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null.

```
68
69  SELECT * FROM Passenger LEFT OUTER JOIN Ticket ON Passenger.P_ID=Ticket.P_ID;
```

6:00:05 PM × | 5:57:47 PM × | 5:57:21 PM × | 5:57:12 PM × | 5:55:39 PM × | 5:54:53 PM × | 5:51:30 PM × | 5:50:42 PM × | 5:49:13 PM × | 5:47

Success
(7 rows) 0.4 s

Explore | SQL | Data | Chart | Export | Search results...

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER	Ticket_No	P_ID	SOURCE	DESTINATION	TRAIN_ID	RES_S
1	Davy Bansal	4	M	644534342	20001	1	Darbangha	Thane	11899	C
1	Davy Bansal	4	M	644534342	20005	1	Chennai	Thane	34521	C
2	Gurdeep Singh	5	M	983135265	20002	2	Pune	Thane	12138	C
3	Aryan Sharma	6	M	848567341	20003	3	Chennai	Darbangha	11862	C
4	Gajendra Pal	7	M	747567354	20004	4	Bathinda	Thane	12138	C
NULL	Tara Sutaria	1	F	999292939	NULL	NULL	NULL	NULL	NULL	NULL
107	Ashish Kumar	2	M	999999111	20006	107	Chennai	Bathinda	34521	C

RIGHT OUTER JOIN

This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null.

```
68
69  SELECT * FROM Passenger RIGHT OUTER JOIN Ticket ON Passenger.P_ID=Ticket.P_ID;
```

6:00:38 PM × | 6:00:05 PM × | 5:57:47 PM × | 5:57:21 PM × | 5:57:12 PM × | 5:55:39 PM × | 5:54:53 PM × | 5:51:30 PM × | 5:50:42 PM × | 5:49:43 PM

Success
(6 rows) 0.6 s

Explore | SQL | Data | Chart | Export | Search results...

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER	Ticket_No	P_ID	SOURCE	DESTINATION	TRAIN_ID	RES_STATU
1	Davy Bansal	4	M	644534342	20001	1	Darbangha	Thane	11899	C
2	Gurdeep Singh	5	M	983135265	20002	2	Pune	Thane	12138	C
3	Aryan Sharma	6	M	848567341	20003	3	Chennai	Darbangha	11862	C
4	Gajendra Pal	7	M	747567354	20004	4	Bathinda	Thane	12138	C
1	Davy Bansal	4	M	644534342	20005	1	Chennai	Thane	34521	C
107	Ashish Kumar	2	M	999999111	20006	107	Chennai	Bathinda	34521	C

FULL OUTER JOIN

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

```
68
69    SELECT * FROM Passenger LEFT OUTER JOIN Ticket ON Passenger.P_ID=Ticket.P_ID
70    UNION
71    SELECT * FROM Passenger RIGHT OUTER JOIN Ticket ON Passenger.P_ID=Ticket.P_ID;
```

6:02:19 PM × | 6:01:09 PM × | 6:00:50 PM × | 6:00:38 PM × | 6:00:05 PM × | 5:57:47 PM × | 5:57:21 PM × | 5:57:12 PM × | 5:55:39 PM × | 5:54:58 P

Success
(7 rows) 0.6 s

Explore | SQL | Data | Chart | Export | Search results...

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER	Ticket_No	P_ID	SOURCE	DESTINATION	TRAIN_ID	RES_STATU
1	Davy Bansal	4	M	644534342	20001	1	Darbangha	Thane	11899	C
1	Davy Bansal	4	M	644534342	20005	1	Chennai	Thane	34521	C
2	Gurdeep Singh	5	M	983135265	20002	2	Pune	Thane	12138	C
3	Aryan Sharma	6	M	848567341	20003	3	Chennai	Darbangha	11862	C
4	Gajendra Pal	7	M	747567354	20004	4	Bathinda	Thane	12138	C
NULL	Tara Sutaria	1	F	999292939	NULL	NULL	NULL	NULL	NULL	NULL
107	Ashish Kumar	2	M	999999111	20006	107	Chennai	Bathinda	34521	C

Views

Views in SQL are kind of Virtual tables.A view also has rows and columns as they are in a real table in the Database.We can create a view by selecting fields from one or more tables present in the database.A view can either have all the rows and columns of a table or some specific based on a certain condition.

Advantages of views over tables:

-Views provide table security by restricting access to predetermined set of rows and columns.

-Views can join and simplify multiple tables into a single virtual table.

-Views can also be used to rename the columns without affecting the base table provided the number of columns in view must match the number of columns specified in the SELECT statement.Renaming helps to hide the actual column names of Base table.

-We can create multiple Views on the same table for different users.

CREATING VIEWS

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

```
CREATE VIEW view name AS

SELECT column1, column2.....

FROM table name

WHERE condition;
```

view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows

Creating views from single table

In this we creates a view named Passenger_View from Passenger table which will show only two columns of Passenger table. Syntax-

```
79
80    CREATE VIEW Passenger_View AS SELECT P_ID,P_Name FROM Passenger;
81    SELECT * FROM Passenger_View;
```

7:47:54 PM | 7:47:37 PM | 7:46:58 PM | 7:46:21 PM | 7:46:12 PM | 7:42:

Success
(6 rows) 0.4 s

Explore | SQL | Data | Chart | Export

P_ID	P_Name
1	Davy Bansal
2	Gurdeep Singh
3	Aryan Sharma
4	Gajendra Pal
106	Tara Sutaria
107	Ashish Kumar

Creating views from more than one table

In this we create a view named Full_Join_View from two tables named Passenger and Ticket. To create a view from multiple tables we have use Full outer join on the two tables and store the result in a view. Syntax-

```
CREATE OR REPLACE VIEW Full_Join_View AS SELECT Passenger.P_ID,Passenger.P_Name,Ticket.SOURCE,Ticket.DESTINATION FROM Passenger LEFT OUTER JOIN Ticket ON Passenger.P_ID=Ticket.P_I
UNION
SELECT Passenger.P_ID,Passenger.P_Name,Ticket.SOURCE,Ticket.DESTINATION FROM Passenger RIGHT OUTER JOIN Ticket ON Passenger.P_ID=Ticket.P_ID;

SELECT * FROM Full_Join_View;
```

7:30:22 PM | 7:30:17 PM | 7:28:54 PM | 7:28:32 PM | 7:26:27 PM | 7:26:05 PM | 7:23:55 PM | 7:23

Success
(6 rows) 0.3 s

Explore | SQL | Data | Chart | Export

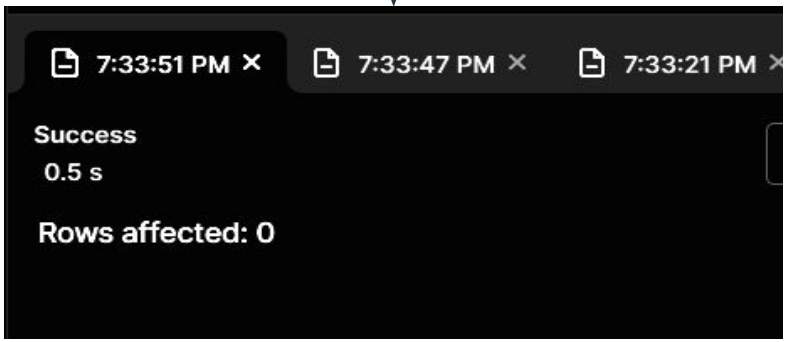
P_ID	P_Name	SOURCE	DESTINATION
1	Davy Bansal	Darbangha	Thane
2	Gurdeep Singh	Pune	Thane
3	Aryan Sharma	Chennai	Darbangha
4	Gajendra Pal	Bathinda	Thane
106	Tara Sutaria	Chennai	Thane
107	Ashish Kumar	Chennai	Bathinda

DELETING VIEWS

-This is used to delete the existing view.

Syntax- DROP VIEW view_name;

```
78    DROP VIEW Full_Join_View;
```



AFTER DROPPING VIEW

UPDATING VIEWS

This UPDATION CAN BE-

1. RENAME view name.
2. Inserting a COLUMN into a view.
3. Deleting COLUMN from a view
4. RENAME COLUMN names.
5. Inserting a ROW into a view.
6. Deleting a ROW from a view.
7. Updating a particular ROW in a view.

ADDING OR REMOVING COLUMNS IN A VIEW

We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.The same statement is also used for changing the name of a view, changing its column names.Query as follows:

```
122    CREATE OR REPLACE VIEW Passenger_View(ID,Name,Mobile) AS SELECT P_ID,P_Name,PH_NUMBER FROM Passenger;
123    SELECT * FROM Passenger_View;;
```

Success
(6 rows) 0.3 s

ID	Name	Mobile
1	Davy Bansal	644534342
2	Gurdeep Singh	983135265
3	Aryan Sharma	848567341
4	Gajendra Pal	747567354
106	Tara Sutaria	999292939
107	Ashish Kumar	999999111

COLUMN ADDED

Inserting a row in a view

1.We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View. The changes will also be made in the table itself also apart from the virtual table. The syntax for it is as follows:

```
87    INSERT INTO Passenger_View VALUES(108,'Karan Dua',937463512);
88    SELECT * FROM Passenger_View;
```

ROW INSERTED

EFFECT ON ACTUAL TABLE AFTER INSERTION IN VIEW

BEFORE INSERTION

```
141  SELECT * FROM Passenger;
```

Success
(6 rows) 0.3 s

Explore | SQL | Data | C

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Davy Bansal	4	M	644534342
2	Gurdeep Singh	5	M	983135265
3	Aryan Sharma	6	M	848567341
4	Gajendra Pal	7	M	747567354
106	Tara Sutaria	1	F	999292939
107	Ashish Kumar	2	M	999999111

AFTER INSERTION

```
INSERT INTO Passenger_View VALUES(108,'Karan Dua',937463512);
SELECT * FROM Passenger;
```

Success
(7 rows) 0.5 s

Explore | SQL | Data | Cl

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Davy Bansal	4	M	644534342
2	Gurdeep Singh	5	M	983135265
3	Aryan Sharma	6	M	848567341
4	Gajendra Pal	7	M	747567354
106	Tara Sutaria	1	F	999292939
107	Ashish Kumar	2	M	999999111
108	Karan Dua	NULL	NULL	937463512

ROW INSERTED IN ACTUAL TABLE ALSO

Deleting a row from a View

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

```
87    DELETE FROM Passenger_View WHERE P_ID-108;
88    SELECT * FROM Passenger_View;
```

Success
(6 rows) 0.5 s

Explore | SQL | Data | Chart

P_ID	P_Name	PH_NUMBER
1	Davy Bansal	644534342
2	Gurdeep Singh	983135265
3	Aryan Sharma	848567341
4	Gajendra Pal	747567354
106	Tara Sutaria	999292939
107	Ashish Kumar	999999111

ROW DELETED

EFFECT ON ACTUAL TABLE AFTER DELETION IN VIEW

BEFORE DELETION

```
141  SELECT * FROM Passenger;
```

Success
(7 rows) 0.5 s

Explore | SQL | Data | Cl

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Davy Bansal	4	M	644534342
2	Gurdeep Singh	5	M	983135265
3	Aryan Sharma	6	M	848567341
4	Gajendra Pal	7	M	747567354
106	Tara Sutaria	1	F	999292939
107	Ashish Kumar	2	M	999999111
108	Karan Dua	NULL	NULL	937463512

AFTER DELETION

```
DELETE FROM Passenger_View WHERE P_ID=108;
SELECT * FROM Passenger;
```

Success
(6 rows) 0.3 s

Explore | SQL | Data | C

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Davy Bansal	4	M	644534342
2	Gurdeep Singh	5	M	983135265
3	Aryan Sharma	6	M	848567341
4	Gajendra Pal	7	M	747567354
106	Tara Sutaria	1	F	999292939
107	Ashish Kumar	2	M	999999111

ROW DELETED FROM ACTUAL TABLE ALSO

Updating a row in a View

For updating a row in a view there is one condition i.e. the view must be an updatable view.

–Updatable view is a view which contains all the NOT NULL columns from the underlying table.

If so, we can update a row in a view which also affects the data on the actual table.

We can UPDATE a row in a View in a same way as we do in a table. We can use the UPDATE statement of SQL to UPDATE a row in a View. The changes will also be made in the table itself also apart from the virtual table. The syntax for it is as follows:

UPDATE view_name SET column1=___ , column2=___ , … WHERE condition;

BEFORE UPDATION

```
129   CREATE Train_View AS SELECT TRAIN_ID,TRAIN_NAME FROM Train;
130   SELECT * FROM Train_View;
```

Success
(6 rows) 0.2 s

Explore | SQL | Data

TRAIN_ID	TRAIN_NAME
34521	Amritsar Express
11899	Dehradun Mail
54889	Jammu Rajdhani
22046	NDLS JAN Shatabdi
11862	Panipat Superfast
12138	Punjab Mail

AFTER UPDATION

```
129   UPDATE Train_View SET TRAIN_NAME='Lucknow Mail' WHERE TRAIN_ID=11899;
130   SELECT * FROM Train_View;
```

Success
(6 rows) 0.2 s

Explore | SQL | Data

TRAIN_ID	TRAIN_NAME
34521	Amritsar Express
54889	Jammu Rajdhani
11899	Lucknow Mail
22046	NDLS JAN Shatabdi
11862	Panipat Superfast
12138	Punjab Mail

ROW UPDATED

EFFECT ON ACTUAL TABLE AFTER UPDATION IN VIEW

BEFORE UPDATION

```
136    SELECT * FROM Train;
```

Success
(6 rows) 0.3 s

Explore | SQL | Data | Chart

TRAIN_ID	TRAIN_NAME	STATION_ID
11862	Panipat Superfast	19
11899	Dehradun Mail	58
12138	Punjab Mail	15
22046	NDLS JAN Shatabdi	23
34521	Amritsar Express	11
54889	Jammu Rajdhani	30

AFTER UPDATION

```
UPDATE Train_View SET TRAIN_NAME='Lucknow Mail' WHERE TRAIN_ID=11899;
SELECT * FROM Train;
```

Success
(6 rows) 0.3 s

Explore | SQL | Data

TRAIN_ID	TRAIN_NAME	STATION_ID
11862	Panipat Superfast	19
11899	Lucknow Mail	58
12138	Punjab Mail	15
22046	NDLS JAN Shatabdi	23
34521	Amritsar Express	11
54889	Jammu Rajdhani	30

ROW UPDATED IN ACTUAL TABLE ALSO

SEQUENCES

A sequence is a object that generates a sequence of numeric values according to the specification with which the sequence was created.

The sequence of numeric values is generated in an ascending or descending order at a defined interval and can be configured to restart (cycle) when exhausted.

SYNTAX:

```
CREATE SEQUENCE sequence_name
   START WITH start_num
   INCREMENT BY increment_num
   MAXVALUE maximum_num | NOMAXVALUE
   MINVALUE minimum_num | NOMINVALUE
   CACHE cache_num | NOCACHE
   CYCLE | NOCYCLE
   ORDER | NOORDER;
```

SEQUENCES

P_ID_GENERATOR

For Passenger ID (P_ID)

TICKET_NO_GENERATOR

For Ticket_NO

P_ID_GENERATOR : Sequence for Generating values starting from 1

```
create sequence P_ID_GENERATOR
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 999999999
CACHE 10000
CYCLE;
```

```
Sequence created.
```

We initialize and get the values from the sequence using the
SEQUENCE_NAME.nextval -> Increments the sequence and return the value
SEQUENCE_Name.currval -> returns the latest value in sequence

```
INSERT INTO Passenger(P_NAME,SEAT_NO,P_ID,GENDER,PH_NUMBER) VALUES('Davy Bansal',1,P_ID_GENERATOR.nextval,'M',644534342);
INSERT INTO Passenger(P_NAME,SEAT_NO,P_ID,GENDER,PH_NUMBER) VALUES('Gurdeep Singh',2,P_ID_GENERATOR.nextval,'M',983135265);
INSERT INTO Passenger(P_NAME,SEAT_NO,P_ID,GENDER,PH_NUMBER) VALUES('Aryan Sharma',3,P_ID_GENERATOR.nextval,'M',848567341);
INSERT INTO Passenger(P_NAME,SEAT_NO,P_ID,GENDER,PH_NUMBER) VALUES('Gajendra Pal',4,P_ID_GENERATOR.nextval,'M',747567354);
INSERT INTO Passenger(P_NAME,SEAT_NO,P_ID,GENDER,PH_NUMBER) VALUES('Tara Sutaria',5,P_ID_GENERATOR.nextval,'F',999292939);
INSERT INTO Passenger (P_NAME,SEAT_NO,P_ID,GENDER,PH_NUMBER) VALUES('Ashish Kumar',6,P_ID_GENERATOR.nextval,'M',999999111);
```

P_ID Generator Sequence

OUTPUT : Passenger Table

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Davy Bansal	1	M	644534342
2	Gurdeep Singh	2	M	983135265
3	Aryan Sharma	3	M	848567341
4	Gajendra Pal	4	M	747567354
5	Tara Sutaria	5	F	999292939
6	Ashish Kumar	6	M	999999111

TICKET_NO_GENERATOR : Generates Unique Ticket No starting from Start Value = 1000

```
create sequence TICKET_NO_GENERATOR
START WITH 1000
INCREMENT BY 1
MINVALUE 1000
MAXVALUE 999999999
CACHE 10000
CYCLE;
```

```
Sequence created.
```

Generated Ticket No

OUTPUT: Tickets Table

TICKET_NO	P_ID	SOURCE	DESTINATION	TRAIN_ID	RES_STATUS
1000	1	Darbangha	Thane	11899	CONFIRMED
1001	2	Pune	Thane	12138	CONFIRMED
1002	3	Chennai	Darbangha	11862	WAITING
1003	4	Bathinda	Thane	12138	CONFIRMED
1004	5	Chennai	Bathinda	34521	WAITING

```
INSERT INTO Ticket VALUES (TICKET_NO_GENERATOR.nextval,1,'Darbangha','Thane',11899,'CONFIRMED');
INSERT INTO Ticket VALUES (TICKET_NO_GENERATOR.nextval,2,'Pune','Thane',12138,'CONFIRMED');
INSERT INTO Ticket VALUES (TICKET_NO_GENERATOR.nextval,3,'Chennai','Darbangha',11862,'WAITING');
INSERT INTO Ticket VALUES (TICKET_NO_GENERATOR.nextval,4,'Bathinda','Thane',12138,'CONFIRMED');
INSERT INTO Ticket VALUES (TICKET_NO_GENERATOR.nextval,5,'Chennai','Bathinda',34521,'WAITING');
```

ALTER SEQUENCE

A sequence can be altered and following properties can be changed:

- **Increment by**
- **Max Value**
- **Min Value**
- **Order**
- **CACHE**
- **CYCLE**

Altering sequence only affects, the future values in the sequence

EX: we want to **skip a P_ID** by changing the increment **factor to 2 (earlier was 1)**

```
Alter sequence P_ID_GENERATOR increment by 2;
```

Generating P_ID from Altered Sequence

```
INSERT INTO Passenger(P_ID,P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES(P_ID_GENERATOR.nextval,'Arjun Kumar',8,'F',600034342);
INSERT INTO Passenger(P_ID,P_NAME,SEAT_NO,GENDER,PH_NUMBER) VALUES(P_ID_GENERATOR.nextval,'Vladimir Putin',4,'M',644534342);
```

OUTPUT: Passenger Table

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Camila Cabello	8	F	600034342
2	Jennifer Lawrence	9	F	113135265
3	Jennifer Lopez	10	F	848567888
4	Charlie Puth	11	M	777567354
5	Davy Bansal	4	M	644534342
7	Arjun Kumar	29	F	600034342
9	Vladimir Putin	24	M	644534342

INDEXES

- Indexes are one of the most important **performance optimization** features of SQL
- Indexes are used by queries to find data from tables quickly. Indexes are created on tables and views. Index on a table or a view, is very similar to an index that we find in a book.
- The set of right indexes can drastically improve the performance of the query, and prevents the **Table Scan Problem**.
- Types of Indexes:
  - **Clustered :**
    - Order of rows in index determines/matches the actual physical order of data
    - Do not claim extra storage
  - **Non - Clustered :**
    - Order of rows in index does not match with the physical order of data
    - Stored separately, have pointers pointing to physical rows
  - **Unique :**
    - Enforce the unique constraint on key column(s)
  - **Duplicate :**
    - Duplicate indexes are those that exactly match the Key and Included columns
  - **Simple :** Include Single key Column
  - **Composite :** Includes more than one key Column

UNIQUE INDEX (Simple)

```
create unique index Unique_seat on Passenger(SEAT_NO asc);
```

Unique_seat Index

SEAT_NO	Pointer to Row
4	
5	
6	
7	
8	
9	
10	
11	

Passenger Table

	P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
▶	1	Camila Cabello	8	F	600034342
	2	Jennifer Lawrence	9	F	113135265
	3	Jennifer Lopez	10	F	848567888
	4	Charlie Puth	11	M	777567354
	5	Davy Bansal	4	M	644534342
	6	Gurdeep Singh	5	M	983135265
	7	Aryan Sharma	6	M	848567341
	8	Gajendra Pal	7	M	747567354
*	NULL	NULL	NULL	NULL	NULL

Use Case :

- To Ensure no passenger have duplicate seat

Composite and Unique

```
create unique index Unique_Contact on passenger(PH_NUMBER asc,P_ID asc);
```

Unique Index Enforces Unique constraint on the key column(s)

E.x - If we try to insert another passenger with same seat, an error is generated

```
INSERT INTO Passenger VALUES(
P_ID_GENERATOR.nextval,
'Berlin',
11,
'M',
777567354);
```

Error Code: 1062. Duplicate entry '11' for key 'passenger.Unique_seat'

Passenger Table

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Camila Cabello	8	F	600034342
2	Jennifer Lawrence	9	F	113135265
3	Jennifer Lopez	10	F	848567888
4	Charlie Puth	11	M	777567354
5	Davy Bansal	4	M	644534342
6	Gurdeep Singh	5	M	983135265
7	Aryan Sharma	6	M	848567341
8	Gajendra Pal	7	M	747567354
NULL	NULL	NULL	NULL	NULL

Duplicate Index (Simple)

Duplicate indexes are those that exactly match the included columns and order, but have different name and claim their own storage.

E.x: we create an another Index which optimizes the search queries involving SEAT_NO

```
create unique index Unique_Seat_Passenger on Passenger(SEAT_NO asc);
```

Similar to Unique_Seat Index we created earlier

Output:

```
1831 Duplicate index 'Unique_Seat_Passenger' defined on the table 'hi0.passenger'.
```

Unique_Seat

SEAT_NO	Pointer to Row
4	
5	
6	
7	
8	
9	
10	
11	

Passenger Table

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Camila Cabello	8	F	600034342
2	Jennifer Lawrence	9	F	113135265
3	Jennifer Lopez	10	F	848567888
4	Charlie Puth	11	M	777567354
5	Davy Bansal	4	M	644534342
6	Gurdeep Singh	5	M	983135265
7	Aryan Sharma	6	M	848567341
8	Gajendra Pal	7	M	747567354
NULL	NULL	NULL	NULL	NULL

Unique_Seat_Passenger

Pointer to Row	SEAT_NO
	4
	5
	6
	7
	8
	9
	10
	11

Database should never have Duplicate Indexes because they affect the performance as each duplicate index needs to be modified after insertion,deletion or update operation and takes extra storage

```
drop index Unique_Seat_Passenger on Passenger;
```

Simple Index

```
create index P_Name_Optimize on Passenger(P_NAME asc);
```

Use Case: Optimize Queries involving Passenger Names

P_NAME_Optimize

P_NAME	Pointer to row
Aryan Sharma	
Camila Cabello	
Charlie Puth	
Davy Bansal	
Gajendra Pal	
Gurdeep Singh	
Jennifer Lawrence	
Jennifer Lopez	

P_ID	P_NAME	SEAT_NO	GENDER	PH_NUMBER
1	Camila Cabello	8	F	600034342
2	Jennifer Lawrence	9	F	113135265
3	Jennifer Lopez	10	F	848567888
4	Charlie Puth	11	M	777567354
5	Davy Bansal	4	M	644534342
6	Gurdeep Singh	5	M	983135265
7	Aryan Sharma	6	M	848567341
8	Gajendra Pal	7	M	747567354
NULL	NULL	NULL	NULL	NULL

COMPOSITE INDEX

Creating Index on Fair Table for CLASS and FARE Column
Use Case : Optimize queries for Class and Fare

```
create index Fair_by_class on fare_table(Class asc,Fare asc);
```

Fair_by_class Index

CLASS	Fare	Pointer to Row
First	250	
First	660	
Second	100	
Third	500	

Fair Table

	CLASS	Fare	Ticket_No
▶	FIRST	660	20001
	SECOND	100	20002
	FIRST	250	20003
	THIRD	500	20004
*	NULL	NULL	NULL

PL/SQl

PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements.All the statements of a block are passed to oracle engine all at once which increases processing speed and decreases the traffic.

Disadvantages of SQL:

- SQL doesn’t provide the programmers with a technique of condition checking, looping and branching.
- SQL statements are passed to Oracle engine one at a time which increases traffic and decreases speed.
- SQL has no facility of error checking during manipulation of data.

Features of PL/SQL:

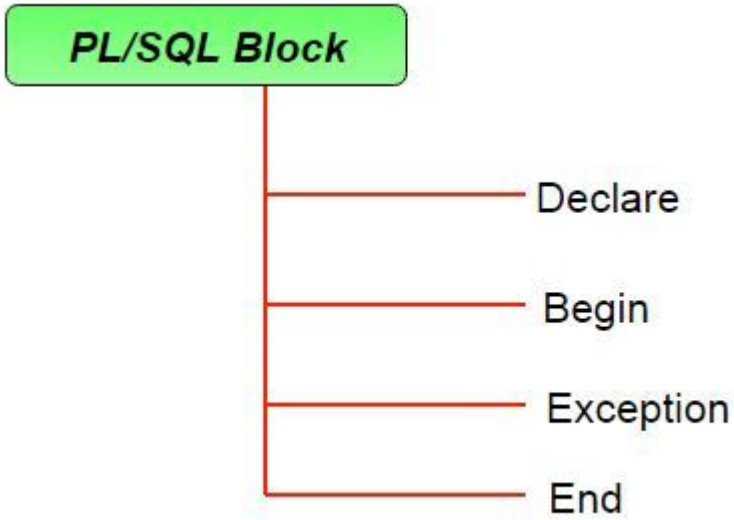
1. PL/SQL is basically a procedural language, which provides the functionality of decision making, iteration and many more features of procedural programming languages.
2. PL/SQL can execute a number of queries in one block using single command.
3. One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
4. PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.
5. Applications written in PL/SQL are portable to computer hardware or operating system where Oracle is operational.
6. PL/SQL Offers extensive error checking.

Differences between SQL and PL/SQL:

SQL	PL/SQL
SQL is a single query that is used to perform DML and DDL operations	PL/SQL is a block of codes that used to write the entire program blocks/ procedure/ function, etc.
It is declarative, that defines what needs to be done, rather than how things need to be done.	PL/SQL is procedural that defines how the things needs to be done.
Execute as a single statement.	Execute as a whole program.
Mainly used to manipulate data.	Mainly used to create application.
Cannot contain PL/SQL code in it.	It is an extension of SQL,so it can contain SQL inside it.

Structure of PL/SQL Block:

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.



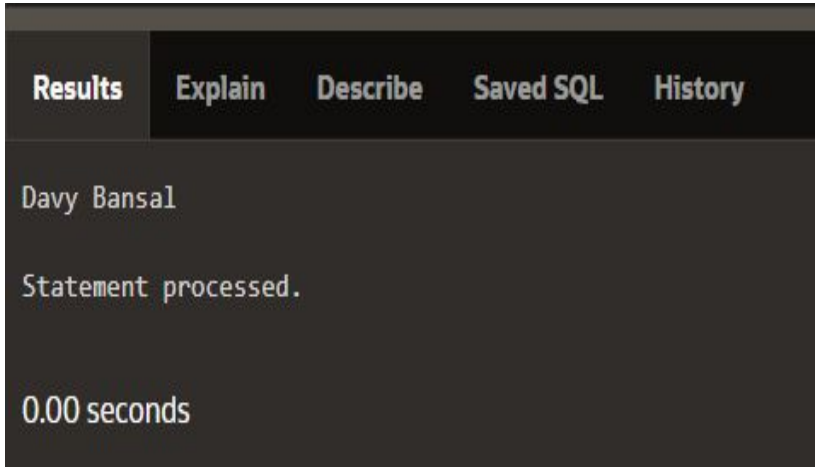
PROCEDURES IN PL/SQL

A stored procedure in PL/SQL is a series of SQL statements which can be stored in the database catalogue and can be reused.. A procedure can be thought of as a function or a method which may or may not returns a value.ALL statements of a block are passed to Oracle engine all at once which increases processing speed and decrease **traffic.**

SYNTAX:

```
CREATE OR REPLACE PROCEDURE proc_name(ID INT) IS
temp VARCHAR(20);
BEGIN
    SELECT P_NAME INTO temp FROM Passenger WHERE P_ID=ID;
    dbms_output.put_line(temp);
END;

-- CALLING OF PROCEDURE
BEGIN
proc_name(1);
END;
```

OUTPUT



Advantages:

- They result in performance improvement of the application. If a procedure is being called frequently in an application in a single connection, then the compiled version of the procedure is delivered.
- They reduce the traffic between the database and the application, since the lengthy statements are already fed into the database and need not be sent again and again via the application.
- They add to code reusability, similar to how functions and methods work in other languages such as C/C++ and Java.

Disadvantages:

- Stored procedures can cause a lot of memory usage.
- MySQL does not provide the functionality of debugging the stored procedures.

CURSOR

Def: Cursors are used to store Database Tables. There are two types of cursors

- **Implicit Cursors: Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.**

Explicit Cursors: Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

```
DECLARE
   pnr Passenger.PNR_No%type;
   name Passenger.Name%type;
   CURSOR c_Passenger is
      SELECT PNR_No,Name FROM Passenger;
BEGIN
   OPEN c_Passenger;
   LOOP
   FETCH c_Passenger into pnr,name;
      EXIT WHEN c_Passenger%notfound;
      dbms_output.put_line(pnr||' '|| name);
      INSERT INTO Cursor_tested VALUES(name);
   dbms_output.put_line(c_Passenger%ROWCOUNT || ' CURSOR TESTING ');
   END LOOP;
   CLOSE c_Passenger;
    EXCEPTION
    WHEN no_data_found THEN
        dbms_output.put_line('No such passenger!');
    WHEN TOO_MANY_ROWS THEN
        dbms_output.put_line('Same PNR number exist!');
    WHEN others THEN
        dbms_output.put_line('Unexpected Error Found!');
END;
/
```

A Cursor is being used to count no of Passengers and output it to the consol by fetching it from table Passenger.

OUTPUT:

```
1000 Arpit
1 CURSOR TESTING
2000 Ashish
2 CURSOR TESTING
3000 Gurdeep
3 CURSOR TESTING
4000 Davy
4 CURSOR TESTING
```

Packages

Packages are schema objects that group logically related PL/SQL types, variables,functions,procedures,cursors and subprograms under a single unit.

A package has two mandatory parts-

- Package Specification or Declaration
- Package Body or Definition

Package Specification-The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

Syntax-

```
CREATE OR REPLACE PACKAGE package_name IS
    PROCEDURE proc_name(parameters);
    FUNCTION func_name(parameters) RETURN type;
END package_name;
```

```
CREATE or REPLACE PACKAGE edit_reserve IS
PROCEDURE update_Reserve(Pnr_no INT);
PROCEDURE deleteByPnr(v_Pnr INT);
End edit_reserve;
```

Package Body- The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package. The CREATE PACKAGE BODY Statement is used for creating the package body.

SYNTAX-

```
CREATE OR REPLACE PACKAGE BODY package_name IS

    PROCEDURE proc_name (parameters) IS

     BEGIN

          Executable code

     END proc_name;

END package_name;
```

```
CREATE OR REPLACE PACKAGE BODY edit_reserve IS
PROCEDURE deleteByPnr(v_Pnr INT) IS
BEGIN
    DELETE FROM reserve WHERE PNR_NO = v_Pnr;
End deleteByPnr;



 PROCEDURE update_Reserve(Pnr_no INT) IS
    BEGIN
        UPDATE Reserve SET Status='N' WHERE PNR_NO=Pnr_no;
        COMMIT;
    END update_Reserve;
End edit_reserve;
```

Triggers

Our trigger part works when someone updates cancellation table which means when someone request for cancellation of tickets..

generally

Trigger is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes.

```
--AFTER CANCEL THE TICKET THAT PNR_No WILL BE
--DELETED FROM RESERVED TABLE AND ALSO THIS PROCESS
--SAVED IN status_audit TABLE

CREATE OR REPLACE TRIGGER cancel_after_insert
AFTER INSERT ON Cancellation
FOR EACH ROW
BEGIN
    DELETE FROM Reserve
    WHERE PNR_No = :NEW.PNR_No;

    INSERT INTO status_audit
    VALUES(
        :NEW.PNR_No,
        :NEW.Journey_Date,
        :NEW.Contact_No,
        'CANCELLED SUCCESSFULLY'
    );
END
```

Exception Handling

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition. There are two types of exceptions −

- System-defined exceptions
- User-defined exceptions

System Defined Exception

These exceptions are predefined in PL/SQL which get raised WHEN certain database rule is violated.

System-defined exceptions are further divided into two categories:

- Named system exceptions.
- Unnamed system exceptions.

Named system exceptions: They have a predefined name by the system like ACCESS_INTO_NULL, DUP_VAL_ON_INDEX, LOGIN_DENIED etc.

Unnamed system exceptions:Oracle doesn't provide name for some system exceptions called unnamed system exceptions.

SYSTEM DEFINED EXCEPTIONS

```
DECLARE
   pnr Passenger.PNR_No%type;
   name Passenger.Name%type;
   CURSOR c_Passenger is
      SELECT PNR_No,Name FROM Passenger;
BEGIN
   OPEN c_Passenger;
   LOOP
   FETCH c_Passenger into pnr,name;
      EXIT WHEN c_Passenger%notfound;
      dbms_output.put_line(pnr||' '|| name);
      INSERT INTO Cursor_tested VALUES(name);
   END LOOP;
   dbms_output.put_line(c_Passenger%ROWCOUNT || ' CURSOR TESTING ');
   CLOSE c_Passenger;
    EXCEPTION
    WHEN no_data_found THEN
        dbms_output.put_line('No such passenger!');
    WHEN TOO_MANY_ROWS THEN
        dbms_output.put_line('Same PNR number exist!');
    WHEN others THEN
        dbms_output.put_line('Unexpected Error Found!');
END;
/
```

Results | Explain | Describe | Saved SQL | History

```
1000 Arpit
3000 Ashish
4000 Davy
5000 Gurdeep
2000 Gajendra
5 CURSOR TESTING

1 row(s) inserted.


0.09 seconds
```

User defined exceptions :

This type of users can create their own exceptions according to the need and to raise these exceptions explicitly raise command is used.

```
-- truncate table reserve;
-- select *from Reserve;
-- drop trigger Invalid_Date;
CREATE OR REPLACE TRIGGER EXCEPTION_TRIG
BEFORE DELETE OR INSERT OR UPDATE ON reserve
FOR EACH ROW
DECLARE
    available_seats reserve.No_of_seats%type;
    pnr_num reserve.PNR_NO%type;
   EMPTY_SEAT_NOT_FOUND EXCEPTION;
 INVALID_PNR_FOUND EXCEPTION;
BEGIN
```

```
    SELECT No_of_seats INTO available_seats FROM (
        SELECT * FROM reserve ORDER BY PNR_NO DESC
    ) WHERE ROWNUM = 1;

    SELECT PNR_NO INTO pnr_num FROM (
        SELECT * FROM reserve ORDER BY PNR_NO DESC
    ) WHERE ROWNUM = 1;

--     dbms_output.put_line(pnr_num);

  IF available_seats<=0 THEN
      RAISE EMPTY_SEAT_NOT_FOUND;
   END IF;
  IF not(pnr_num>999 and pnr_num<10000) THEN
      RAISE INVALID_PNR_FOUND;
   END IF;
```

```
   IF not(pnr_num>999 and pnr_num<10000) THEN
       RAISE INVALID_PNR_FOUND;
    END IF;
EXCEPTION
    WHEN EMPTY_SEAT_NOT_FOUND THEN
    dbms_output.put_line('NO empty seat found');
    WHEN INVALID_PNR_FOUND THEN
    dbms_output.put_line('Please Enter a Valid PNR');
END;
/
```

```
139  DECLARE
140     pnr Passenger.PNR_No%type;
141     name Passenger.Name%type;
142     CURSOR c_Passenger is
143        SELECT PNR_No,Name FROM Passenger;
144  BEGIN
145     OPEN c_Passenger;
146     LOOP
147     FETCH c_Passenger into pnr,name;
148        EXIT WHEN c_Passenger%notfound;
149        dbms_output.put_line(pnr||' '|| name);
150        INSERT INTO Cursor_tested VALUES(name);
151     END LOOP;
152     dbms_output.put_line(c_Passenger%ROWCOUNT || ' CURSOR TESTING ');
153     CLOSE c_Passenger;
154      EXCEPTION
155      WHEN no_data_found THEN
156         dbms_output.put_line('No such passenger!');
157      WHEN TOO_MANY_ROWS THEN
158         dbms_output.put_line('Same PNR number exist!');
159      WHEN others THEN
160         dbms_output.put_line('Unexpected Error Found!');
161  END;
162  /
```

Results | Explain | Describe | Saved SQL | History

```
1000 Arpit
3000 Ashish
4000 Davy
5000 Gurdeep
2000 Gajendra
5 CURSOR TESTING

1 row(s) inserted.


0.09 seconds
```

While Inserting data, We have made a trigger to check whether PNR_NO is valid or not and also is seats are available or not. If any of the above condition fails then we raise exception

```
insert into reserve values(50000,'04-02-2032',4,'Tarnaka',1234123412,'Y');
```

Results | Explain | Describe | Saved SQL | History

```
Please Enter a Valid PNR

1 row(s) inserted.
```

0.01 seconds


thanks