**Report DBMS**

**Team Details**

1. Meet Sheladiya - 19BIT076
2. Renish Jagani – 19BIT108
3. Rishit Samariya – 19BIT109
4. Rishabh Patel – 19BIT141
5. Ujash Thakkar – 19BIT142
6. Ananya Khandelwal - 19BIT144

**WORK DONE TILL 28/02/2021**

1. File system vs. DBMS (Experiment - 2)

2. 24 Tables of Stadium Seat Booking Management system. (Experiment - 2)

3. Relational model of Stadium Seat Booking Management system. (Experiment – 2)

4. ER model of Stadium Seat Booking Management system. (Experiment – 3)

5. Installation of MySQL Software Relational Database Management System. (Experiment – 5.1)

6. Relational Algebra queries for the ABC. Clearly write the definition as well as

Relational algebra queries for each. (Experiment – 4)

1. Introduction to SQL, DDL, DML, DCL, database and table creation, alteration,

Defining Constraints, primary key, foreign key, and unique, not null, check, IN operator. (Experiment – 5.2)

1. Study and use of inbuilt SQL functions - aggregate functions, Built-in functions Numeric, date, string

Functions. (Experiment – 5.3)

1. Study, write and use the set operations, sub-queries, correlated sub-queries in SQL Study and use of group by, having, order by features of SQL. (Experiment – 5.4)
2. Study and use of group by, having, order by features of SQL. (Experiment – 5.5)
3. Study different types of join operations, Exist, Any, All and relevant features of SQL.

(Experiment – 5.6)

1. Study and implement the assignment topic from the Embedded SQL, PL SQL Concepts, Cursors, Stored Procedures, Stored Function, Triggers, Tuple relational calculus, domain relational calculus. (Experiment – 5.7)

**Assignment – 2**

In Assignment-1 we have stated the limitations of the file system in comparison to DBMS by pointing out the differences in the aspects based on **Data Inconsistency, Data Redundancy, Atomicity Problem, Data Security, Data Integrity and Data Isolation**. Each member described the topic assigned to them and gave one example related to the project of each topic.

1. **Data Inconsistency**

|  |  |
| --- | --- |
| **FILE SYSTEM** | **DBMS** |
| **Description-**  Data Inconsistency in file system means that different files may contain different details of a particular object or person. The retrenchment actually leads to inconsistencies. When the same data is stored in multiple locations, flexibility is possible. | **Description: -**  An Inconsistency in database occurs when the configuration file is marked as such after Ingres encountered a problem while processing the database. This leaves the database inaccessible and unusable until the problem is resolved. Data  Inconsistency is a situation where many tables within data deal with the same data but can find it in different inputs. |
| **Example 1 -** Data Inconsistency occurs in file system when Booking a Seat for stadium if any spectators has record of 2 different phone number then system for that data will be unable to identify seat number which is allocated for that spectator, while DBMS would not allow to book the seat. | |
| **Example 2 -** If a person has book seat for two consecutive scheduled matches and have entered the same data for both matches if any one of the match gets cancelled then file system might not give correct message to that person but in DBMS has separate relation between two matches. | |
| **Example 3 -** Data Inconsistency can impel if a spectator orders meal for two different timings while booking seat with same data then file system would not respond because file system could not identify that which order is cancelled but DBMS database has unique relation with different input tables. | |
| **Example 4 -** If a person has initially booked a seat for match and entered data like name, phone number and later book the another seat and entered the same data as previous data then in file system it will show error because if any updating happens system would not recognise that in which ticket updating had happened but in DBMS an foreign key is created where the table of one input is related to the other table. | |
| **Example 5 -** While booking seat If any spectator has his home address at two different data storage location it will occur an error in DBMS because if that spectator moved to some other city and then change the address in one of the location but the other location has previous same address so it will lead from data inconsistency to data redundancy. | |
| **Example 6 -** Data Inconsistency constrains can happens if audience from the different location but same street address name books seat file system could not process the further but in DBMS attribute with same value have unique relations and they are bound with different input tables. | |

1. **Data Redundancy**

|  |  |
| --- | --- |
| **FILE SYSTEM** | **DBMS** |
| **Description: –**  In file system, same piece of data can occur in multiple tables through various inputs and this situation is known as data redundancy which is highest in File System. | **Description: –**  In DBMS Data redundancy is lowest, the data is not repeated more than once. |
| **Example 1** – A redundancy situation can occur in file system where two persons  Booked the same seat for the same match, while DBMS won’t allow the booking of same seat again by the later person. | |
| **Example 2** - Two persons with same name and surname applied for booking the VIP seats, in File system it will create redundancy but in DBMS it will bifurcate the two with unique booking ID. | |
| **Example 3** - File system will create redundancy when more than one person having the same street address will book the tickets, but DBMS will reduce the redundancy by separating streets in one table and then just reference the correct Foreign key to the street table. | |
| **Example 4** – If more than one person choses to order for the same meal during the match interval, the file system will create redundancy and it will ultimately lead to data corruption i.e., the food staff will not be able to distinguish the orders of the audience, on the other hand DBMS will reduce the redundancy by making separate table for meal and connect the foreign key to the table. | |
| **Example 5** – Data redundancy is bound to take place in file system when all those who opted for a jersey of the same team to wear at the match to cheer up the players and it will lead to decrease in efficiency of work flow of distributing the jerseys while this will not be the case in  DBMS. | |
| **Example 6** – The size of the database will be unnecessarily increased in the File System when the data of the audience from same city will be stored, while DBMS will segregate the data by making different tables and interconnecting with the help of keys. | |

1. **Atomicity Problems**

|  |  |
| --- | --- |
| **FILE SYSTEM** | **DBMS** |
| **Description: –**  Atomicity is the guarantee that a series of operations either succeed or fail together or refers to “All or nothing”. Which means either all the operations in a transaction executes or none. | **Description: –**  Databases often have features that allow a series of operations to be committed at once as a single transaction there is only succeed in this. |
| **Example 1** – An Atomicity situation can occur in file system where during the buying of ticket if we buy stadium ticket and amount has deducted from account and from bank’s summary account and amount will credited to the seller. If this all not happen as an atomic transaction and there is failure during transaction and there system are not showing the amount either it is receive or not, then we will not able to stadium ticket and the amount of ticket will be also reduced. Whether in DBMS case the transaction will be place easily and if failure occur then also there system will show the amount. | |
| **Example 2**- If we prebook the seat with some beverages and snacks it will store in ticket counter file system and it will see by food counter for snacks and beverages for which seats but if the seats are booked and there is failure in system during sharing of data, due to which it will be unable to know food counter which food to whom. It is atomicity issue in file system but in DBMS it will be easily access the data after failure to. | |
| **Example 3** – If the same person book ticket for X, Y, Z for one executive seats and for others VIP lounge the data was store by the ticket counters file system and the ticket are given to person but during the match there is failure in system during sharing of data and there is no data for VIP lounge persons they will be not allowed to seat if there data don’t show it is atomicity problem but in the DBMS during the failure the data will not vanish at all. | |
| **Example 4 –** File system will create atomicity problem during the home delivery of tickets if we book the tickets by phone call and the money has been credited in there account they deliver the ticket by post but there is failure in the system during the ticket seller and post office the data will be not seen by post office because of the failure in file system and ticket will be unable to receive at home but in the DBMS the data will be not erase during the failure and it will easily access by the post office. | |
| **Example 5-** The atomicity problem also occur, during the file system while buying the food during the match with table service the money was paid for table service food but during the failure in system data was lost and money was paid for food with table service but it only show for food not table service but in the DBMS the data will be stored in the server and during failure also it will not erase. | |
| **Example 6-** While the buying the supporting team costume it will lead to atomicity problem in file system. The amount of costume was paid Rs.1000, and amount was deducted from the buyer account and it will show to the costume counter but during the transaction of money there is the failure in there system it will lead big deal in file system and after the failure the system will not show the payment is done or not or the money has credited in there account or not but the money will be deducted from buyers account. The counter will not give the receipt too. But in DBMS the data will be stored in server and it will be easily access after the failure from the server. And they will show the amount credited and we will get receipt. | |

1. **Data Security**

|  |  |
| --- | --- |
| **FILE SYSTEM** | **DBMS** |
| **Description** - File systems provide less security than DBMS and also possess the less security enchasing elements as compared to DBMS. | **Description** - DBMS has more security mechanisms as compared to file system. It contains range of security controls designed to protect the Database of system. |
| **Example- 1** To access the database of customers purchasing tickets online we can use security measure in DBMS that include verification while accessing data and allows only authenticated users to access the sensitive database. | |
| **Example-2** DBMS provides with features such as access control unlike of File System that includes security mechanisms in a database management system to grant protection against unauthorized access. | |
| **Example-3** Sensitive database of customers trying to purchase the Stadium ticket online can be protected using DBMS. | |
| **Example-4** Datasets such as Seat Number, Payments, and Personal records can be easily backed up and recovered in DBMS in case of data loss. | |
| **Example-5** Unauthorized access of customer’s database by hackers is more prominent in the case of File System than DBMS. | |
| **Example-6** Techniques such as encryption can be used in DBMS to protect sensitive information of ticket buyers. | |

1. **Data Integrity**

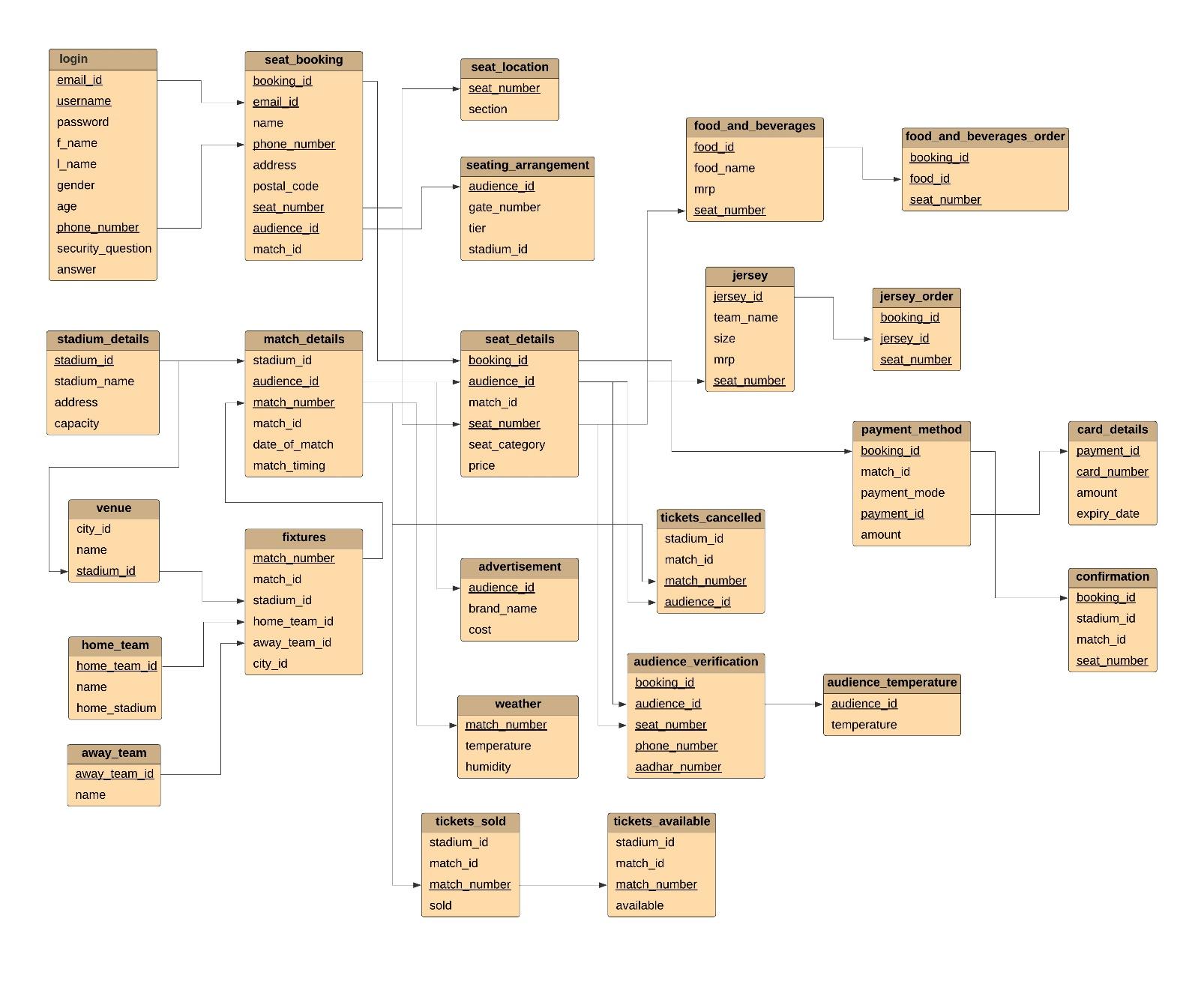
|  |  |
| --- | --- |
| **FILE SYSTEM** | **DBMS** |
| **Description: -**  There will be times where it is essential to add certain constraints to the data before entering it into the database. No procedure is provided by the file system to verify these constraints automatically. | **Description: -**  Whereas by imposing user-defined data constraints on its own, DBMS preserves data integrity. |
| **Example 1** — While booking a stadium seat, a user could mistakenly attempt to enter a date field with a phone number. It would prevent the user from making these errors if the device enforces data integrity. | |
| **Example 2** — A person attempts to input a phone number in an incorrect format while entering his personal details for booking a stadium seat. | |
| **Example 3** — In the application of stadium seat booking system, an error tries to erase wrong records. | |
| **Example 4** — While booking a seat, a user might enter wrong details and attempts to erase a record from a table, but as part of a connection, another table cites the record. | |
| **Example 5** — When there is already a foreign key in a similar table pointing to that number, a user attempts to change the primary key value. | |
| **Example 6** — While entering the date of the match, user attempts beyond an appropriate range to enter a date. | |

1. **Data Isolation**

|  |  |
| --- | --- |
| **FILE SYSTEM** | **DBMS** |
| **Description: -**  It is often difficult to find information stored in different files in different departments. Sharing information from multiple, separate files is a complex process and usually requires computer programming knowledge. | **Description: -**  The word ‘Isolation’ means separation. In DBMS, Separation is a data domain where no data should affect another and can appear simultaneously. In short, working on one database should start when working on the first database ends. |
| **Example 1** - Data Isolation occurs when a person is booking a seat uses payment system, they actually create a sub-account with the same permissions as everyone else, but they exist differently across the entire network. | |
| **Example 2** - While booking a seat when a person's financial Transaction 2 changes the row, but does not make any changes. Activity 1 and read the unintended details. Now if Transaction 2 reverses its changes already read by Transaction 1 or updates different changes to the database, viewing the data may be incorrect in Transaction 1 records. | |
| **Example 3** - If a person1 while booking seats issues a transaction against at the same time that person2 issues a different transaction, both transactions should operate independently. Database should either perform person1′s entire transaction before executing person2’s or vice-versa. This prevents person1’s transaction from reading intermediate data produced as a side effect of part of person2’s transaction that will not eventually be committed to the database. | |
| **Example 4** - Data Isolation in this situation occurs if a person buys ticket and books seat and another person trying to book the same seat then the one whose transaction completes first the seat allocate to that person because in DBMS both activities are happening separately. | |
| **Example 5** – while booking a seat if person wants to books VIP seats but in file system doesn’t specify that which one is VIP and which one is Executive lodge and file system books the Executive lodge but in  DBMS specification between two distinct label seat are mentioned. | |
| **Example 6** – A person wants to books a specific team material but when that person receive other team material then data isolation occurs in file system but in DBMS data of one table would not affect the data of other table. | |

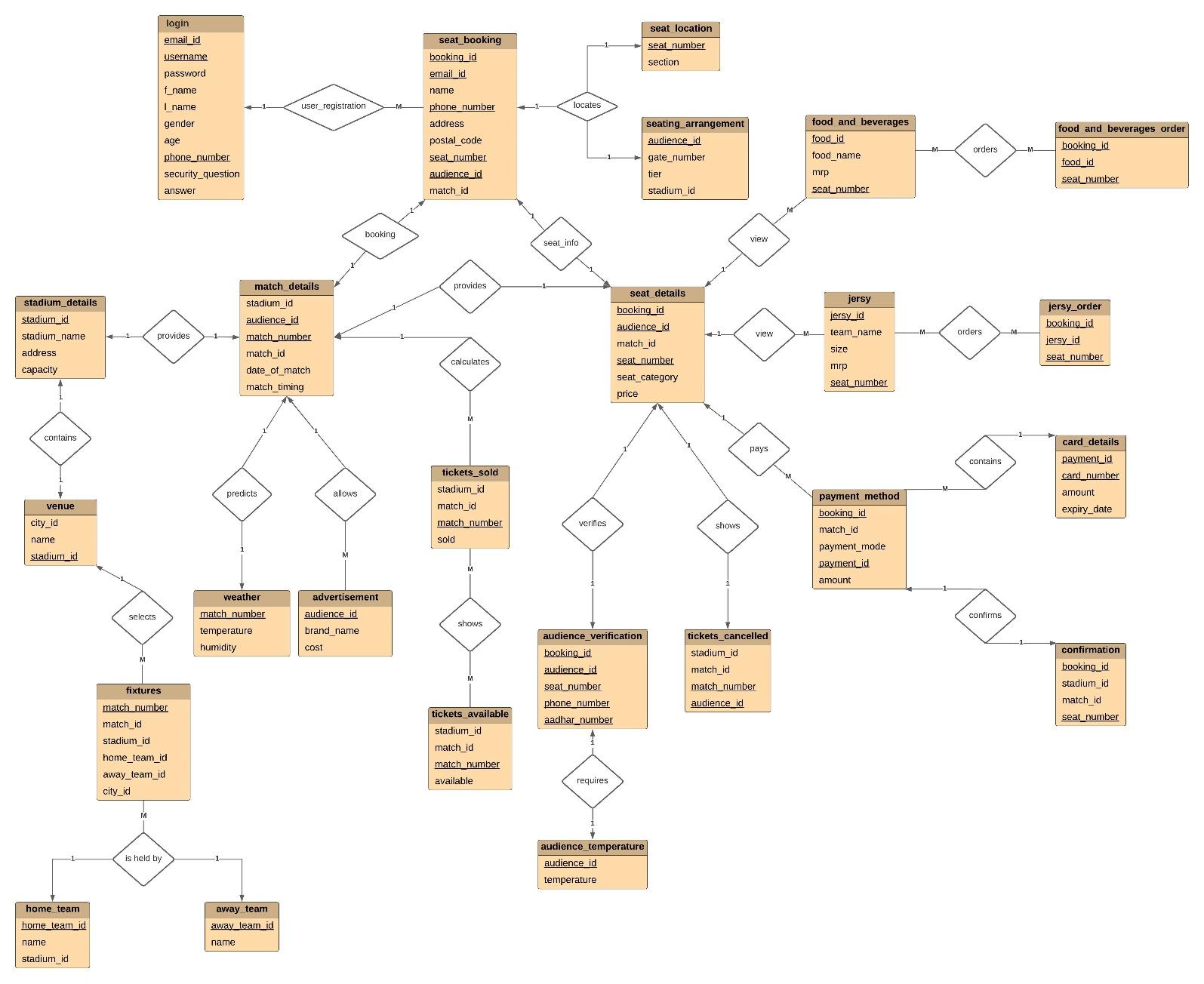
**Assignment - 2**

In Assignment-2 the assigned topic is relational model and it is group assignment so we have divided 4 tables for every member in our team and we have created Relational model in **lucid chart** and each member has given different entities and their attributes. In that we have created **primary key** and **foreign keys** and made relation between them In Relational model for Stadium Seat Booking management system we have shown relations and underlined primary key and made **relation** between them and we have finally done our group assignment - 2 on 21st Feb.

****

**Assignment – 3**

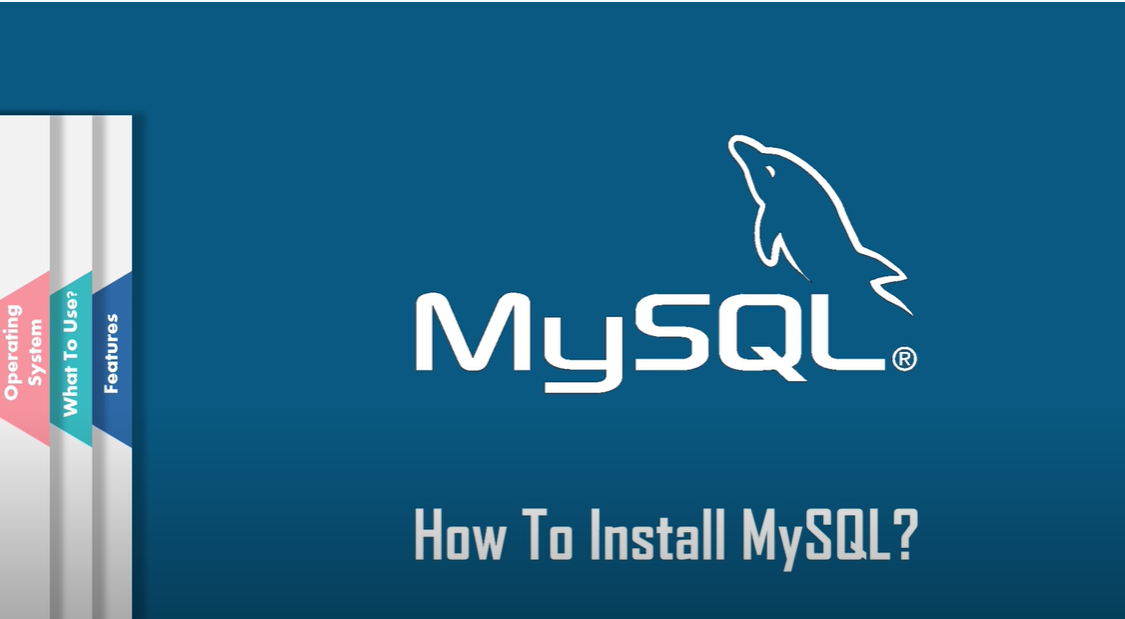
In Assignment-3 we have formed **Entity Relational Model** by establishing the relations **(One-to-One, One-to-Many, Many-to-One, Many-to-Many)** between different entities. Each entity in entity set is containing at least one relationship in relationship set. The **Primary key** and **foreign key** are underlined in the diagram itself. This model was formed in lucid chart and everyone contributed in it.

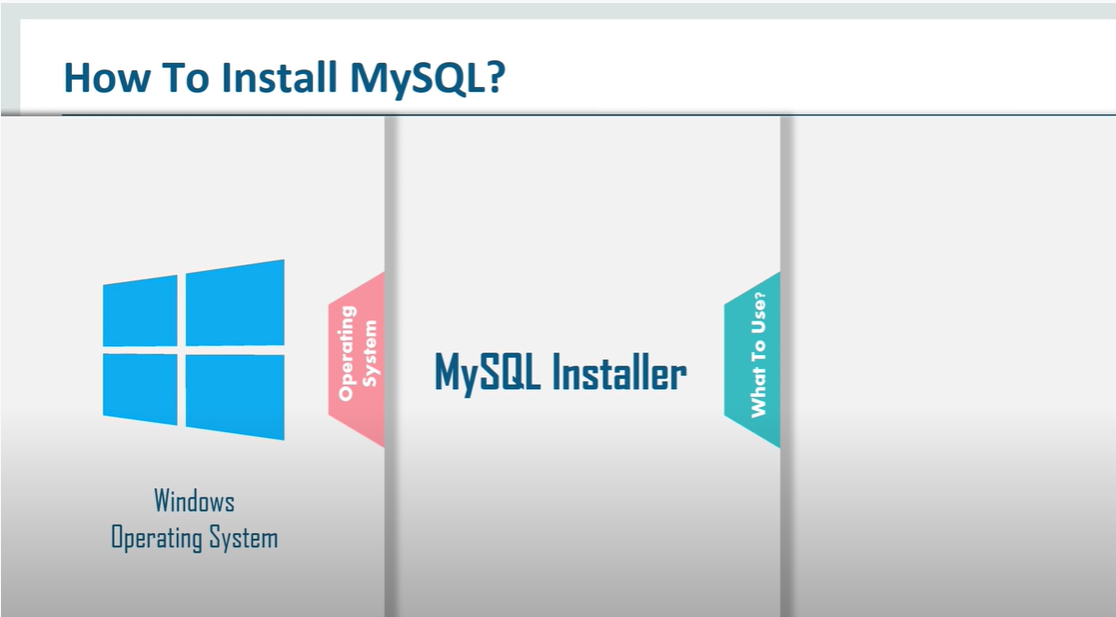


**Assignment – 5.1**

**Topic:- Installation of MySQL Relational Database Management System**

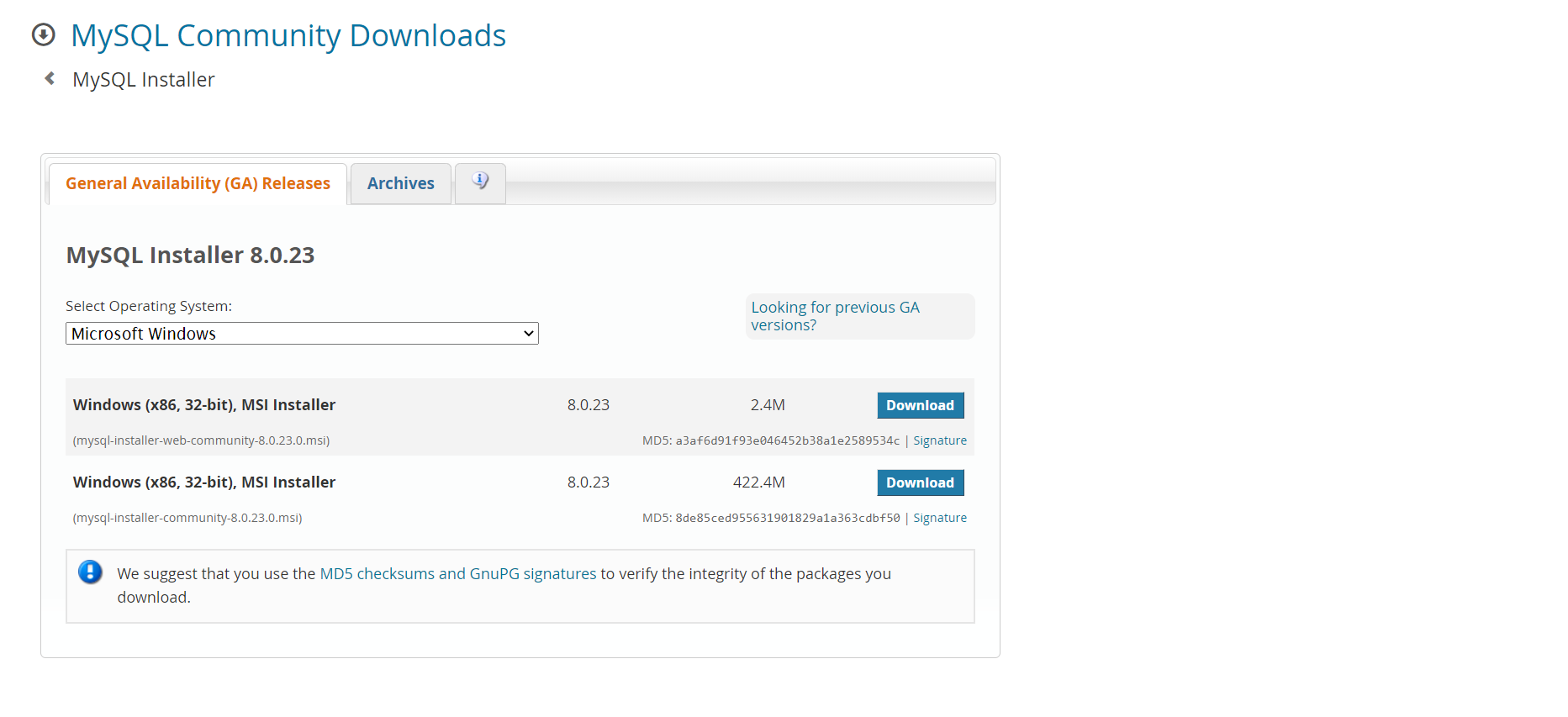
**Objective:-** To learn basics of MySQL software that can be used for SQL Query evaluations.



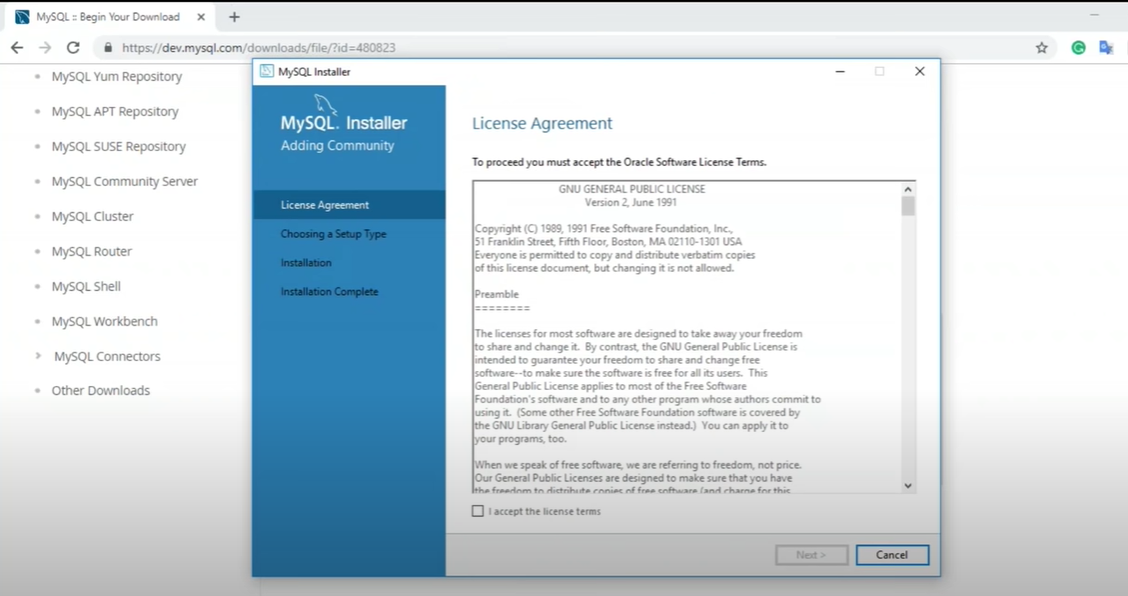


**Follow the Steps Below:-**

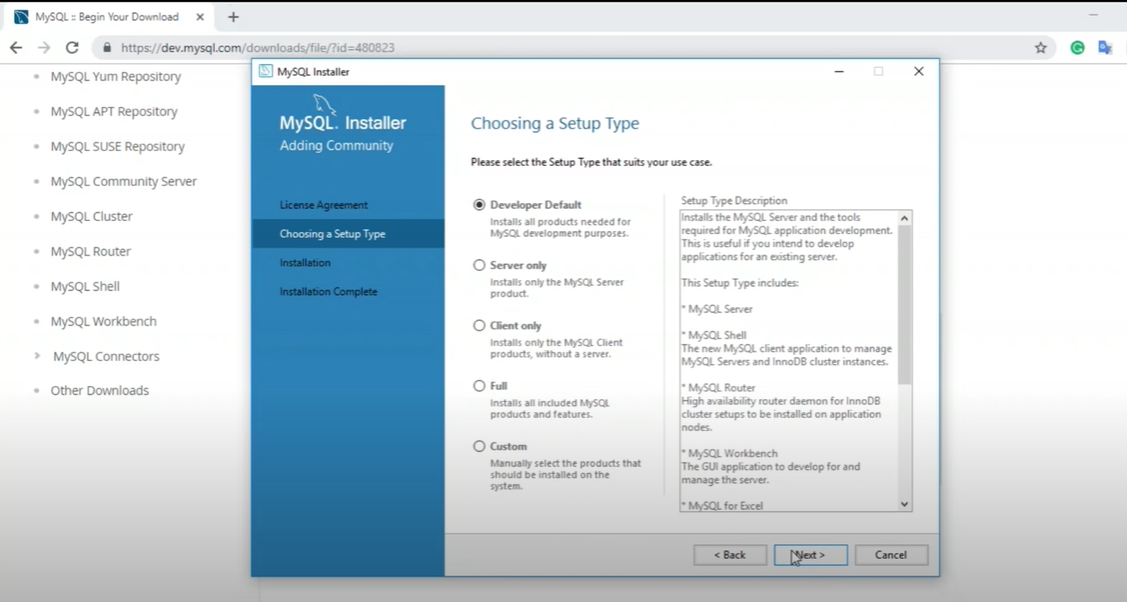
**STEP: - 1**



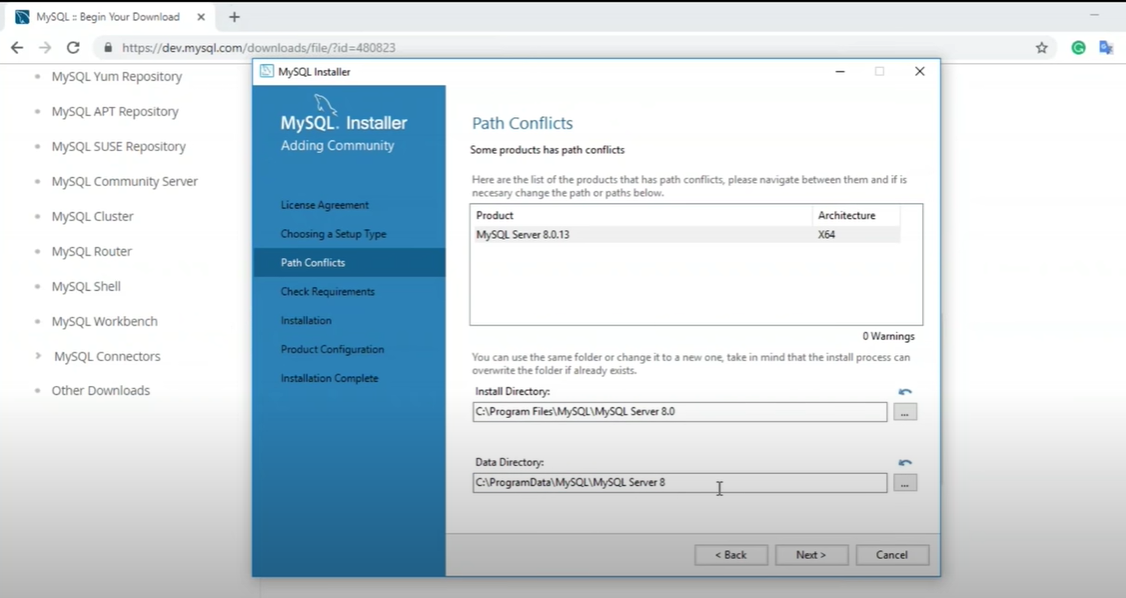
**STEP: - 2**



**STEP: - 3**



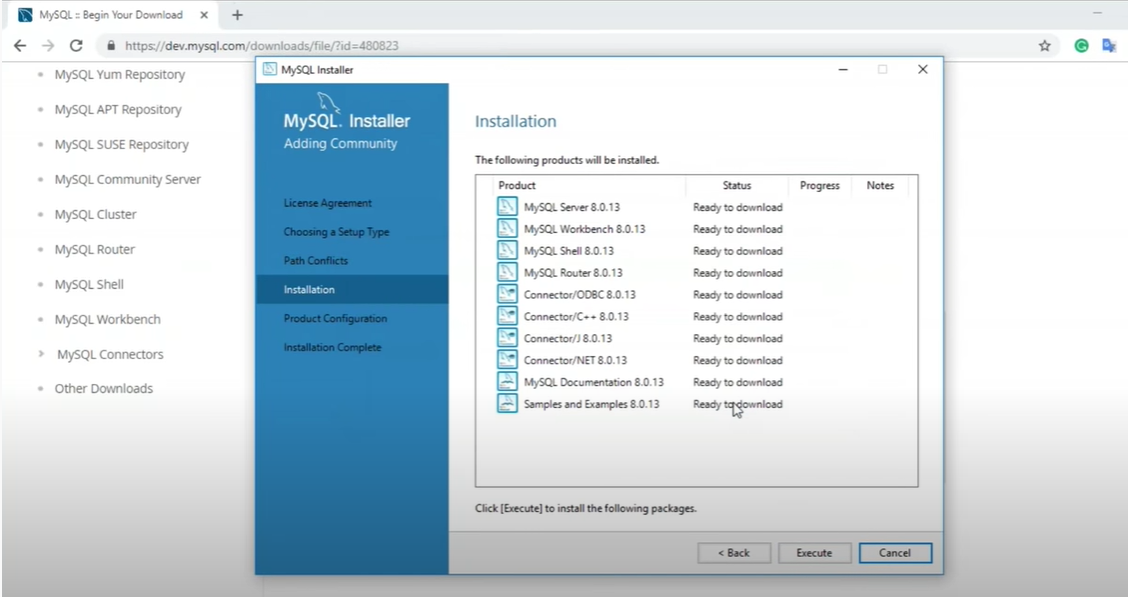
**STEP: - 4**



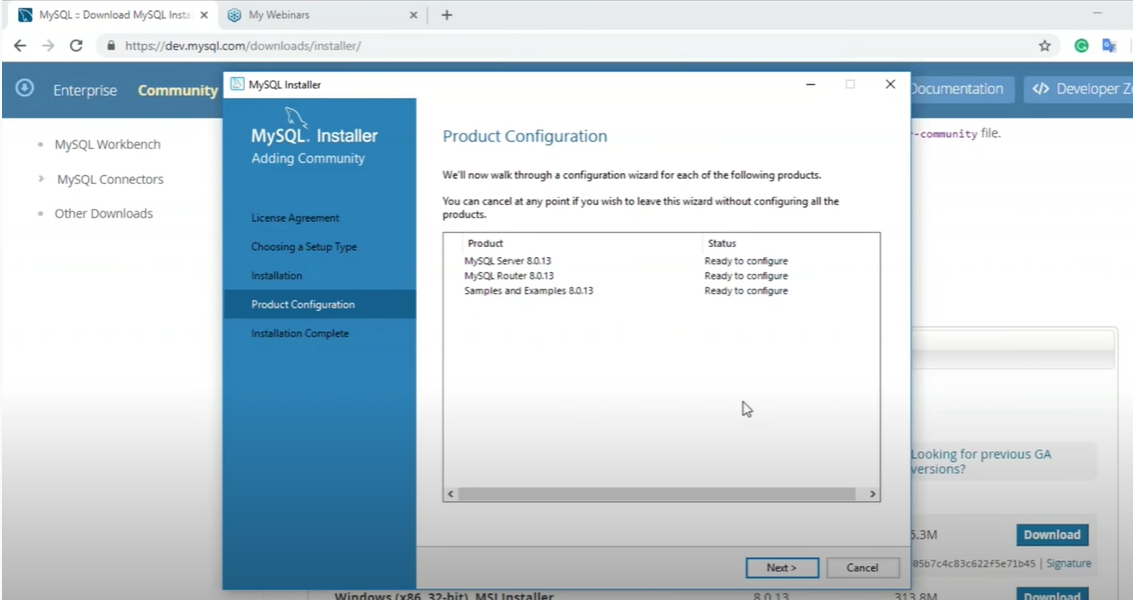
**STEP: - 5**



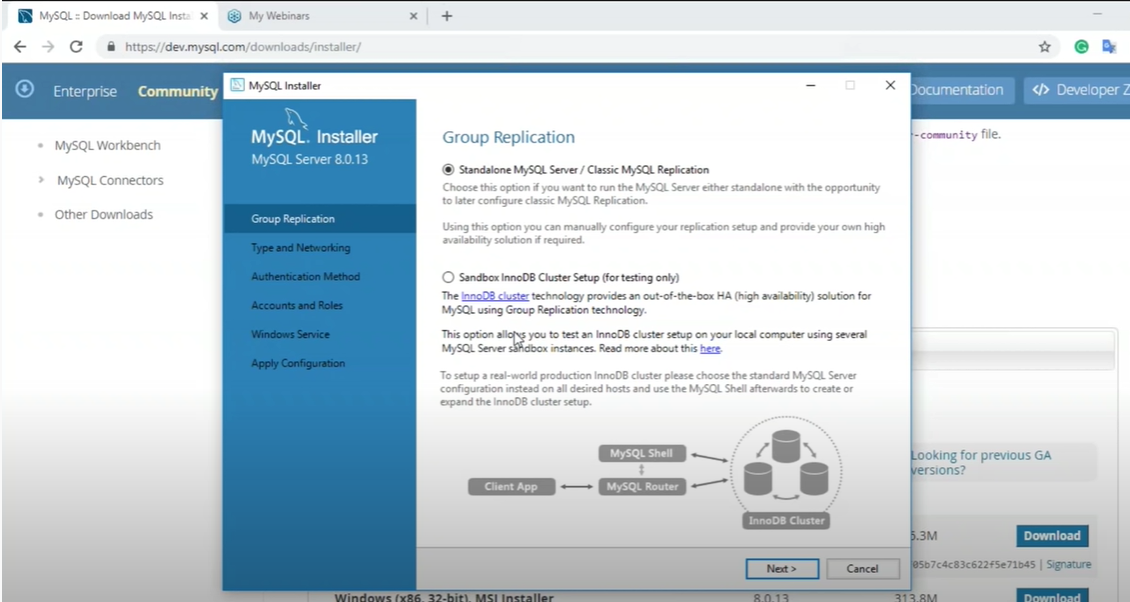
**STEP: - 6**



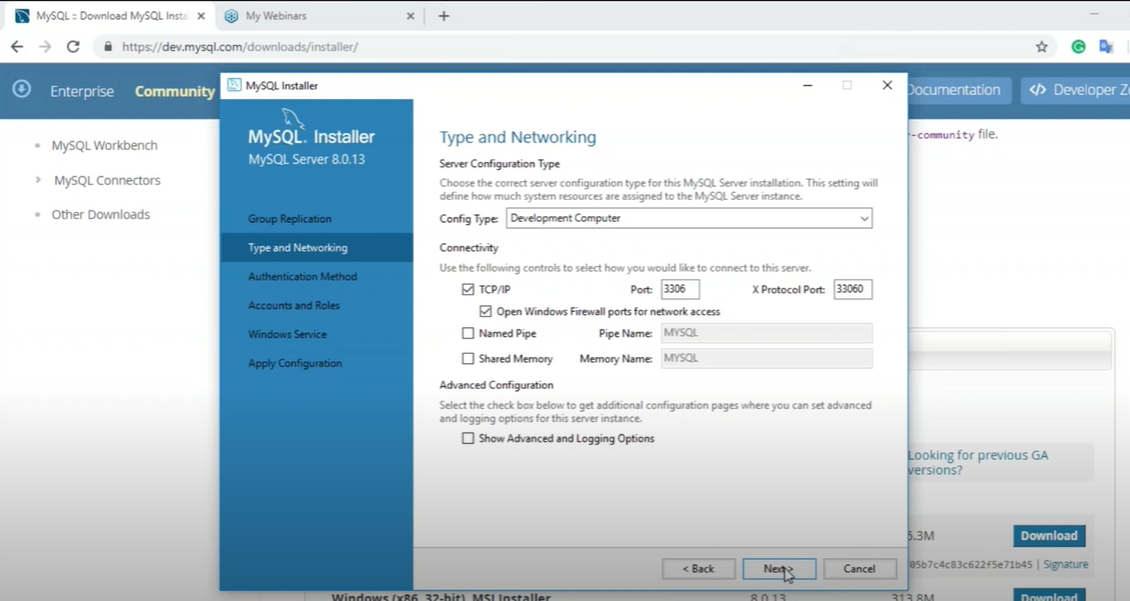
**STEP: - 7**



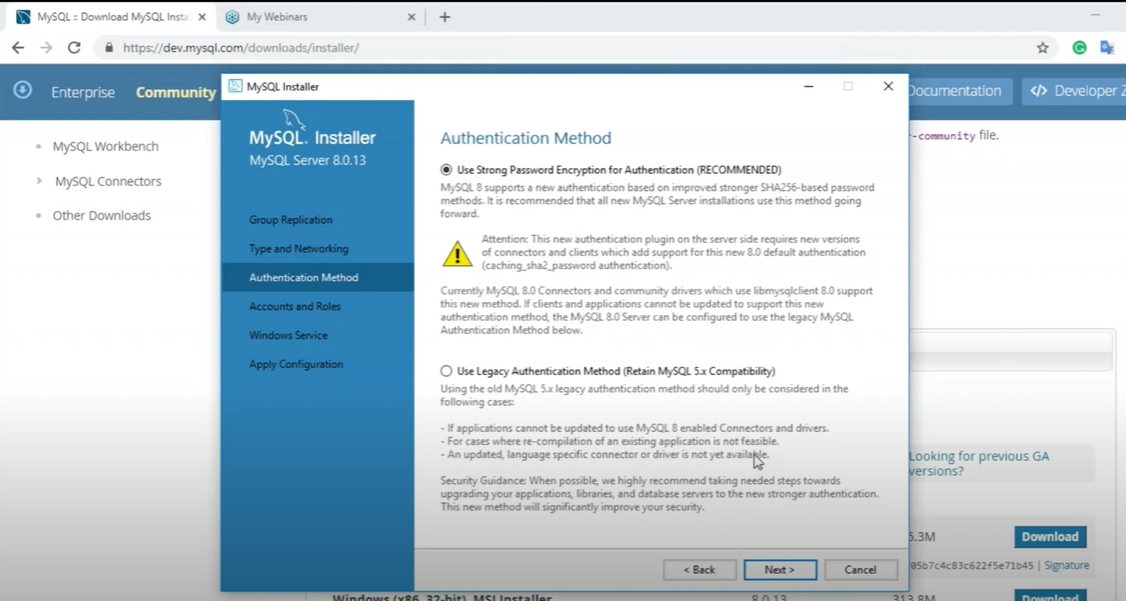
**STEP: - 8**



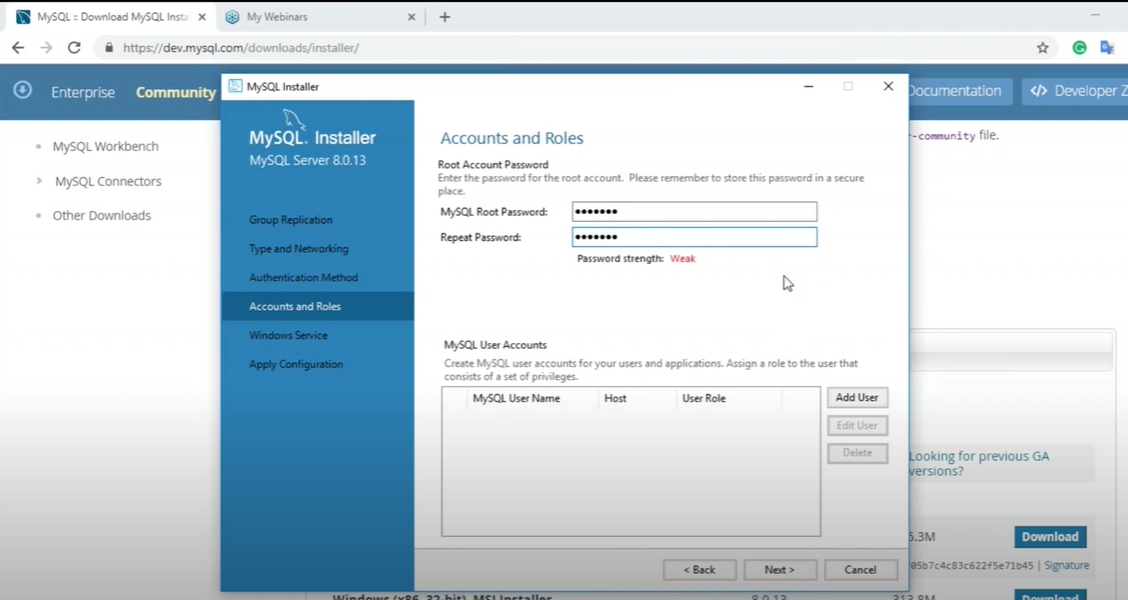
**STEP: - 9**



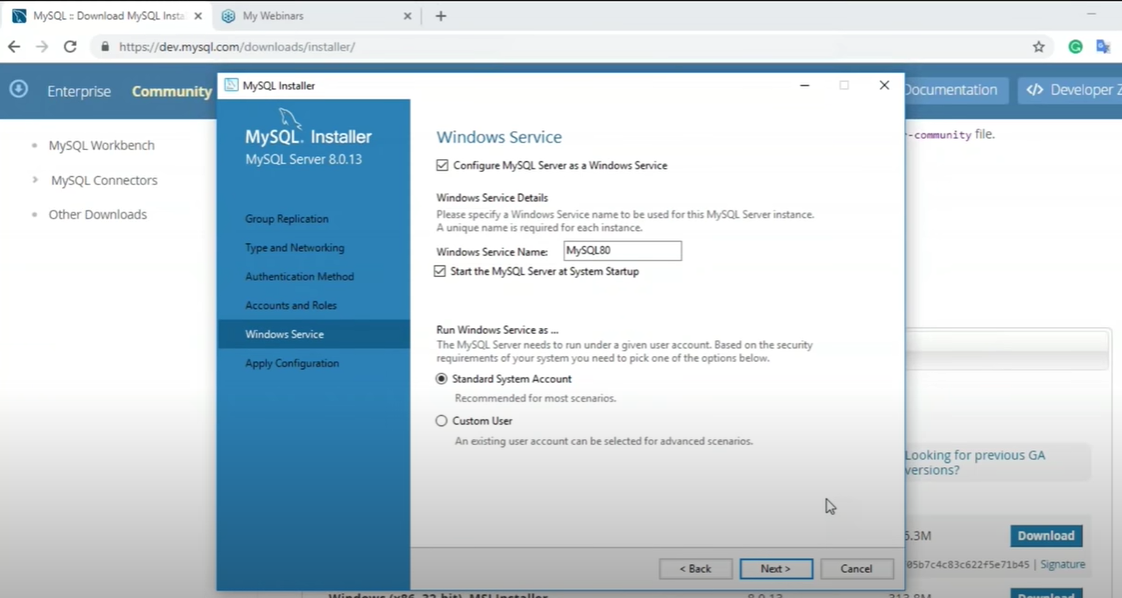
**STEP: - 10**



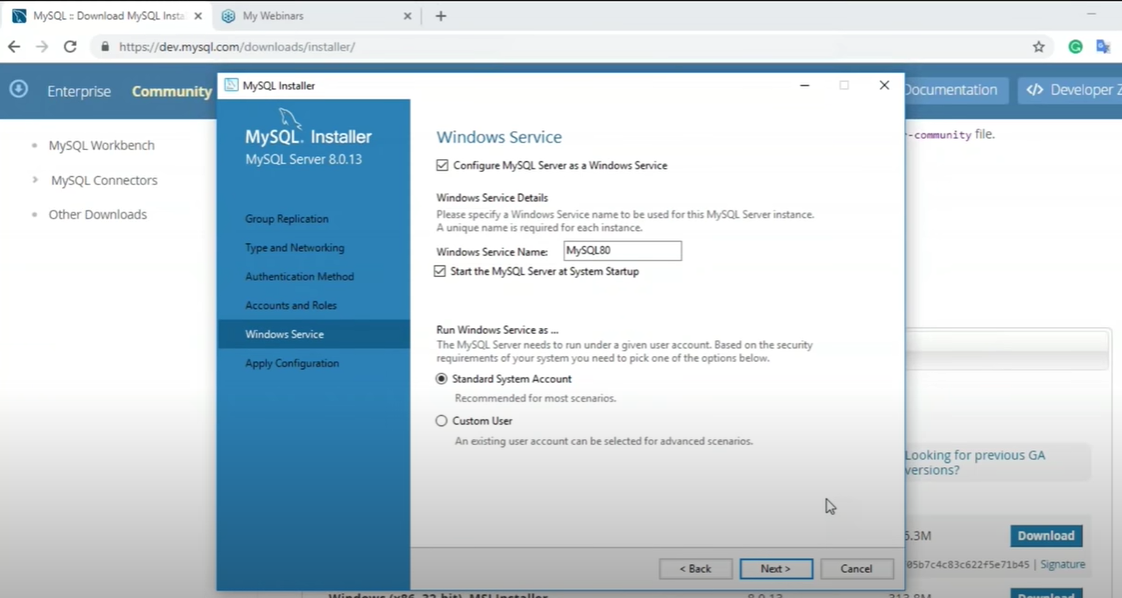
**STEP: - 11**



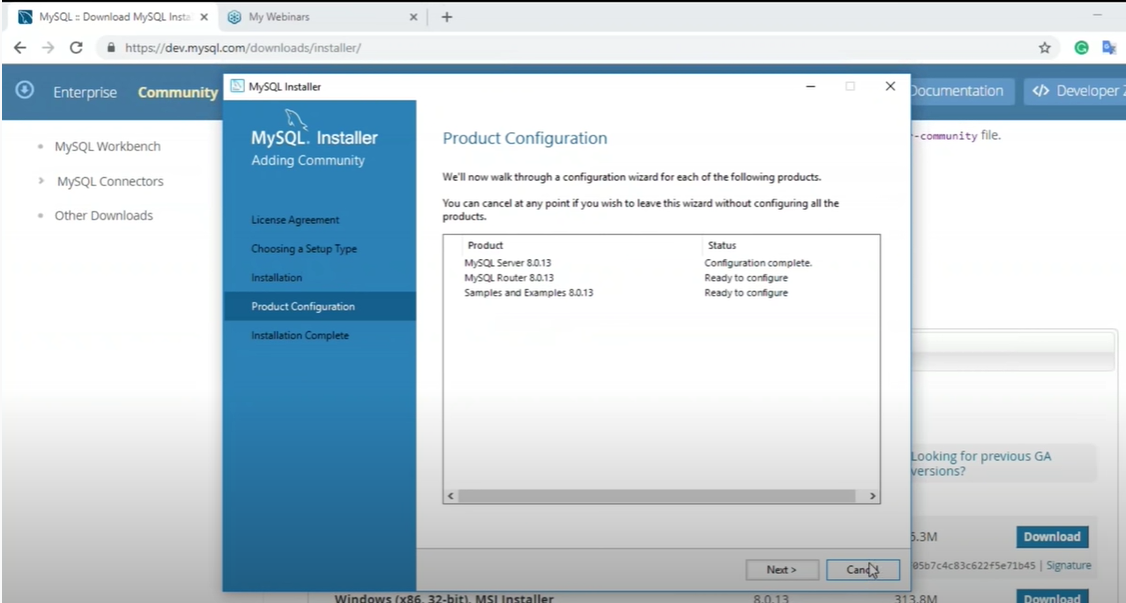
**STEP: - 12**



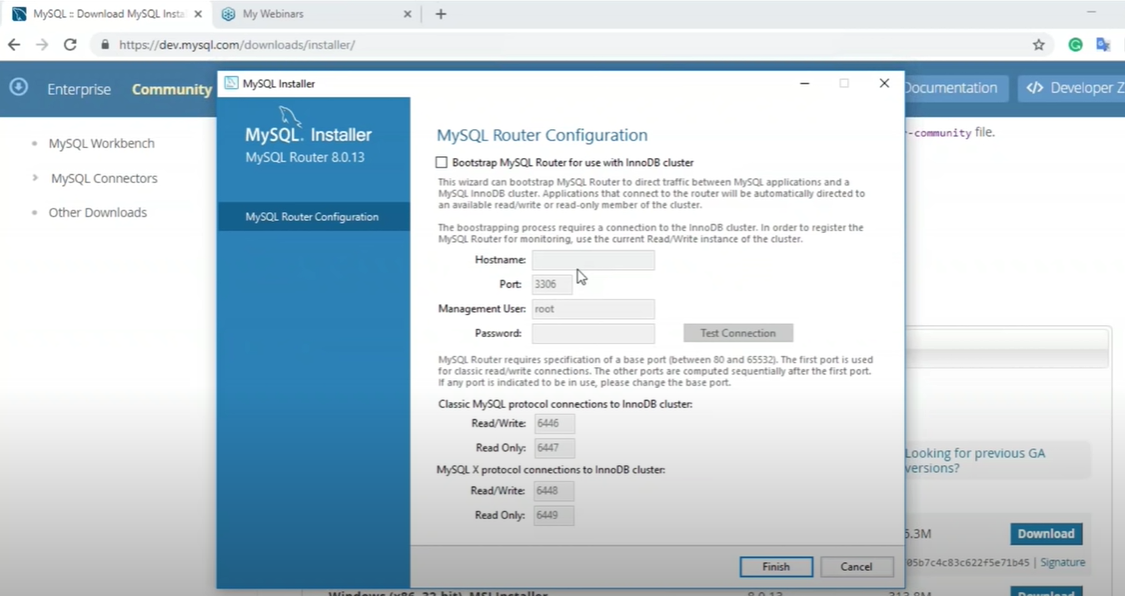
**STEP: - 13**



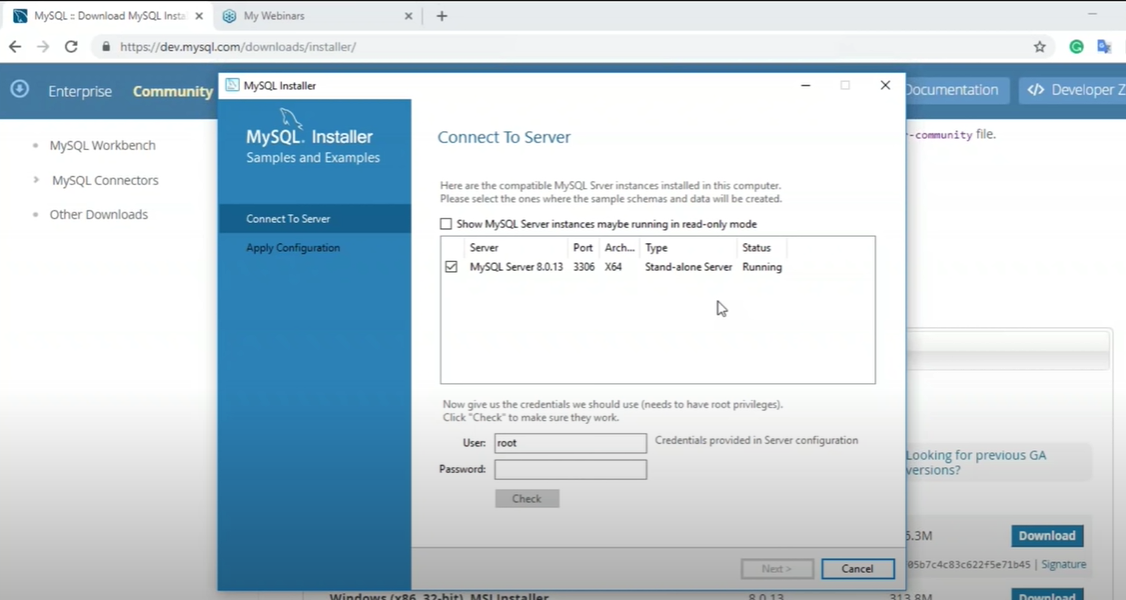
**STEP: - 14**



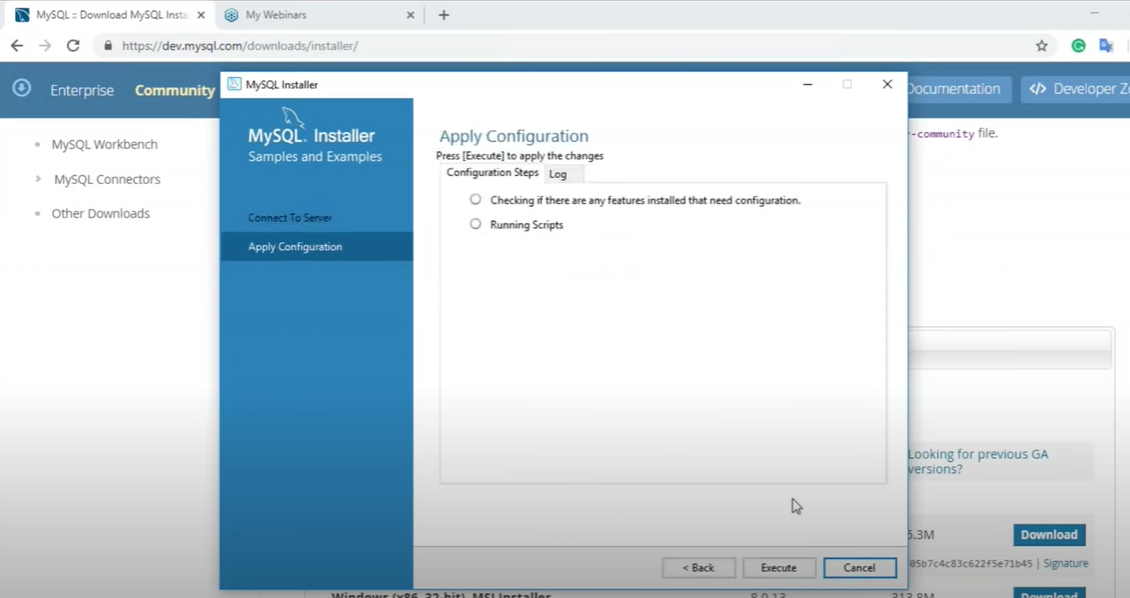
**STEP: - 15**



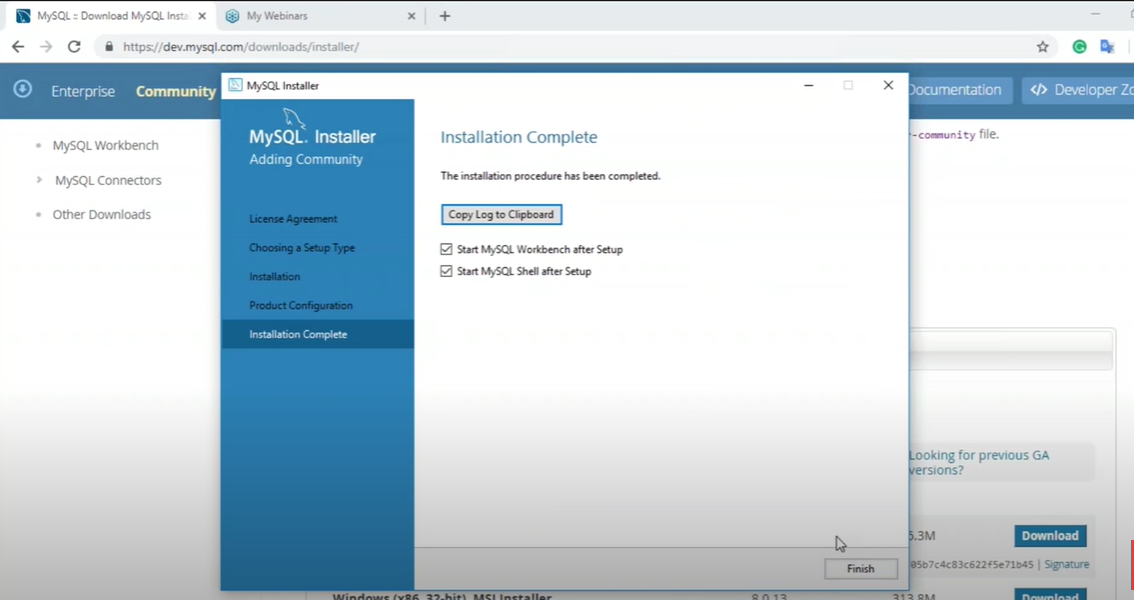
**STEP: - 16**



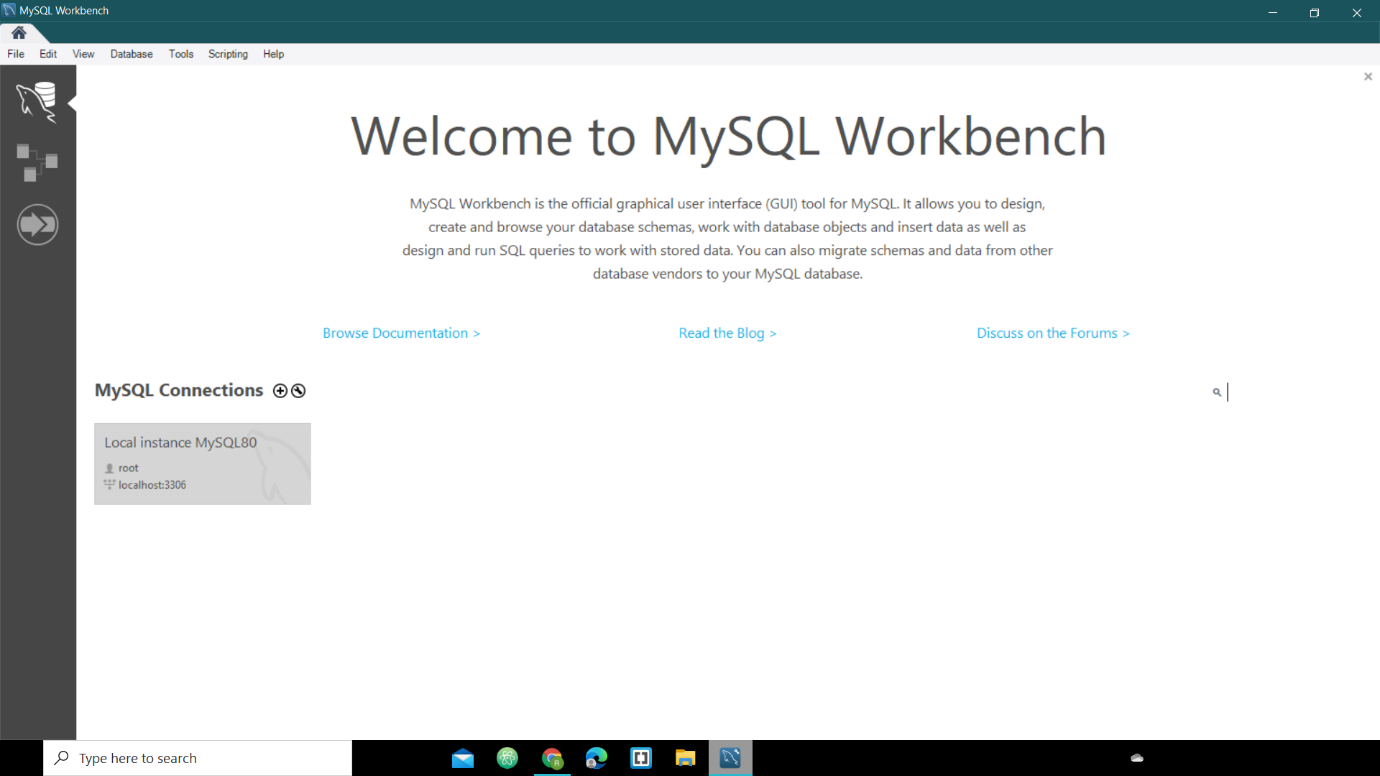
**STEP: - 17**



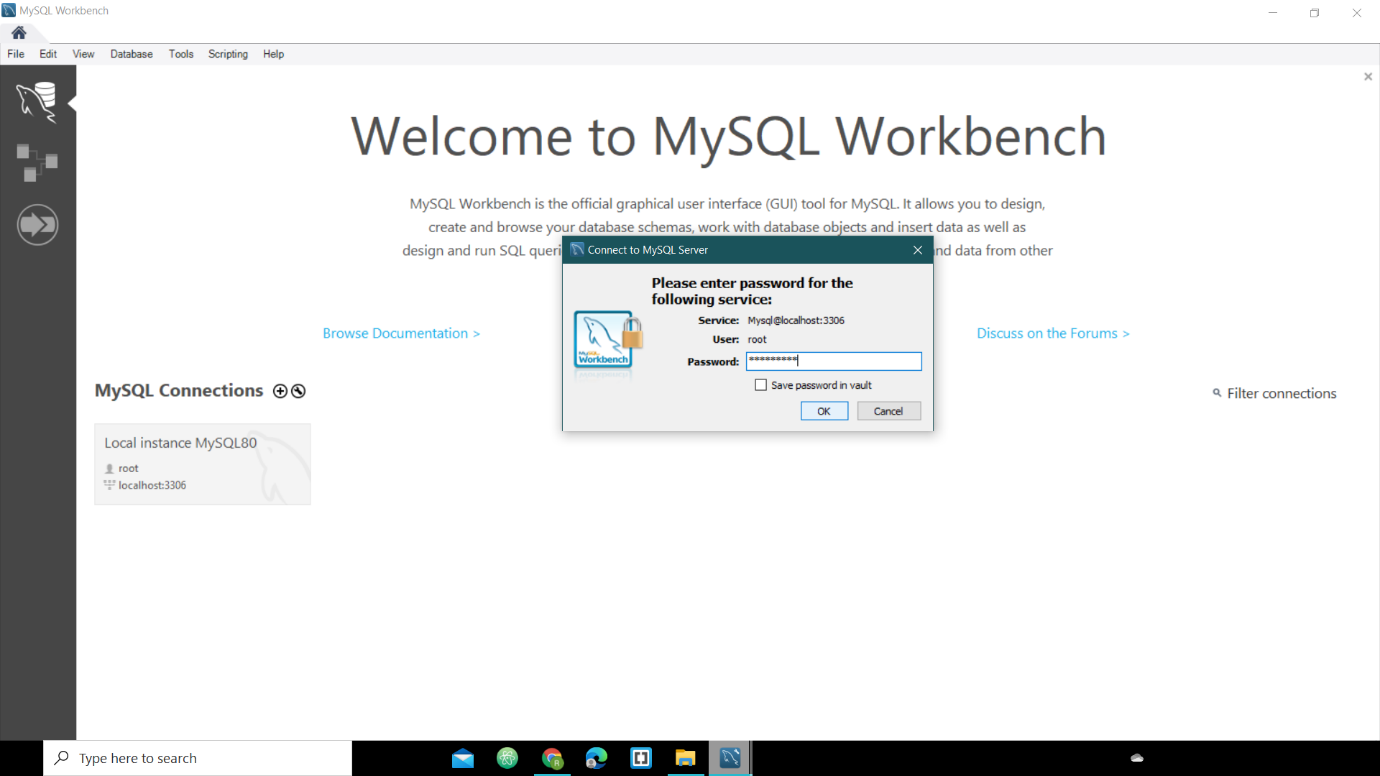
**STEP: - 18**



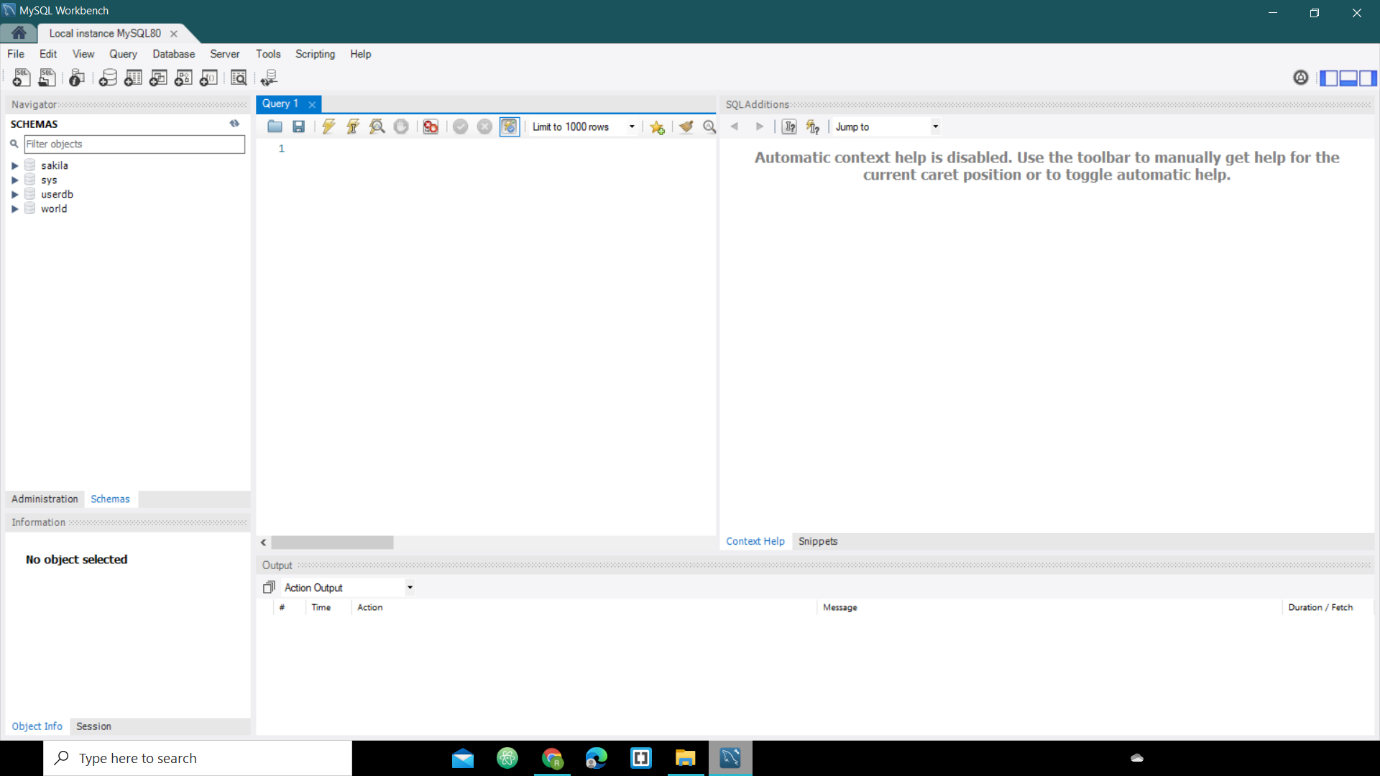
**STEP: - 19**



**STEP: - 20**



**STEP: - 21**



**Assignment – 4**

**19BIT076 (Meet Sheladiya)**

1. **Selection**

Query:

SELECT \* FROM card\_details WHERE amount>1500

Description:

The above query will select rows from card\_details table where amount is greater than 1500

1. **Projection**

Query:

SELECT booking\_id, seat\_number FROM confirmation

Description:

The above query will select columns booking\_id and seat\_number from confirmation table

1. **Cartesian Product**

Query:

SELECT payment\_method.booking\_id, card\_details.amount

FROM

payment\_method, card\_details

Description:

The above query will select columns booking\_id and amount from the tables payment\_method and card\_details respectively and join them.

1. **Union**

Query:

SELECT audience\_id, seat\_number FROM seat\_details

UNION

SELECT audience\_id, ROUND (temperature) FROM audience\_temperature

Description:

The above query will select columns from respective tables and display unique data and display the common data present in both tables only once.

1. **Set Difference**

Query:

SELECT audience\_id, seat\_number FROM seat\_details

MINUS

SELECT audience\_id, ROUND (temperature) FROM audience\_temperature

Description:

The above query will select columns from respective tables and display unique data from columns of seat\_details table which is not present in columns of audience\_temperature table.

1. **Natural Join**

Query:

SELECT \* FROM payment\_method

NATURAL JOIN

card\_details

Description:

The above query will display the common columns from the tables only once.

1. **Composition of any two form (1-6) operators**

Query:

SELECT \* FROM payment\_method

NATURAL JOIN

card\_details

Description:

The above query uses the composition of two operators SELECT and NATURAL JOIN.

1. **Composition of any three form (1-6) operators**

Query:

SELECT \* FROM payment\_method

NATURAL JOIN

card\_details

UNION

confirmation

Description:

The above query uses the composition of three operators SELECT, NATURAL JOIN and UNION. Firstly, the columns of the two tables payment\_method and card\_details will be Natural Joined and then we will Union them with confirmation table.

**19BIT108(Renish Jagani)**

* **Selection:-**

**Query:-**

SELECT \* FROM venue WHERE city\_id =21;

**Description**

The above query will select rows from table venue where city\_id is 21.

* **Projection:-**

**Query:-**

SELECT match\_number, match\_id from fixtures;

**Description**

The above query will select columns match\_number and match\_id from fixtures.

* **Cartesian Product:-**

**Query:-**

SELECT home\_team. home\_team\_ID, away\_team. name, away\_team. away\_team\_id

FROM home\_team, away\_team;

**Description:**  The above query will display all the selected columns from tables home\_team, away\_team

* **Union**

**Query:-**

SELECT match\_number,stadium\_id FROM fixtures

UNION

seat\_number, stadium\_id FROM conformation;

**Description: -** The above query will display only unique data from the selected attributes from respected tables.

* **Set Difference:-**

**Query:-**

SELECT match\_number, stadium\_id FROM fixtures

MINUS

SELECT seat\_number, stadium\_id FROM conformation;

**Description: -** The above query will subtract the common data and display the attributes of fixtures which are not present in conformation.

* **Natural Joint:-**

**Query:-**

SELECT \* FROM

venue NATURAL JOIN fixtures;

**Description:** The columns with the same name of associated tables will appear once only.

* **Composition of any two from (1-6) operators**

**Query:-**

SELECT \* FROM

fixtures NATURAL JOIN conformation

**Description:** The above query uses the composition of two operators SELECT AND NATURAL JOIN.

* **Composition of any three of above from (1-6) operators**

SELECT \* FROM

fixtures NATURAL JOIN conformation

UNION

venue

**Description**:The above query uses the composition of three operators SELECT, NATURAL JOIN and union. Firstly, the attributes of two tables fixtures and conformation will be Natural Joined and all the duplicate values of common attribute will be eliminated and then we will union it with venue table.

**19BIT109(Risht Samariya)**

* **Selection:-**

**Query:**

SELECT \* FROM food name WHERE MRP=Rs 100

**Description-** Selection will select tuples from table food\_and\_beverages where the food name mrp is Rs100

* **Projection:-**

**Query:**

SELECT food\_id, seat\_number from food\_and\_beverages\_order

**Description**- Projection will select the column of food\_id and seat\_number from table food\_and \_beverages\_order

* **Cartesian Product:-**

**Query:**

SELECT jersy. jersy\_id, jersy\_order. booking\_id FROM jersy, jersy\_order

**Description**- It will merge jersy\_id and seat\_number from jersy table to booking\_id from jersy\_order table

* **Union:-**

**Query:**

SELECT seat\_number from food\_and\_beverages

UNION

SELECT seat\_number from jersy

**Description**- Union will merge the seat\_number of table food\_and\_beverages with seat\_number of table jersy which will show only unique data which is not common in both column.

* **Set difference:-**

**Query:**

SELECT booking\_id from food\_and\_beverages\_order

MINUS

SELECT booking\_id from jersy\_order

**Description**- It will subtract the common data from booking\_id of both table food\_and\_beverages\_order and jersy\_order and it will display it.

* **Natural Join:--**

**Query:-**

SELECT\*FROM food\_and\_beverages NATURAL JOIN jersy

**Description**- The columns of seat\_number and MRP in both table will only appear once in the output because it is common in both the table.

* **Composition of any two from (1-6) operators Query:-**

**Query:-**

SELECT booking\_id from food\_and\_beverages

UNION

SELECT booking\_id from jersy

**Description**- there is two operation composition projection and union

* **Composition of any three of above from (1-6) operators Query:-**

**Query:-**

SELECT \* FROM

food\_and\_beverages NATURAL JOIN jersy

UNION

Food\_and\_beverages\_order

**Description**- there is three operation composition projection(select), Natural Join and Union

**19BIT141(Rishabh Patel)**

1. **Select Operation**

Query: - SELECT \* FROM seat\_booking WHERE seat\_number >= 45

Description – In Select operation, we are selecting seat\_number attribute from seat\_booking and having constraint that seat\_number must be greater than greater 45.

1. **Projection**

Query: - SELECT username, password FROM login

Description – In above projection operation we select username and password from login table and output both table.

1. **Cartesian Product**

Query: - SELECT login. email\_id, login. username, seat\_booking. seat\_number

FROM login, seat\_booking

Description – In Cartesian product is selecting email\_id and seat\_number from login and seat\_number from seat\_booking and displaying all three attributes as an output.

1. **Union**

Query: - SELECT email\_id, age FROM login

UNION

SELECT email\_id, seat\_number FROM seat\_booking

Description – In Union Operation it includes email\_id, age from login and email\_id, seat\_number from seat\_booking and it eliminates duplicate tuples.

1. **Set Difference**

Query: - SELECT email\_id, age FROM login

MINUS

SELECT email\_id, seat\_number FROM seat\_booking

Description – The output of set difference (A-B), is removing common values of age and seat\_number and display only unique data from the login table.

1. **Natural Join**

Query: - SELECT \* FROM login NATURAL JOIN seat\_booking

Description – In Natural Join, displaying the attributes only once which are present in both

The tables and uncommon as it is.

1. **Composition of any two from (1-6)operations**

Query: - SELECT login.email\_id, login.username, seat\_booking.audience\_id

FROM login, seat\_booking

UNION

SELECT audience\_id, gate\_number, tier FROM seat\_arrangement

Description: - In this above query, there are composition of two operations **Cartesian product** and **Union** operator firstly selecting name, phone\_number seat\_number attributes from seat\_booking table and section from seat\_location and cross joining two tables and finally talking union with gate\_number which results in displaying all values of respective attributes only once.

1. **Composition of any three from (1-6)operations**

Query: - SELECT email\_id, booking\_id, audience\_id FROM seat\_booking

NATURAL JOIN seat\_location

UNION

SELECT email\_id, phone\_number, username FROM login

MINUS

SELECT audience\_id FROM seat\_arrangement

Description:- In the above query, the Composition of three operators are **Natural Join, Union** and **Set Difference** first we select all attributes from seat\_booking and seat\_location tables and perform natural join so we get all unique columns and common columns only once then taking union with email\_id and phone\_number attribute from login input table so we include both attribute in our output then finally set difference with audience\_id attribute from seat\_arrangement so final output is that we will get all the attributes except of audience\_id attribute because we minus it from previous output table.

**19BIT142(Ujash Thakkar)**

1. **Selection Operation**

**QUERY:**

SELECT tuples FORM match\_details WHERE match\_number =15.

**Description:**

The above query will select rows form table match\_details where match\_number is 15.

1. **Projection**

**QUERY:**

SELECT seat\_number and booking\_id FORM audience\_verification.

**Description:**

In above query will select seat\_number and booking\_id from audience\_verification.

1. **Cartesian Product**

**QUERY:**

SELECT booking\_id, match\_number, audience\_id FORM audience\_verification, tickets\_cancelled.

**Description:**

In above query Cartesian product is selecting booking\_id, audience\_id form audience\_verification and match \_number from tickets\_cancelled.

1. **Union**

**QUERY:**

SELECT stadium\_id, audience\_id, match\_number FORM match\_details.

UNION

SELECT stadium\_id, stadium\_number FORM stadium\_details.

**Description:**

In Union Operation in includes stadium\_id, audience\_id, match\_number FORM match\_details and stadium\_id, stadium\_number FORM stadium\_details and it eliminates supplicate tuples.

1. **Set Difference**

**QUERY:**

SELECT stadium\_id, audience\_id, match\_number FORM match\_details.

MINUS

SELECT stadium\_id, stadium\_number FORM stadium\_details.

**Description:**

The above query will subtract the common data and display the unique data.

1. **Natural Join**

**QUERY:**

SELECT \*FROM tickets\_cancelled NATURAL JOIN audience\_verification.

**Description:**

In above query natural join, print the attrubutes only once which are not present in both tables and uncommon as it is.

1. **Composition of any two from (1-6) operators**

**QUERY:**

SELECT seat\_number

FROM audience\_verification

NATURAL JOIN tickets\_cancelled

**Description:** The above query uses the composition of two operators SELECT AND NATURAL JOIN.

1. **Composition of any three of above from (1-6) operators**

**QUERY:**

SELECT seat\_number

FROM audience\_verification.

NATURAL JOIN tickets\_cancelled

UNION match\_detail

**Description:** The above query uses the composition of 3 operators SELECT, UNION AND NATURAL JOIN.

**19BIT144(Ananya Khandelwal)**

1. **Selection** – SELECT \* FROM tickets\_sold where sold >= 500.

**Description** – Selection will select tuples from table tickets sold where the tickets sold are more than 500.

1. **Projection** – SELECT brand\_name, cost from advertisement.

**Description** – Projection will select the column of brand\_name and cost from table advertisement.

1. **Cartesian product** - SELECT tickets\_sold. stadium\_id, tickets\_sold. sold, tickets\_available. available

FROM tickets\_sold, tickets\_available.

**Description** – Cartesian product will merge stadium\_id, sold and available from tables ticekts\_sold and tickets\_available respectively.

1. **Union** – Select match\_number, sold from tickets sold

UNION

select match\_number, available from tickets available.

**Description** – Union will display only unique data from the selected attributes which is not common in both the tables.

1. **Set Difference** - Select match\_number, sold from tickets sold

MINUS

select match\_number, available from tickets\_available.

**Description** – Set difference will subtract the common data and display the attributes of tickets sold which are not present in tickets available.

1. **Natural Join** – SELECT \* FROM

tickets\_sold

NATURAL JOIN

weather.

**Description** – The columns with same name in both the tables will appear only once in the output.

1. **Composition of Two Operators** – SELECT \* FROM

Weather NATURAL JOIN tickets\_available

**Description** – This query will composite two individual operators i.e., SELECT & NATURAL JOIN.

1. **Composition of Three Operators** – SELECT \* FROM

Weather NATURAL JOIN tickets\_available.

UNION

tickets\_sold.

**Description** – This query will composite three individual operators i.e., SELECT, NATURAL JOIN, UNION.

**Assignment-5.2**

**19BIT076 (Meet Sheladiya)**

1. **payment\_method**

CREATE TABLE `payment\_method` (

`booking\_id` varchar (255),

`match\_id` varchar (255),

`payment\_mode` varchar (255),

`payment\_id` varchar (255) PRIMARY KEY AUTO\_INCREMENT,

`amount` int DEFAULT NULL

);

1. **card\_details**

CREATE TABLE `card\_details` (

`payment\_id` varchar (255) PRIMARY KEY AUTO\_INCREMENT,

`card\_number` varchar (255),

`amount` int DEFAULT NULL,

`expiry\_date` varchar (255)

);

1. **confirmation**

CREATE TABLE `confirmation` (

`booking\_id` varchar (255),

`stadium\_id` varchar (255),

`match\_id` varchar (255),

`seat\_number` int DEFAULT NULL PRIMARY KEY AUTO\_INCREMENT

);

1. **audience\_temperature**

CREATE TABLE `audience\_temperature` (

`audience\_id` varchar (255) PRIMARY KEY AUTO\_INCREMENT,

`temperature` double DEFAULT NULL

)

**19BIT108(Renish Jagani)**

* CREATE TABLE ‘venue(

‘city\_id’ int,

‘name’ varchar (255),

‘stadium\_id’ varchar (255) PRIMARY KEY AUTO\_INCREMENT,

);

* CREATE TABLE ‘fixtures’ (

‘match\_number’ int PRIMARY KEY AUTO\_INCREMENT,

‘match\_id’ varchar (255),

‘stadium\_id’ varchar (255),

‘home\_team\_id’ varchar (255),

‘away\_team\_id’ varchar (255),

‘city\_id’ int,

);

* CREATE TABLE ‘home\_team\_id’ (

‘home\_team\_id’ varchar (255) PRIMARY KEY AUTO\_INCREMENT,

‘name’ varchar (255),

‘home\_stadium’ varchar (255),

);

* CREATE TABLE ‘away\_team\_id’ (

‘away\_team\_id’ varchar (255) PRIMARY KEY AUTO\_INCREMENT,

‘name’ varchar(255),

)

**19BIT109(Rishit Samariya)**

* CREATE TABLE ‘food\_and\_beverages’(

‘seat\_number’ int PRIMARY KEY AUTO\_INCREMENT,

‘food\_id’ varchar (255),

‘food\_name’ varchar (255),

‘MRP’ int,

);

* CREATE TABLE ‘food\_and\_beverages\_order’(

‘seat\_number’ int PRIMARY KEY AUTO\_INCREMENT,

‘food\_id’ varchar (255),

‘booking\_id’ varchar (255),

);

* CREATE TABLE ‘jersy’(

‘seat\_number’ int PRIMARY KEY AUTO\_INCREMENT

‘jersy\_id’ varchar (255),

‘team\_name’ varchar (255),

‘size’ int,

‘MRP’ int,

);

* CREATE TABLE ‘jersy\_order’(

‘seat\_number’ int PRIMARY KEY AUTO\_INCREMENT

‘jersy\_id’ varchar (255),

‘booking\_id’ varchar (255),

);

**19BIT141(Rishabh Patel)**

* **Login:-**

CREATE TABLE `stadium\_seat\_booking\_database`. `login` (

`email\_id` VARCHAR (45) NOT NULL,

`username` VARCHAR (245) NOT NULL,

`password` VARCHAR (245) NOT NULL,

`f\_name` VARCHAR (245) NOT NULL,

`l\_name` VARCHAR (245) NOT NULL,

`gender` VARCHAR (245) NOT NULL,

`age` INT NOT NULL,

`phone\_number` INT (10) NOT NULL,

`security\_question` VARCHAR (245) NOT NULL,

`answer` VARCHAR (245) NOT NULL,

PRIMARY KEY (`email\_id`, `username`, `password`));

* **Seat\_booking:-**

CREATE TABLE `stadium\_seat\_booking\_database`.`seat\_booking` (

`booking\_id` VARCHAR (245) NOT NULL,

`email\_id` VARCHAR (245) NOT NULL,

`name` VARCHAR (245) NOT NULL,

`phone\_number` INT (10) NOT NULL,

`address` VARCHAR (245) NOT NULL,

`postal\_code` INT (10) NOT NULL,

`seat\_number` INT (10) NOT NULL,

`audience\_id` VARCHAR (245) NOT NULL,

`match\_id` VARCHAR (245) NOT NULL,

PRIMARY KEY (`booking\_id`, `phone\_number`, `seat\_number`, `audience\_id`, `match\_id`));

* **Seat arrangement:-**

CREATE TABLE `stadium\_seat\_booking\_database`.`seat\_arrangement` (

`audience\_id` VARCHAR (245) NOT NULL,

`gate\_number` INT (10) NOT NULL,

`tier` INT (10) NOT NULL,

`stadium\_id` VARCHAR (245) NOT NULL,

PRIMARY KEY (`audience\_id`));

* **Seat Location:-**

CREATE TABLE `stadium\_seat\_booking\_database`.`seat\_location` (

`seat\_number` INT (10) NOT NULL,

`section` VARCHAR (245) NOT NULL,

PRIMARY KEY (`seat\_number`));

* **Seat Details:-**

CREATE TABLE `stadium\_seat\_booking\_database`.`seat\_details` (

`booking\_id` VARCHAR (245) NOT NULL,

`audience\_id` VARCHAR (245) NOT NULL,

`match\_id` VARCHAR (245) NOT NULL,

`seat\_number` INT (10) NOT NULL,

`seat\_category` VARCHAR (245) NOT NULL,

`price` DOUBLE (10) NOT NULL,

PRIMARY KEY (`booking\_id`, `audience\_id`, `seat\_number`));

**19BIT142(Ujash Thakkar)**

1. audience\_verification

CREATE TABLE’ audience\_verification` (

‘booking\_id’ varchar (255),

‘audience\_id’ varchar (255), PRIMARY KEY AUTO\_INCREMENT

‘phone\_number’ int (10),

‘seat\_number’ int (10), PRIMARY KEY AUTO\_INCREMENT

‘aadhar\_number’ int (12),

PRIMARY KEY (`booking\_id’, ‘phone\_number’, ‘aadhar\_number’));

1. ticket\_cancelled

CREATE TABLE` ticket\_cancelled ‘(

‘stadium\_id` varchar (255),

‘match\_id` varchar (255),

‘match\_number’ int DEFAULT NULL, PRIMARY KEY AUTO\_INCREMENT

‘audience\_id’ varchar (255), PRIMARY KEY AUTO\_INCREMENT

1. stadium\_details

CREATE TABLE ‘stadium\_details’ (

‘stadium\_id’ varchar (255), PRIMARY KEY AUTO\_INCREMENT

‘stadium\_name’ varchar (255),

‘address’ varchar (255),

‘capacity’ int,

1. match\_details

CREATE TABLE ‘match\_details’ (

‘stadium\_id’ varchar (255),

‘audience\_id’ varchar (255), PRIMARY KEY AUTO\_INCREMENT

‘match\_number’ INT DEFAULT NULL, PRIMARY KEY AUTO\_INCREMENT

‘match\_id` varchar (255),

‘date\_of\_match’ varchar (255),

‘match\_timing’ varchar (255),

**19BIT144(Ananya Khandelwal)**

1. CREATE TABLE `tickets\_sold` (

`Stadium\_id` varchar (255),

`Match\_id` varchar (255),

`Match\_number` int DEFAULT NULL, PRIMARY KEY AUTO\_INCREMENT

`Sold` int DEFAULT NULL

);

1. CREATE TABLE `advertisement` (

`Audience\_id` varchar (255), PRIMARY KEY AUTO\_INCREMENT

`Brand\_name` varchar (255),

`Cost (In Rs) ` double DEFAULT NULL

);

1. TABLE `weather` (

`Match\_Number` int DEFAULT NULL, PRIMARY KEY AUTO\_INCREMENT

`Temperature (Celsius) ` double DEFAULT NULL,

`Humidity (%) ` int DEFAULT NULL

;

1. CREATE TABLE `tickets\_available` (

`Stadium\_id` varchar (255),

`Match\_id` varchar (255),

`Match\_number` int DEFAULT NULL, PRIMARY KEY AUTO\_INCREMENT

`Available` int DEFAULT NULL

)

**Assignment – 5.3**

**19BIT076 (Meet Sheladiya)**

1. **Aggregate functions**

**Max**

Query:

SELECT MAX (temperature) FROM audience\_temperature

1. **Numeric**

Query:

SELECT ROUND (temperature) FROM audience\_temperature

1. **Date**

Query:

SELECT \* FROM card\_details WHERE expiry\_date=’2023-05-27’

1. **String function**

Query:

SELECT LENGTH (payment\_mode) FROM payment\_method

**19BIT108 (Renish Jagani)**

* **Aggregate function:-**

SELECT COUNT (city\_id) FROM venue;

* **Built-in Numeric Function:-**

SELECT MAX (humidity) FROM weather;

* **String Functions:-**

SELECT name, ASCII (name)

FROM home\_team;

* **Date Function:-**

SELECT YEAR ('2017/08/25');

**19BIT109 (Rishit Samariya)**

* **Aggregate function-**

**Query:-**

SELECT MAX (MRP) FROM food\_and\_beverages;

* **Built-in Numeric Function-**

**Query:-**

SELECT ABS (Size) FROM Jersy;

* **String Function-**

**Query:-**

Select food\_name, UPPER (food\_name) from food\_and\_beverages

* **Date Function-**

**Query:-**

SELECT \* FROM card\_details WHERE expiry\_date=’2025-01-02’

**19BIT141 (Rishabh Patel)**

1. **Aggregate function:-**

SELECT SUM (age) FROM login;

1. **Built-in Numeric Function:-**

SELECT ROUND (price) FROM seat\_details;

1. **String Function:-**

SELECT CONCAT (f\_name, l\_name) FROM login;

1. **Date Function:-**

SELECT \* FROM card\_details WHERE expiry\_date = ‘2023-24-11’

**19BIT142(Ujash Thakkar)**

1. **Aggregate function**

**Query:**

SELECT COUNT (audience\_id) FROM ticket\_cancelled

1. **Numeric function**

**Query:**

SELECT ABS (seat\_number) FROM audience\_verification

1. **Date function**

**Query:**

SELECT date\_of\_match FROM match\_details WHERE

date\_of\_match=’2021-04-03’;

1. **String function**

**Query:**

SELECT stadium\_name, LOWER (stadium\_name) FROM stadium\_details

**19BIT144 (Ananya Khandelwal)**

1. **Aggregate Function:-**

SELECT avg (cost) FROM advertisement.

1. **Built in Numeric function:-**

SELECT ROUND (temperature) FROM weather where match number = 2.

1. **Date:-**

SELECT YEAR (‘2019/0510’)

1. **String:-**

SELECT length (brand\_name) FROM advertisement where audience id = \_\_\_\_\_\_\_\_\_.

**Assignment-5.4**

**19BIT076 (Meet Sheladiya)**

1. **Set Operations**
2. **Union**

Query:

SELECT audience\_id, seat\_number FROM seat\_details

UNION

SELECT audience\_id, ROUND (temperature) FROM audience\_temperature

1. **Union All**

Query:

SELECT audience\_id, seat\_number FROM seat\_details

UNION ALL

SELECT audience\_id, ROUND (temperature) FROM audience\_temperature

1. **Intersect**

Query:

SELECT audience\_id, seat\_number FROM seat\_details

INTERSECT

SELECT audience\_id, ROUND (temperature) FROM audience\_temperature

1. **Minus**

Query:

SELECT audience\_id, seat\_number FROM seat\_details

MINUS

SELECT audience\_id, ROUND (temperature) FROM audience\_temperature

1. **Sub-queries**

Query:

SELECT \* FROM audience\_temperature WHERE temperature>37

1. **Correlated sub-queries**

Query:

SELECT booking\_id, match\_id, amount

FROM payment\_method

WHERE payment\_id = (SELECT payment\_id FROM card\_details WHERE amount>4900)

**19BIT108 (Renish Jagani)**

* **Set operations:-**

SELECT match\_number, stadium\_id FROM fixtures

UNION

seat\_number, stadium\_id FROM conformation;

* **Sub queries:-**

SELECT \*

FROM fixtures

WHERE match\_id IN (SELECT match\_id

FROM fixtures

WHERE city\_id > 20);

* **Co-related Sub queries:-**

SELECT match\_id from fixtures where stadium\_id= (SELECT STADIUM\_ID FROM VENUE WHERE city\_id>6)

**19BIT109 (Rishit Samariya)**

1. **Set Operations:-**

* **Union-**

**Query:-**

SELECT seat\_number from food\_and\_beverages

UNION

SELECT seat\_number from jersy

* **Union All-**

**Query:-**

SELECT seat\_number from food\_and\_beverages

UNION All

SELECT seat\_number from jersy

* **Intersect-**

**Query:-**

SELECT seat\_number from food\_and\_beverages

INTERSECT

SELECT seat\_number from jersy

* **Minus-**

**Query:-**

SELECT booking\_id from food\_and\_beverages\_order

MINUS

SELECT booking\_id from jersy\_order

1. **Sub-queries:-**

**Query:-**

SELECT \* FROM food name WHERE MRP=Rs 100

1. **Correlated sub-queries:-**

**Query:-**

SELECT food\_name, seat\_number,

FROM food\_and\_beverages

WHERE food\_id = (SELECT food\_id FROM food\_and\_beverages\_order WHERE seat\_number = 10)

**19BIT141 (Rishabh Patel)**

1. **Set Operations:-**

* **INTERSECT:-**

**Query:-**

SELECT email\_id, username, password FROM login

INTERSECT

SELECT email\_id, name, phone\_number FROM seat\_booking;

* **UNION:-**

**Query:-**

SELECT email\_id, age FROM login

UNION

SELECT email\_id, seat\_number FROM seat\_booking;

* **UNION ALL:-**

**Query:-**

SELECT \* FROM seat\_booking

UNION ALL

SELECT \* FROM seat\_arrangement;

* **MINUS:-**

**Query:-**

SELECT audience\_id, name, seat\_number FROM seat\_booking

MINUS

SELECT audience\_id, gate\_number, tier;

1. **Sub-Queries:-**

* **UPDATE:-**

**Query:-**

UPDATE seat\_details

SET price = price \* 0.5

WHERE 4 >= (SELECT COUNT (match\_id) FROM seat\_booking

WHERE seat\_booking.seat\_number = seat\_details.seat\_number)

1. **Correlated Sub-Queries:-**

**Query:-**

SELECT name, email\_id, seat\_number

FROM seat\_booking

WHERE booking\_id = (SELECT booking\_id FROM seat\_details

WHERE audience\_id = ‘A-011’);

**19BIT142(Ujash Thakkar)**

1. **Set Operations**

**Union**

**QUERY:**

SELECT stadium\_id, audience\_id, match\_number FORM match\_details.

UNION

SELECT stadium\_id, stadium\_number FORM stadium\_details.

1. **Sub-queries**

**QUERY:** SELECT \* FROM audience\_temperature WHERE temperature>37

1. **Correlated sub-queries**

**QUERY:**

SELECT stadium\_id, stadium\_name, FROM stadium\_details

WHERE match\_id = (SELECT match\_id FROM match\_details

WHERE match\_number>23)

**19BIT144 (Ananya Khandelwal)**

1. **Set Operations:-**

SELECT \* FROM weather

INTERSECT

SELECT \* FROM tickets\_available.

1. **Sub Queries:-**

SELECT \* FROM advertisement WHERE audience\_id IN (SELECT audience\_id FROM advertisement WHERE cost > 100000)

1. **Correlated Sub Queries:-**

SELECT stadium\_id, match\_id FROM tickets\_available

WHERE match\_number= (SELECT match\_number FROM tickets\_sold WHERE sold>4900)

**Assignment-5.5**

**19BIT076 (Meet Sheladiya)**

1. **Group By;-**

Query:-

SELECT payment\_mode, SUM(amount) FROM payment\_method

GROUP BY payment\_mode

1. **Having;-**

Query:-

SELECT stadium\_id, COUNT(booking\_id) FROM confirmation

GROUP BY stadium\_id

HAVING COUNT(booking\_id)>1

1. **Order By;-**

Query:-

SELECT \* FROM card\_details

ORDER BY amount

**19BIT108 (Renish Jagani)**

* **Group By:-**

SELECT COUNT (stadium\_id), name  
FROM venue  
GROUP BY name;

* **Having:-**

SELECT COUNT (stadium\_id), name  
FROM venue  
GROUP BY name

HAVING COUNT (stadium\_id)> 1

(RETURNS ONLY CITY THAT HAS MORE THAN ONE STADIUM)

* **Order By:-**

SELECT \* FROM fixtures ORDER BY city\_id;

**19BIT109 (Rishit Samariya)**

* **Group By-**

**Query:-**

SELECT food\_name, AVG (MRP) FROM food\_and\_beverages

GROUP BY food\_name;

* **ORDER BY-**

**Query:-**

SELECT \* FROM food\_and\_beverages ORDER BY MRP DESC;

* **HAVING-**

**Query:-**

SELECT COUNT (MRP), food\_name FROM food\_and\_beverages

GROUP BY food\_name

HAVING COUNT (MRP)>=50;

**19BIT141 (Rishabh Patel)**

1. **Group By:-**

**Query:-**

SELECT security\_question, COUNT (email\_id) FROM login

GROUP BY security\_question;

1. **Having:-**

**Query:-**

SELECT COUNT (email\_id), age FROM login

GROUP BY age

HAVING COUNT (email\_id) >= 8;

1. **ORDER BY:-**

**Query:-**

SELECT COUNT (name), address FROM seat\_booking

GROUP BY address

ORDER BY COUNT (name) DESC;

**19BIT142(Ujash Thakkar)**

1. **Group By**

Query:

SELECT stadium\_name, stadium\_id

FROM stadium\_details

GROUP BY stadium\_name;

1. **Having**

Query:

SELECT stadium\_name, stadium\_id

FROM stadium\_details

GROUP BY stadium\_name

HAVING COUNT (\*) < 8;

1. **Order By**

Query:

SELECT \*

FROM match\_details

ORDER BY date\_of\_match;

**19BIT144 (Ananya Khandelwal)**

1. **Group by:-**

SELECT humidity, count (match\_number) FROM weather

GROUP BY humidity.

1. **Having:-**

SELECT stadium\_id, match\_id, match\_number FROM tickets\_available

GROUP BY available

HAVING available>500.

1. **Order by:-**

SELECT \* FROM weather

ORDER by temperature

**Assignment – 5.6**

**19BIT076 (Meet Sheladiya)**

1. **Join**
2. **Inner Join**

Query:-

SELECT card\_details.amount, payment\_method.booking\_id, payment\_method.match\_id FROM payment\_method

INNER JOIN card\_details

ON payment\_method. payment\_id=card\_details.payment\_id

1. **Left Join**

Query:-

SELECT card\_details.amount, payment\_method.booking\_id, payment\_method.match\_id FROM payment\_method

LEFT JOIN card\_details

ON payment\_method.payment\_id=card\_details.payment\_id

1. **Right Join**

Query:-

SELECT card\_details.amount, payment\_method.booking\_id, payment\_method.match\_id FROM payment\_method

RIGHT JOIN card\_details

ON payment\_method.payment\_id=card\_details.payment\_id

1. **Full Join**

Query:-

SELECT card\_details.amount, payment\_method.booking\_id, payment\_method.match\_id FROM payment\_method

FULL JOIN card\_details

ON payment\_method.payment\_id=card\_details.payment\_id

1. **Exists**

Query:-

SELECT booking\_id, match\_id

FROM payment\_method

WHERE EXISTS (SELECT \* FROM card\_details WHERE payment\_method.payment\_id=card\_details.payment\_id AND amount>2500)

1. **Any**

Query:-

SELECT booking\_id

FROM payment\_method

WHERE payment\_id = ANY (SELECT payment\_id FROM card\_details WHERE amount<2000)

1. **All**

Query:-

SELECT booking\_id

FROM payment\_method

WHERE payment\_id = ALL (SELECT payment\_id FROM card\_details WHERE amount>1000)

**19BIT108 (Renish Jagani)**

* **Join operations inner join:-**

SELECT fixtures.match\_number, fixtures.match\_id,home\_team.name

from fixtures

INNER JOIN home\_team

ON home\_team.home\_team\_id = fixtures.home\_team\_id

* **Left join:-**

SELECT fixtures. match number, fixtures.match\_id,home\_team.name

from fixtures

LEFT JOIN home\_team

ON home\_team.home\_team\_id = fixtures.home\_team\_id

* **Right join:-**

SELECT fixtures.match\_number, fixtures.match\_id,home\_team.name

from fixtures

RIGHT JOIN home\_team

ON home\_team.home\_team\_id = fixtures.home\_team\_id

* **Full join:-**

SELECT fixtures.match\_number, fixtures.match\_id,home\_team.name

from fixtures

FULL JOIN home\_team

ON home\_team.home\_team\_id = fixtures.home\_team\_id

* **Exist:-**

SELECT match\_id, match\_number from fixtures

WHERE EXISITS (SELECT \* FROM away\_team WHERE away\_team.away\_team\_id = fixtures.home\_team\_id)

* **All:-**

SELECT match\_id from fixtures where stadium\_id= All (SELECT STADIUM\_ID FROM VENUE WHERE CITY\_ID>6)

* **Any:-**

SELECT match\_id from fixtures where stadium\_id= Any (SELECT STADIUM\_ID FROM VENUE WHERE CITY\_ID>6)

**19BIT109 (Rishit Samariya)**

* **INNER JOIN-**

**Query:-**

SELECT Food\_and\_beverages\_order.booking\_id, Food\_and\_beverages.MRP, Food\_and\_beverages\_order.booking\_id

FROM food\_and\_bevearges

INNER JOIN Food\_and\_beverages\_order.

ON food\_and\_beverages.seat\_number = food\_and\_beverages\_order.seat\_number;

* **Left Join-**

**Query:-**

SELECT jersy\_order.booking\_id, jersy.team\_name

FROM jersy\_order

LEFT JOIN jersy

ON jersy.seat\_number =jersy\_order.seat\_number;

* **Right Join-**

**Query:-**

SELECT Food\_and\_beverages\_order.booking\_id, Food\_and\_beverages.booking\_id

FROM food\_and\_bevearges

RIGHT JOIN Food\_and\_beverages\_order.

ON food\_and\_beverages.seat\_number = food\_and\_beverages\_order.seat\_number;

* **FULL JOIN-**

**Query:-**

SELECT Food\_and\_beverages\_order.booking\_id, Food\_and\_beverages.booking\_id

FROM food\_and\_bevearges

FULL JOIN Food\_and\_beverages\_order.

ON food\_and\_beverages.seat\_number = food\_and\_beverages\_order.seat\_number;

* **EXISTS-**

**Query:-**

SELECT food\_id from food\_and\_beverages\_order

WHERE EXISTS (SELECT \* FROM jersy\_order

WHERE food\_and\_beverages\_order.seat\_number = jersy\_order.seat\_number)

* **All-**

**Query:-**

SELECT food\_name

FROM food\_and\_beverages

WHERE food\_id = ALL (SELECT food\_id FROM food\_and\_beverages\_order WHERE seat\_number = 10)

* **ANY-**

**Query:-**

SELECT food\_name

FROM food\_and\_beverages

WHERE food\_id = ANY (SELECT food\_id FROM food\_and\_beverages\_order WHERE seat\_number = 10)

**19BIT141 (Rishabh Patel)**

* **Join Operations:-**

1. **INNER JOIN:-**

**Query:-**

SELECT seat\_arrangement. gate\_number, seat\_details. match\_id, seat\_details. seat\_number FROM seat\_details

INNER JOIN seat\_arrangement

ON seat\_details. audience\_id = seat\_arrangement. audience\_id;

1. **LEFT JOIN:-**

**Query:-**

SELECT seat\_arrangement. gate\_number, seat\_details. match\_id, seat\_details. seat\_number FROM seat\_details

LEFT JOIN seat\_arrangement

ON seat\_details. audience\_id = seat\_arrangement. audience\_id;

1. **RIGHT JOIN:-**

**Query:-**

SELECT seat\_arrangement. gate\_number, seat\_details. match\_id, seat\_details. seat\_number FROM seat\_details

RIGHT JOIN seat\_arrangement

ON seat\_details. audience\_id = seat\_arrangement. audience\_id;

1. **FULL JOIN:-**

**Query:-**

SELECT seat\_arrangement. gate\_number, seat\_details. match\_id, seat\_details. seat\_number FROM seat\_details

FULL JOIN seat\_arrangement

ON seat\_details. audience\_id = seat\_arrangement. audience\_id;

* **Exists:-**

**Query:-**

SELECT f\_name, l\_name FROM login

WHERE EXISTS (SELECT \* FROM seat\_booking

WHERE login. email\_id = seat\_booking. Email\_id AND age>=25)

* **Any:-**

**Query:-**

SELECT name

FROM seat\_booking

WHERE audience\_id = ANY (SELECT audience\_id FROM seat\_arrangement

WHERE gate\_number = 2 OR gate\_number = 4)

* **ALL:-**

**Query:-**

SELECT name

FROM seat\_booking

WHERE audience\_id = ALL (SELECT audience\_id FROM seat\_arrangement

WHERE gate\_number = 2 OR gate\_number = 4);

**19BIT142(Ujash Thakkar)**

1. **Inner Join**

**Query:**

**SELECT** audience\_verification. phone number, match\_details. match\_number, match\_details. match\_timing FROM match\_details

**INNER JOIN** audience\_verification

**ON** match\_details. audience\_id = audience\_verification. audience\_id;

1. **Left Join**

**Query:**

**SELECT** audience\_verification. phone number, match\_details. match\_number, match\_details. match\_timing FROM match\_details

**LEFT JOIN** audience\_verification

**ON** match\_details. audience\_id = audience\_verification. audience\_id;

1. **Right join**

**Query:**

**SELECT** audience\_verification. phone number, match\_details. match\_number, match\_details. match\_timing FROM match\_details

**RIGHT JOIN** audience\_verification

**ON** match\_details. audience\_id = audience\_verification. audience\_id;

1. **Full join**

**Query:**

**SELECT** audience\_verification. phone number, match\_details. match\_number, match\_details. match\_timing FROM match\_details

**FULL JOIN** audience\_verification

**ON** match\_details. audience\_id = audience\_verification. audience\_id;

1. **Exists**

**Query:**

**SELECT** booking\_id, seat\_number

FROM audience\_verification

WHERE **EXISTS** (SELECT \* FROM match\_details WHERE

audience\_verification.audience\_id=match\_details.audience\_id)

1. **Any**

**Query:**

**SELECT** booking\_id

FROM audience\_verification

WHERE audience\_id = (SELECT audience\_id FROM ticket\_cancelled WHERE seat\_number is=564)

1. **All**

**Query:**

**SELECT** booking\_id

FROM audience\_verification

WHERE audience\_id = (SELECT audience\_id FROM ticket\_cancelled WHERE seat\_number is=465)

**19BIT144 (Ananya Khandelwal)**

1. **INNER JOIN**

Query -

SELECT weather.temperature, tickets\_available.stadium\_id, tickets\_available.match\_id FROM tickets\_available

INNER JOIN weather

ON tickets\_available.match\_number = weather.match\_number

1. **LEFT JOIN**

Query

SELECT weather.temperature, tickets\_available.stadium\_id, tickets\_available.match\_id FROM tickets\_available

LEFT JOIN weather

ON tickets\_available.match\_number = weather.match\_number

1. **RIGHT JOIN**

Query

SELECT weather.temperature, tickets\_available.stadium\_id, tickets\_available.match\_id FROM tickets\_available

RIGHT JOIN weather

ON tickets\_available.match\_number = weather.match\_number

1. **FULL JOIN**

Query

SELECT weather.temperature, tickets\_available.stadium\_id, tickets\_available.match\_id FROM tickets\_available

FULL JOIN weather

ON tickets\_available.match\_number = weather.match\_number

1. **EXIST**

SELECT temperature, humidity FROM weather

WHERE EXISTS (SELECT \* FROM tickets\_available WHERE weather.match\_number=tickets\_available.match\_number

1. **ALL**

SELECT brand\_name FROM advertisement

WHERE match\_number=ALL (SELECT match\_number FROM tickets\_sold WHERE available>500)

1. **ANY**

SELECT sold FROM tickets\_sold

WHERE match\_number=ANY (SELECT match\_number FROM tickets\_available WHERE available>4900)

**Assignment – 5.7**

Study and implement the assignment topic from the Embedded SQL, PL SQL Concepts, Cursors, Stored Procedures, Stored Function, Triggers, Tuple relational calculus, domain relational calculus.

**Objective: - To Learn and implement advanced concepts of Embedded SQL**

**EXEC SQL** connect to **server** user **user\_name** using **password**;

// initially Connecting the database to the host language and required username and password to login

**Query – 1:-**

EXEC SQL BEGIN DECLARE SECTION;

Int age;

EXEC END DECLARE SECTION;

EXEC SQL

declare c cursor for

SELECT email\_id, first\_name FROM LOGIN

WHERE 50>: age;

EXEC SQL open c;

EXEC SQL fetch c into :se, :sn;

EXEC SQL close c;

**Query – 2:-**

EXEC SQL

declare c cursor for select match\_id, match\_number FROM fixtures

where city\_id>:5;

EXEC SQL open c;

EXEC SQL fetch c into :si, :sn;

EXEC SQL close c;

**Query – 3:-**

EXEC SQL

declare a cursor for select stadium\_id, available FROM tickets\_available

WHERE available>:750;

EXEC SQL open a;

EXEC SQL fetch into :si, :si;

EXEC SQL close a;

**Query – 4:-**

EXEC SQL

declare d cursor for select payment\_id, card\_number FROM card\_details

WHERE amount>:500;

EXEC SQL open d;

EXEC SQL fetch d into :si, :si;

EXEC SQL close d;

**Query – 5:-**

EXEC SQL

declare e cursor for select food\_id, seat\_number FROM food\_and\_beverages

where MRP>:100;

EXEC SQL open e;

EXEC SQL fetch e into :si, :si;

EXEC SQL close e;

**Query:-6:-**

EXEC SQL

Declare j cursor for stadium\_id, stadium\_name, FROM stadium\_details

WHERE capacity>:450;

EXEC SQL open j;

EXEC SQL fetch j into :si, :sn;

EXEC SQL close j;