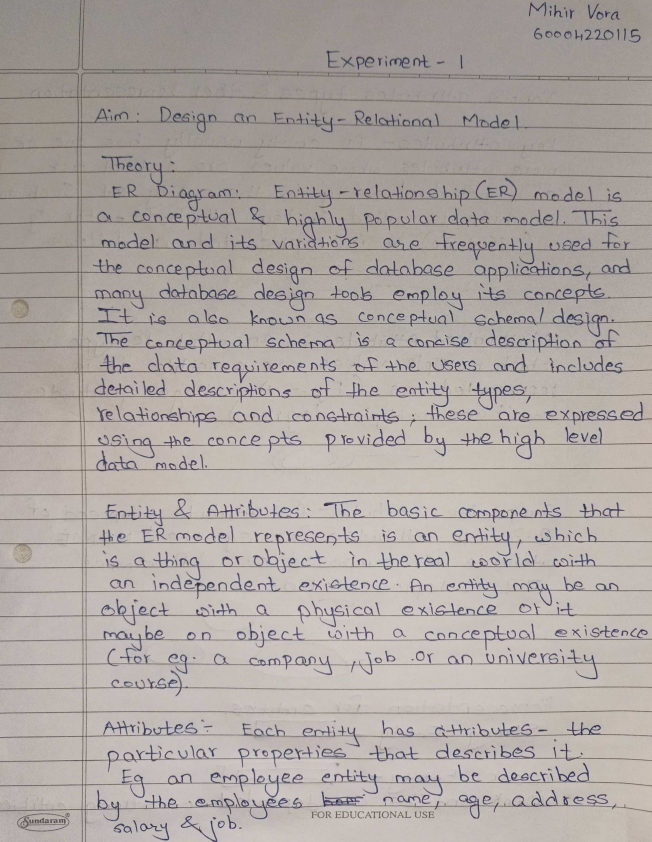
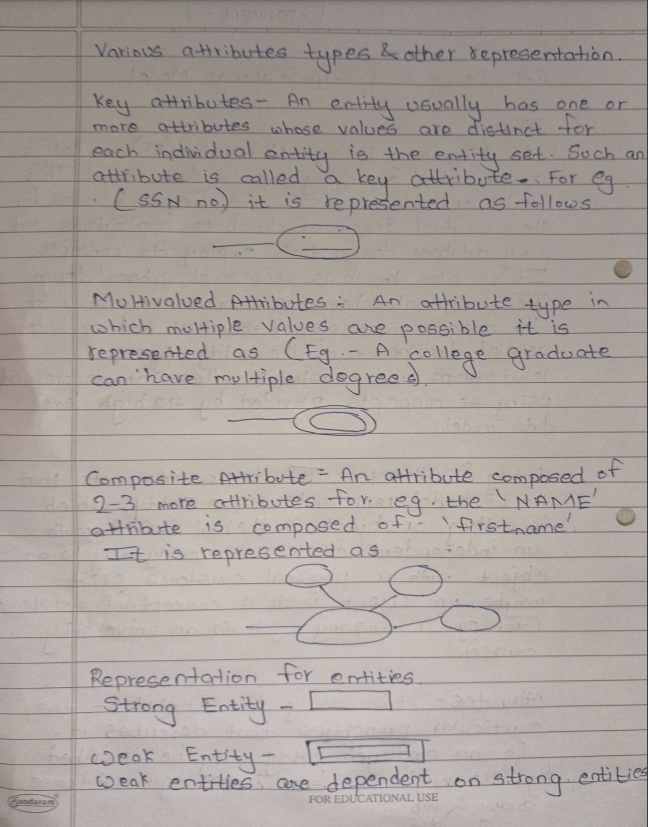
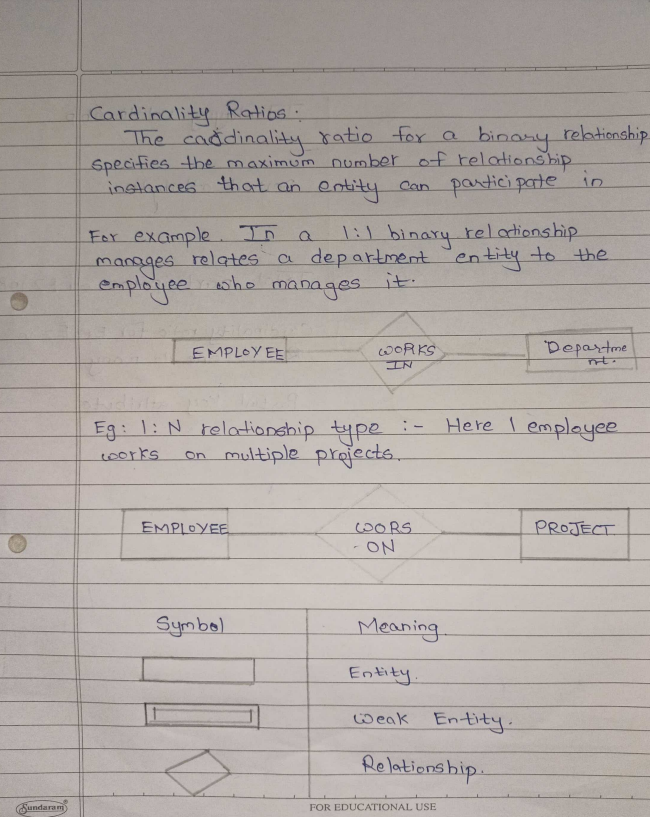
NAME : MIHIR Ashish VORa

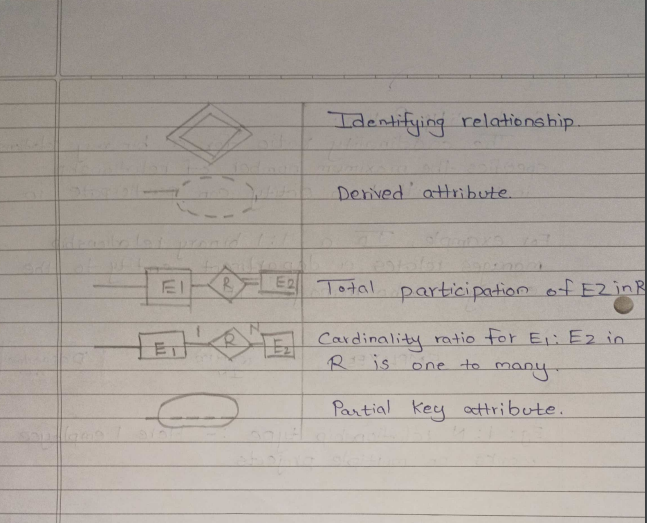
SAP ID : 60004220115

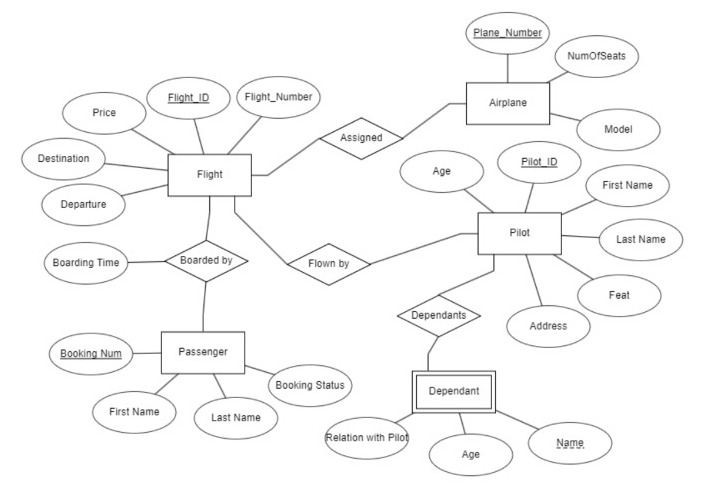
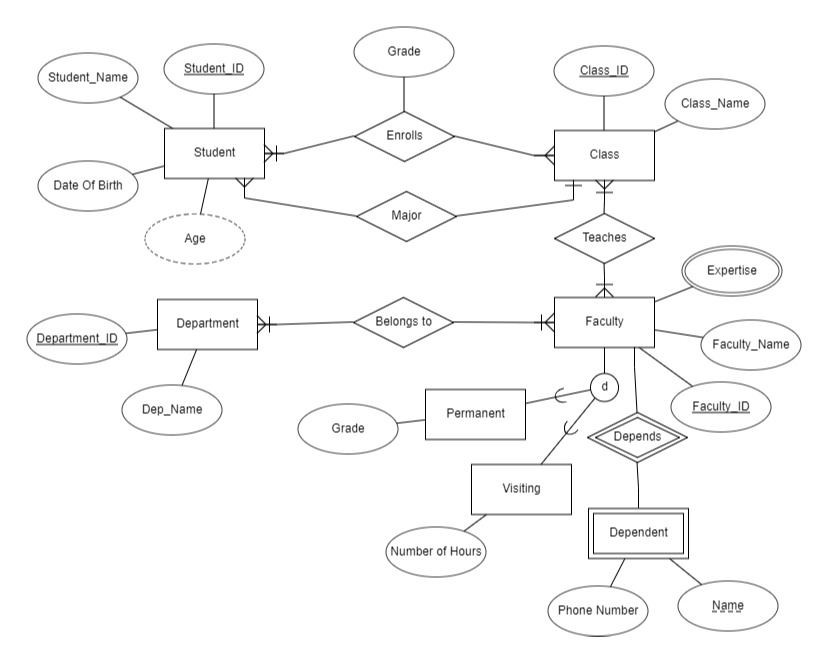
TOPIC : DBMS PRACTICAL 1









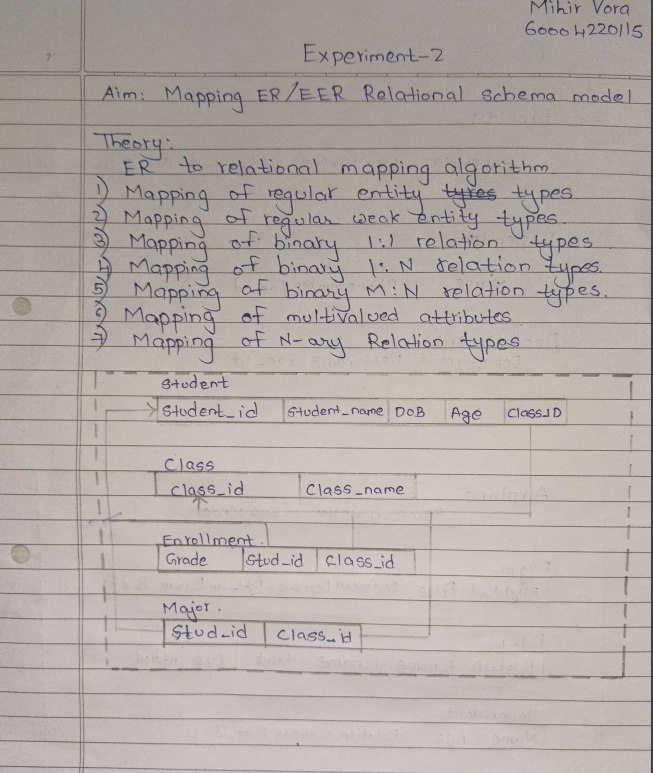


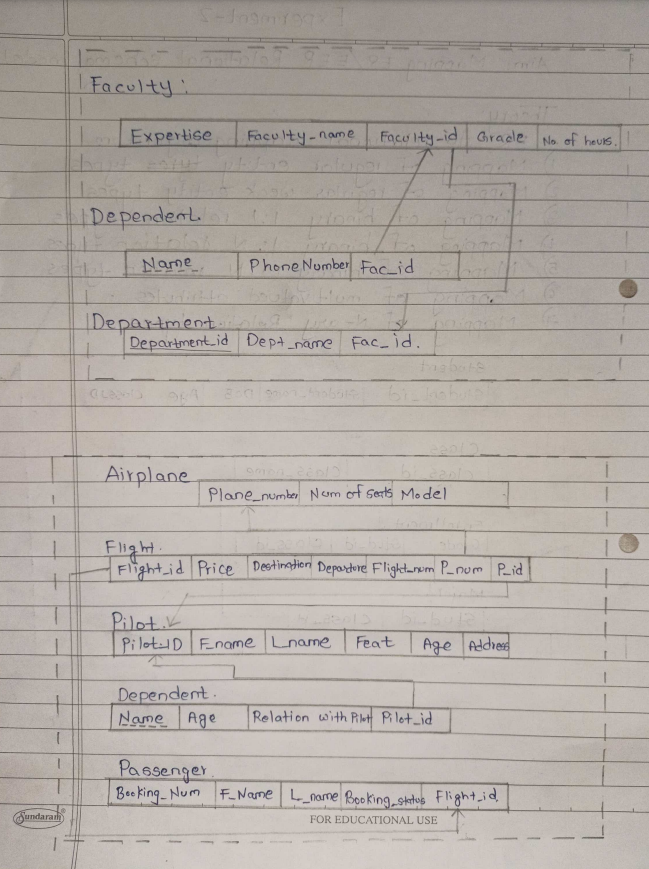
Conclusion: Thus, we designed an entity relationship model.

NAME : MIHIR Ashish VORa

SAP ID: 60004220115

TOPIC : DBMS PRACTICAL 2





Conclusion: Thus, we mapped ER/EER diagram Relational Schema model

NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 3

Execution DATE : 10-10-2022

Submission date : 26-11-2022

### Experiment No-3

**Aim:** Create and populate database using Data Definition Language (DDL) and DML Commands for your specified System.

**Theory:**

**DDL-** Data Definition Language (DDL) statements are used to define the database structure or schema. Data Definition Language understanding with database schemas and describes how the data should consist in the database, therefore language statements like CREATE TABLE or ALTER TABLE belongs to the DDL. DDL is about "metadata".

DDL includes commands such as CREATE, ALTER and DROP statements. DDL is used to CREATE, ALTER OR DROP the database objects (Table, Views, Users).

 Data Definition Language (DDL) are used different statements:

1. CREATE - to create objects in the database
2. ALTER - alters the structure of the database
3. DROP - delete objects from the database
4. TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
5. RENAME - rename an object

Use Following schema to perform the experiment.

**Student** (stuId, lastName, firstName, major, credits)

**Faculty** (facId, name, department, rank) **Class** (classNumber, facId, schedule, room) **Enroll** (classNumber, *stuId*, grade)

NOTE: Underlined Text: Primary Key, Italic Text: Forign Key

#### DDL (Data Definition Language)

1. **CREATE** 
   1. **To create a database**

CREATE DATABASE *dbname*; Eg-

CREATE DATABASE UniversityData;

* 1. **To select existing database**

Use dbname;

Eg- USE UniversityData;

* 1. **To create a Table**

The create table command defines each column of the table uniquely. Each column hasminimum of three attributes.

* + - Name
    - Data type
    - Size (column width).

Each table column definition is a single clause in the create table syntax. Each table column definition is separated from the other by a comma. Finally, the SQL statementis terminated with a semicolon.

**Syntax:** Create table table name( fieldname1 datatype(),fieldname2 datatype()...);

1. **ALTER**

By The use of ALTER TABLE Command we can **modify** our exiting table.

* + **Adding New Columns**

**Syntax:**

**ALTER TABLE <table\_name>**

**ADD (<NewColumnName> <Data\_Type>(<size>),....... n)**

Example: Add a new column,cTitle, to our Classtable ALTER TABLE Class ADD cTitle CHAR(30);

The schema of the Class table would then be: Class(classNumber,facId,schedule,room,cTitle)

**Dropping a Column from the Table**

**Syntax:**

**ALTER TABLE <table\_name> DROP COLUMN <column\_name> Example:**

Example: Drop the cTitle column from the Class table ALTER TABLE Class DROP COLUMN cTitle;

This command will drop particular column

If we want to add, drop, or change a constraint, we can use the same ALTER TABLE command. For example, if we created the Class table and neglected to make facIda foreign key in Class, we could add the constraint at any time by writing:

ALTER TABLE Class ADD CONSTRAINT Class\_facId\_fk FOREIGN KEY (facId)REFERENCES Faculty (facId)ON DELETE NO ACTION);

We could drop an existing named constraint using the ALTER TABLE command.

For example, to drop the check condition on the creditsattribute of Student that we created earlier, we could write:

ALTER TABLE Student DROP CONSTRAINT Student\_credits\_cc;

* + **Modifying Existing Column**

**Syntax:**

ALTER TABLE <table\_name> MODIFY <column\_name> NewDataType>(<NewSize>) Example:

ALTER TABLE Student MODIFY stuld Varchar(20);

* + **Renaming Existing Table**

**Syntax:**

**ALTER TABLE <table\_name> RENAME <new\_table\_name>**

Example:

ALTER TABLE student RENAME new\_student;

ALTER TABLE new\_student RENAME student; (To revert the change)

1. **RENAME Syntax:**

RENAME TABLE <OldTableName> TO <NewTableName>

Example:

RENAME table visiting TO visiting\_staff;

1. **DROP Syntax:**

DROP TABLE <table\_name>

Example:

DROP TABLE visiting\_staff;

1. **TRUNCATE Syntax:**

TRUNCATE TABLE <Table\_name> Example:

TRUNCATE TABLE visiting\_staff;

1. **SHOW**

To check available databases and tables

**Syntax**

SHOW DATABASES; SHOW TABLES;

1. **DESCRIBE**

To obtain information about table structure or query execution plans.

**DESCRIBE < table\_name> DESC < table\_name>**

Example- DESC Student;

* **Apply Integrity Constraints for the specified system**

MySQL CONSTRAINT is used to define rules to allow or restrict what values can be stored in columns. The purpose of inducing constraints is to enforce the integrity of a database.

MySQL CONSTRAINTS are used to limit the type of data that can be inserted into a table. MySQL CONSTRAINTS can be classified into two types - column level and table level.

The column level constraints can apply only to one column where as table level constraints are applied to the entire table.

MySQL CONSTRAINT is declared at the time of creating a table.

**MySQL CONSTRAINTs are :**

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT
* AUTO INCREMENT

|  |  |
| --- | --- |
| **CONSTRAINT** | **DESCRIPTION** |
| NOT NULL | In MySQL NOT NULL constraint allows to specify that a column can not contain any NULL value. MySQL NOT NULL can be used to CREATE and ALTER a table.  Eg. lastName CHAR(20) NOT NULL,  firstName CHAR(20) NOT NULL |
| UNIQUE | The UNIQUE constraint in MySQL does not allow to insert a duplicate value in a column. The UNIQUE constraint maintains the uniqueness of a column in a table. More than one UNIQUE column can be used in a table.  Eg. CONSTRAINT Class\_schedule\_room\_uk UNIQUE (schedule, room) |
| PRIMARY KEY | A PRIMARY KEY constraint for a table enforces the table to accept unique data for a specific column and this constraint creates a unique index for accessing the table faster.  Eg: CONSTRAINT Faculty\_facId\_pk PRIMARY KEY (facId)); |
| FOREIGN KEY | A FOREIGN KEY in MySQL creates a link between two tables by one specific column of both tables. The specified column in one table must be a PRIMARY KEY and referred by the column of another table knownas FOREIGN KEY.  Eg. CONSTRAINT Class\_facId\_fk FOREIGN KEY (facId) REFERENCES Faculty (facId) ON DELETE NO ACTION); |
| CHECK | A CHECK constraint controls the values in the associated column. The CHECK constraint determines whether the value is valid or not from a logical expression.  Eg: CONSTRAINT Student\_credits\_cc CHECK ((credits =0) AND (credits 150); |
| DEFAULT | In a MySQL table, each column must contain a value ( including a NULL). While inserting data into a table, if no value is supplied to a column, then the column gets the value set as DEFAULT.  credits SMALLINT DEFAULT 0 CHECK ((credits =0) AND (credits 150)) |
| AUTO INCREMENT | Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Often this is the primary key field that we would like to be created automatically every time a new record is inserted.  Eg. Field name int AUTO\_INCREMENT PRIMARY KEY |

#### DML(Data Manipulation Language)

A data manipulation language (DML) is a family of computer languages including commands permitting users to manipulate data in a database. This manipulation involves inserting data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data. DML is mostly incorporated in SQL databases.

Use following database to perform the experiment.

**Database: 1) INSERT**

 This command adds one or more records to a database table.

**Syntax**

INSERT INTO "table\_name" ("column1", "column2", ...) VALUES ("value1", "value2", ...);

**Example**

1) insert into Student (stuld, lastname, firstname, major, credits) values(‘S1001’,'Smith',Tom',

'History', 90);

1. **SELECT**

The SELECT statement is used to select data from a database.

**Syntax**

SELECT \* FROM *table\_name*;

**Example-**

Select \* from Student;

1. **UPDATE**

The UPDATE statement is used to update existing records in a table.

**Syntax**

UPDATE *table\_name* SET *column1*=*value1*, *column2*=*value2*,... WHERE *some\_column*=*some\_value*;

**Example-**

* 1. update Student set Name='JANEE' where stuID=S1020;

1. **DELETE**

This command removes one or more records from a table according to specified conditions.

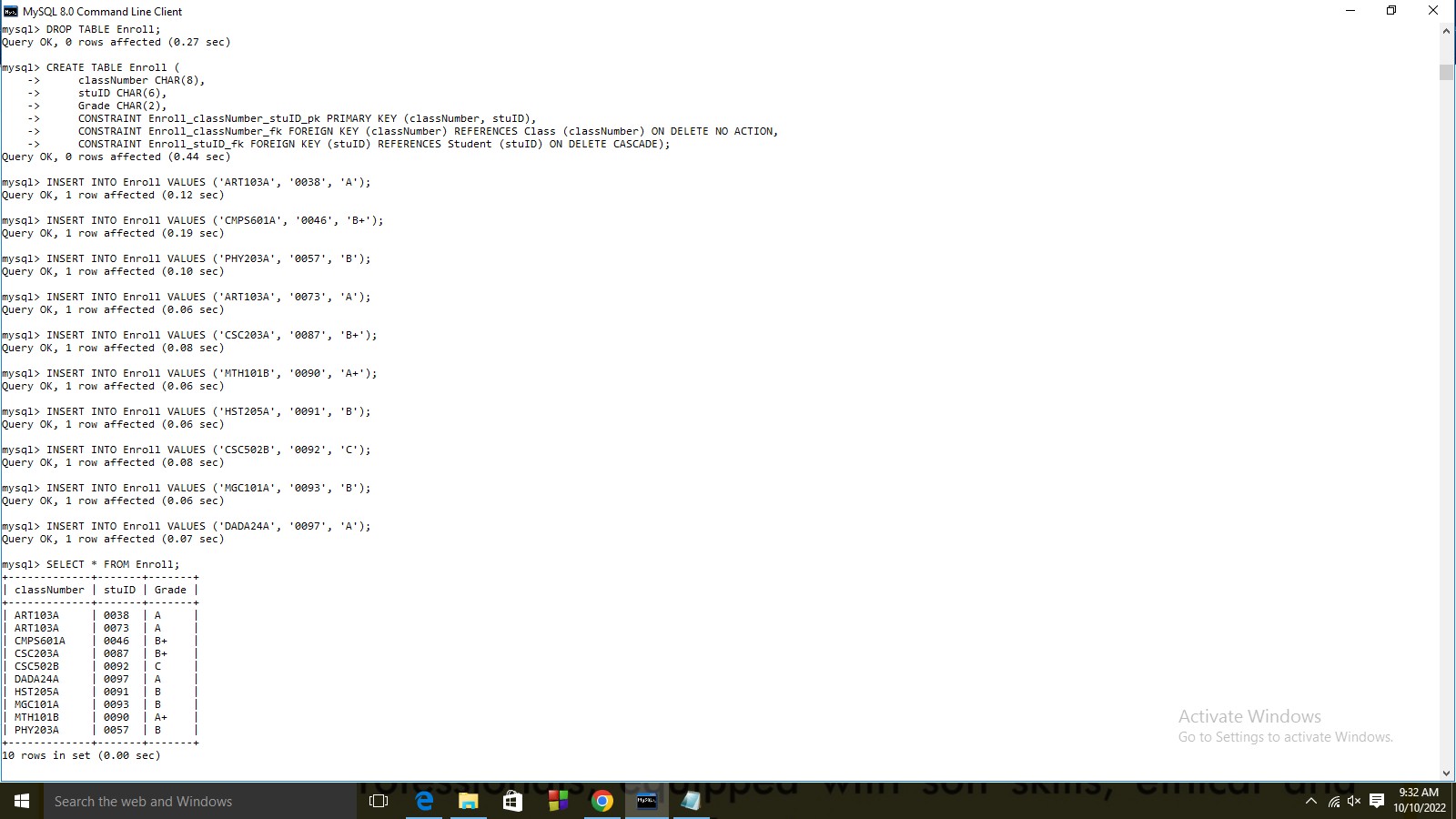
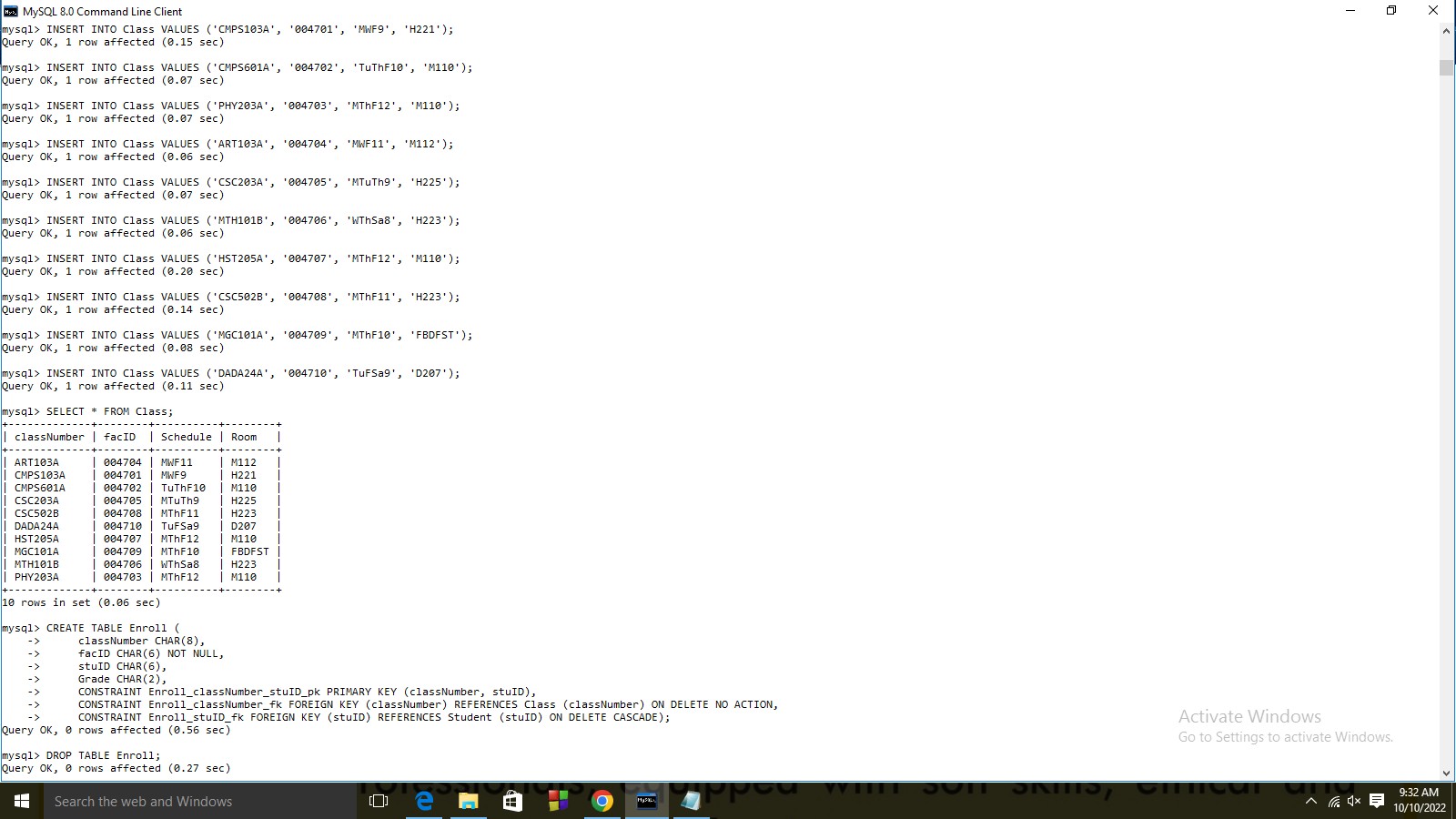
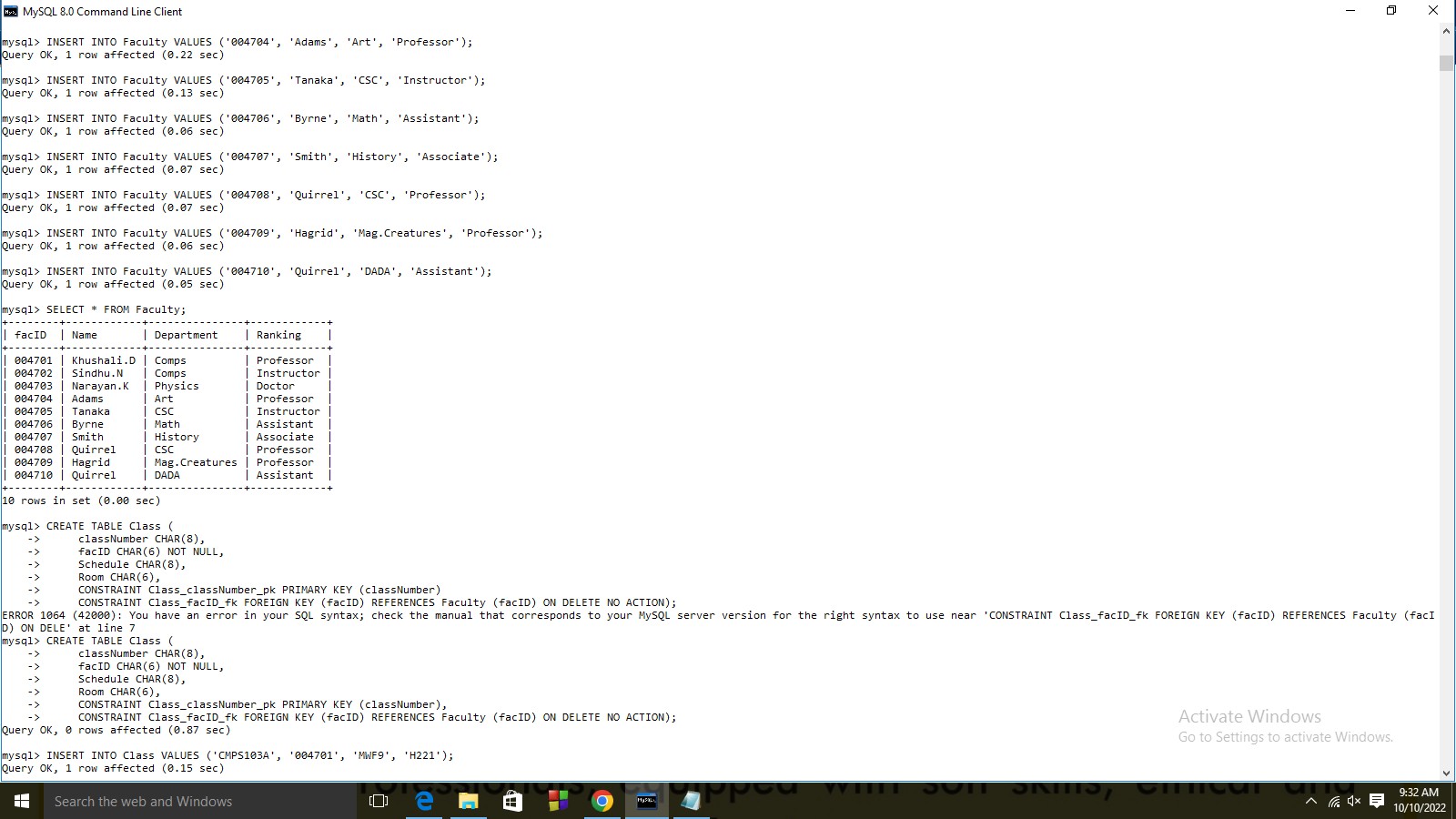
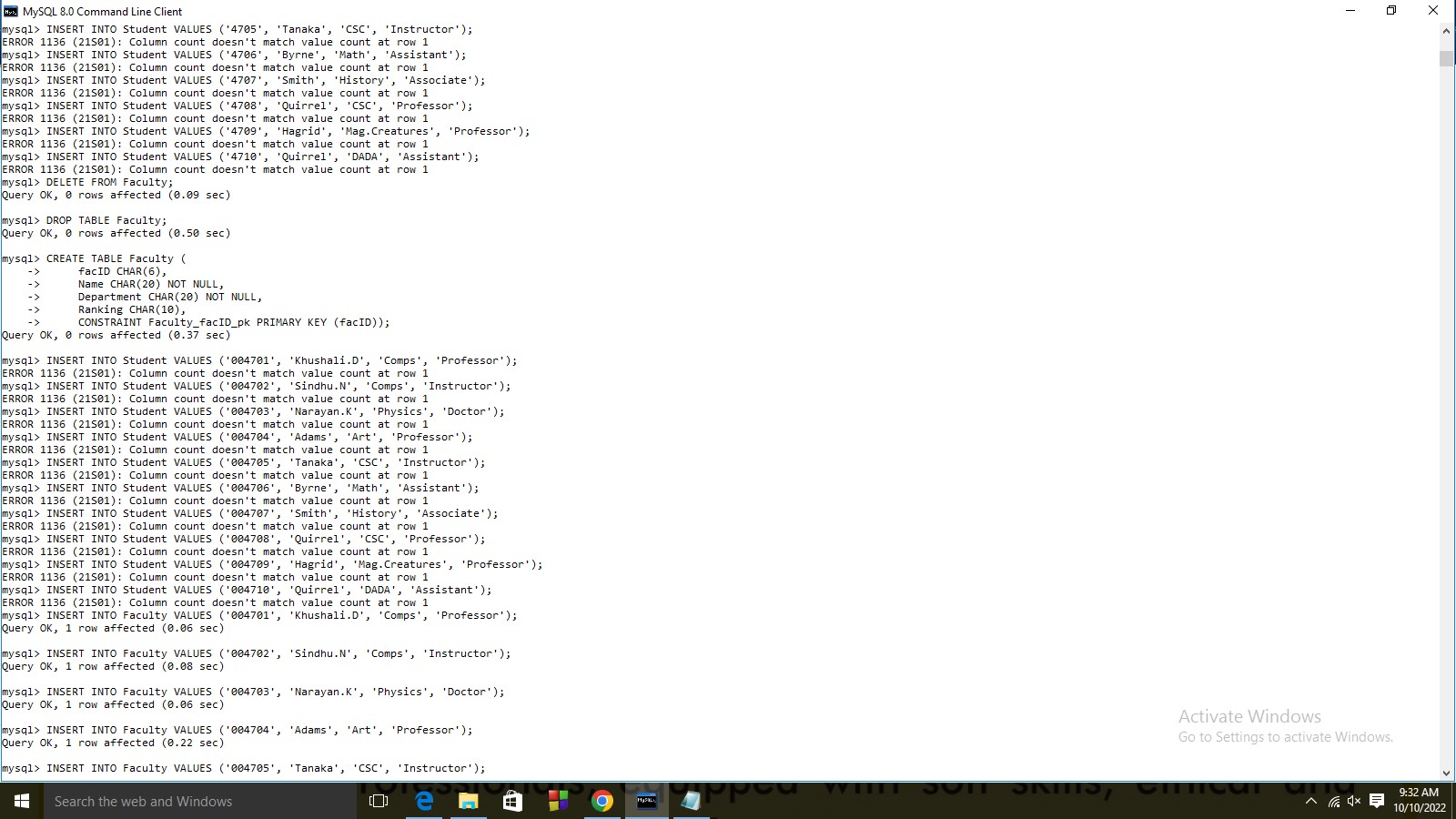
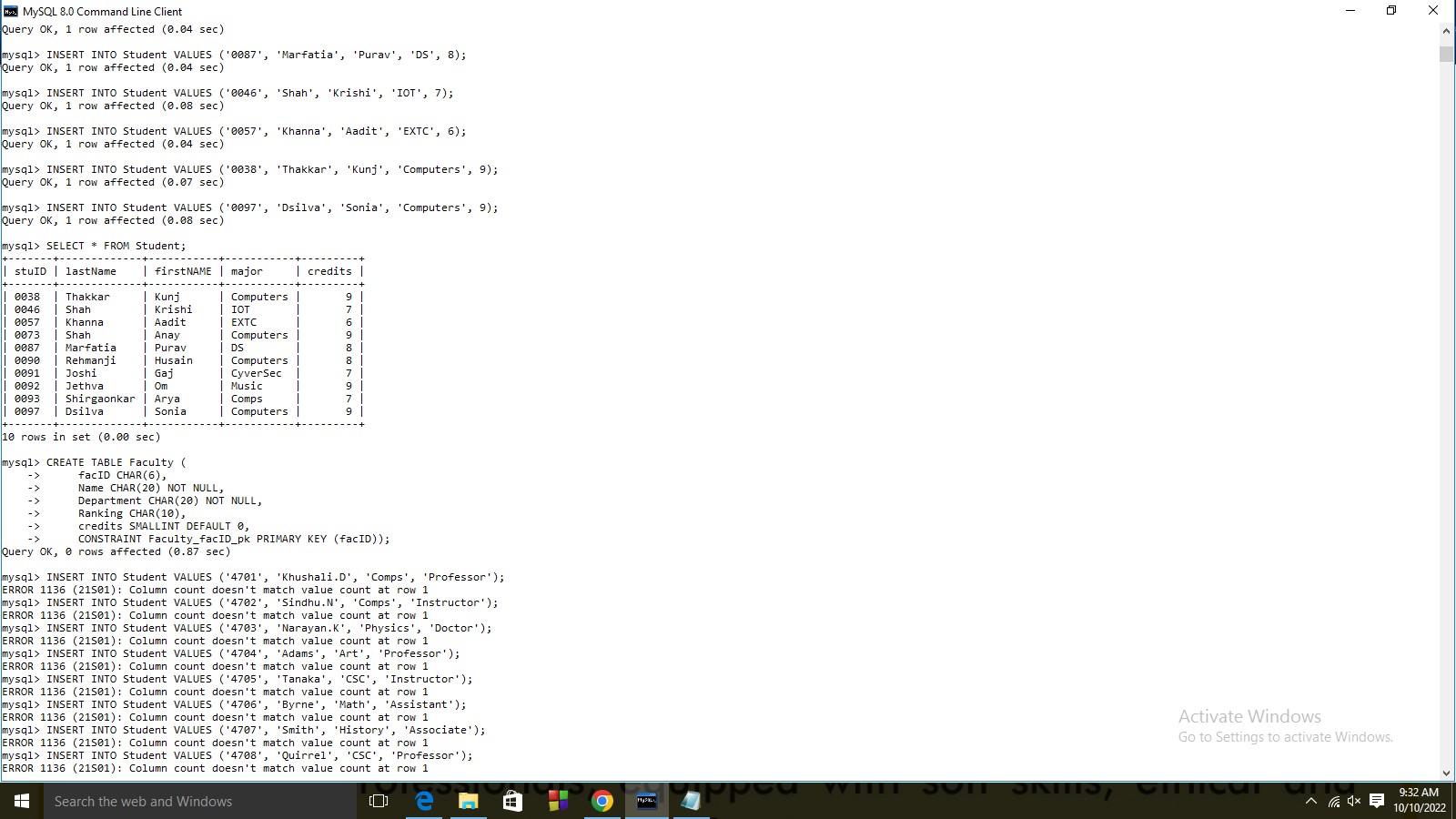
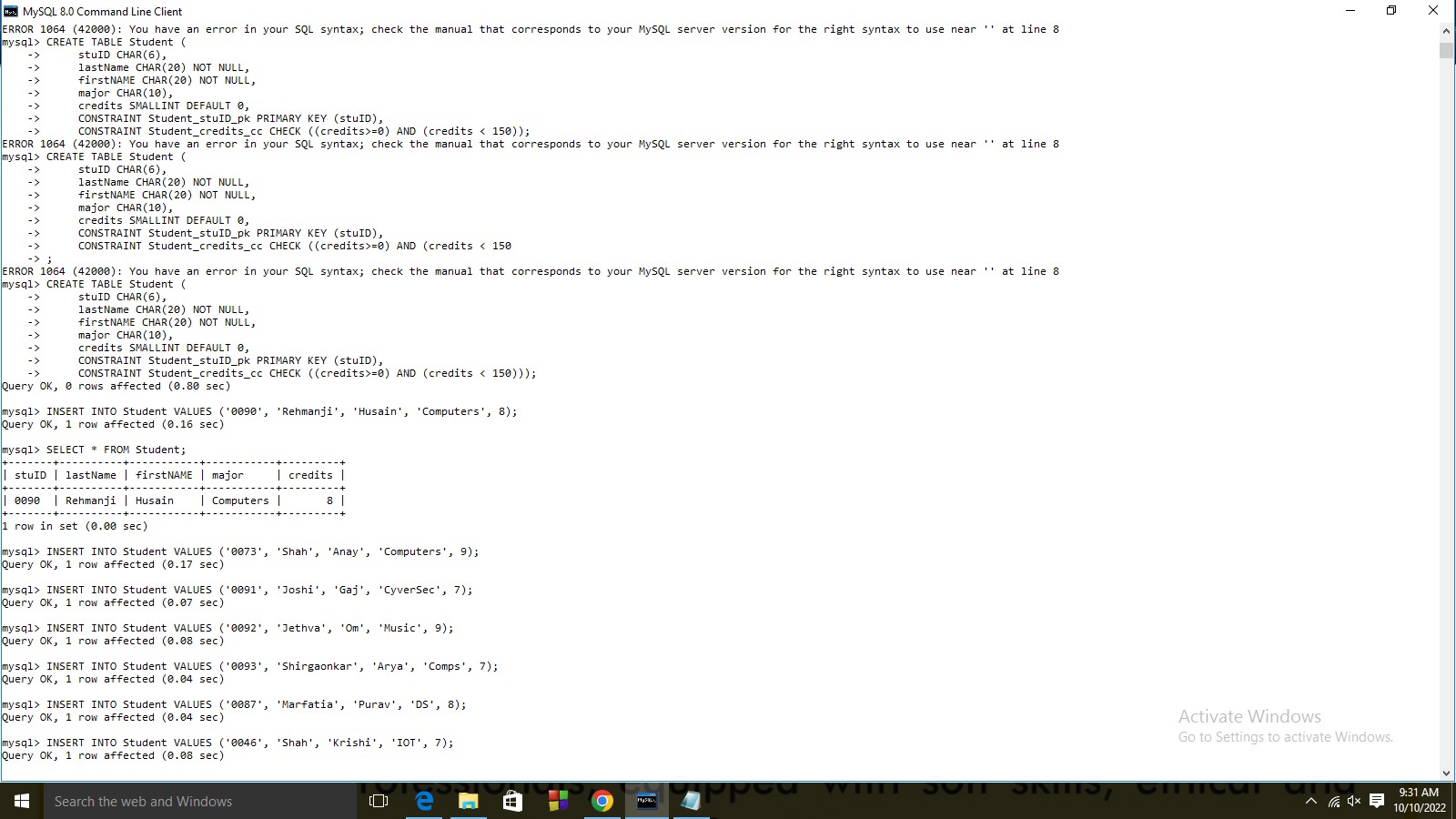
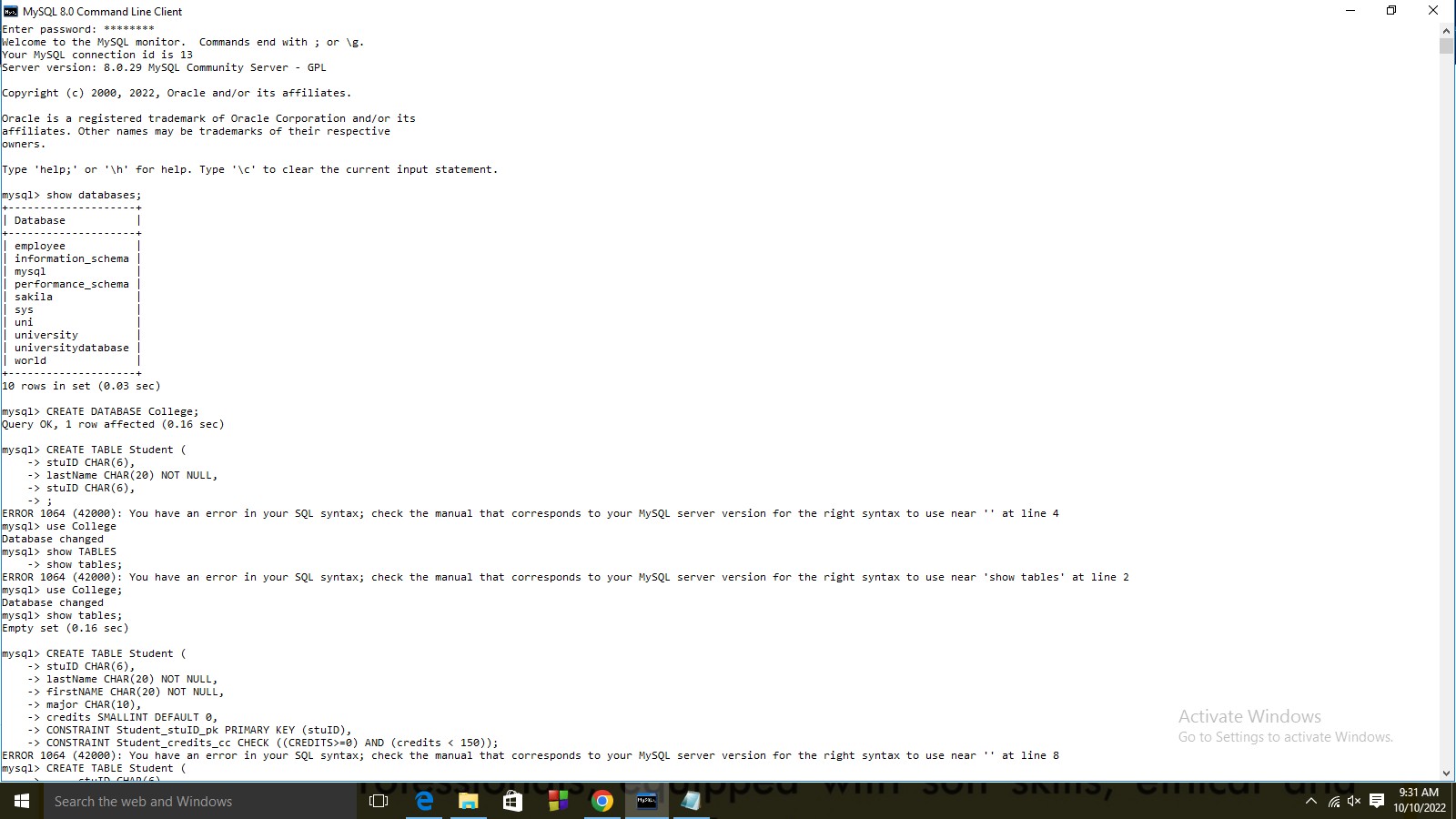
**Syntax:** DELETE FROM *table\_name* WHERE *some\_column*=*some\_value*;

**Example-**1) delete from Student

where stuID=S1020;

**Conclusion:** Data definition Language is used to create database and apply Integrity Constraints for the project and used DML to populate database.

**Screenshots on MYSQL:**



## -Mihir Vora 60004220115

NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 4

Submission date : 28-11-2022

### Experiment No-4

**AIM:** To perform simple queries and string manipulation operations.

#### THEORY:

A query is a request for data or information from a database table or combination of tables. This data may be generated as results returned by Structured Query Language (SQL) or as pictorials, graphs or complex results, e.g., trend analyses from data- mining tools.

For a machine to understand a request for information in the first place, the query must be written according to a code known as query language. SQL represents one of the standard languages used for database management purposes, while MySQL, instead, is the software using that specific language. Although SQL is a fairly universal query language, other commonly used ones include DMX, Datalog and AQL.

The most commonly used SQL command is SELECT statement. SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT Statement, you must specify the column(s) name and the table name. The whole query is called SQL SELECT Statement.

**Syntax-**

SELECT [DISTINCT|ALL ] { \* | [fieldExpression [AS newName]} FROM tableName [alias] [WHERE condition][GROUP BY fieldName(s)] [HAVING condition] ORDER BY fieldName(s) .

* + **SELECT** is the SQL keyword that lets the database know that you want to retrieve data.
  + **[DISTINCT | ALL]** are optional keywords that can be used to fine tune the results returned from the SQL SELECT statement. If nothing is specified then ALL is assumed as the default.
  + **{\*| [fieldExpression [AS newName]}** at least one part must be specified, "\*" selected all the fields from the specified table name, fieldExpression performs some computations on the specified fields such as adding numbers or putting together two string fields into one.
  + **FROM** tableName is mandatory and must contain at least one table, multiple tables must be separated using commas or joined using the JOIN keyword.
  + **WHERE** condition is optional, it can be used to specify criteria in the result set returned from the query.
* **GROUP BY** is used to put together records that have the same field values.
* **HAVING** condition is used to specify criteria when working using the GROUP BY keyword.
* **ORDER BY** is used to specify the sort order of the result set.

**Queries:**

1. Select all column from a table Select \* from tablename;
2. Select specific column of list from a table Select col1, col2 from tablename;
3. Select …. where clause with various operators (<,>,<=,> =,IN,NOT IN,BETWEEN..AND,NOT BETWEEN… AND)

Select col1 from tablename where col2>value1;

Select \* from tablename where col2 in(value1,value2,value3) Select \* from tablename where col2 between value1 and value2

Select \* from tablename where col2 NOT between value1 and value2

1. Select ….where clause with multiple conditions

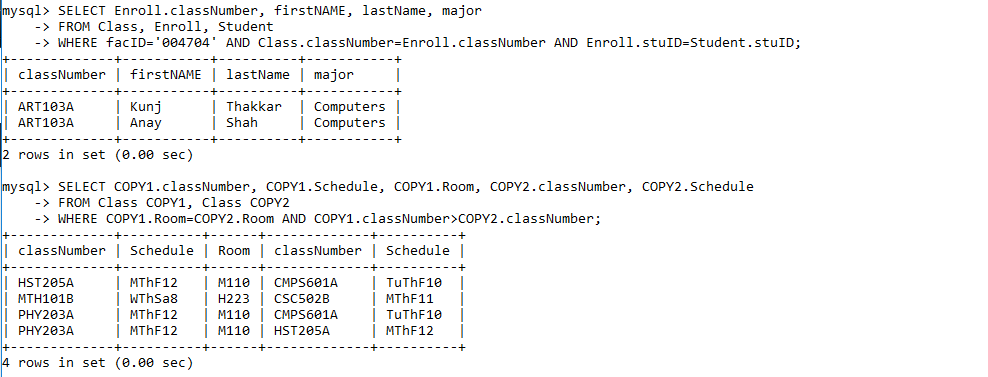
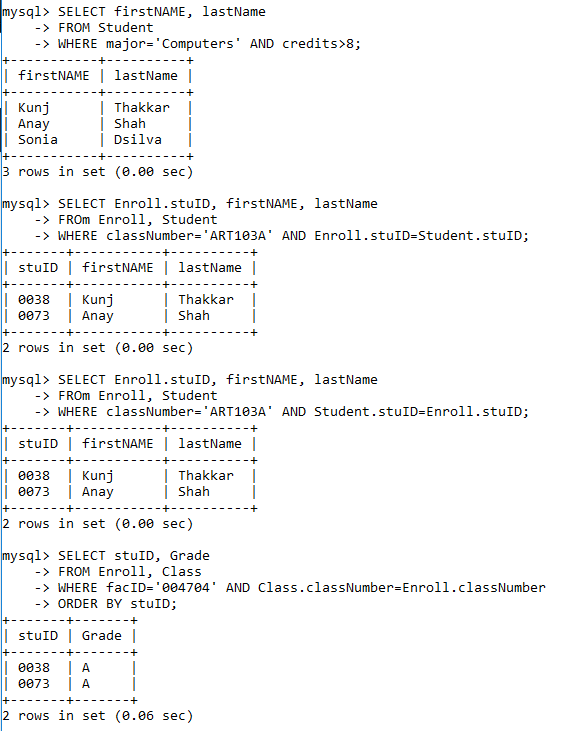
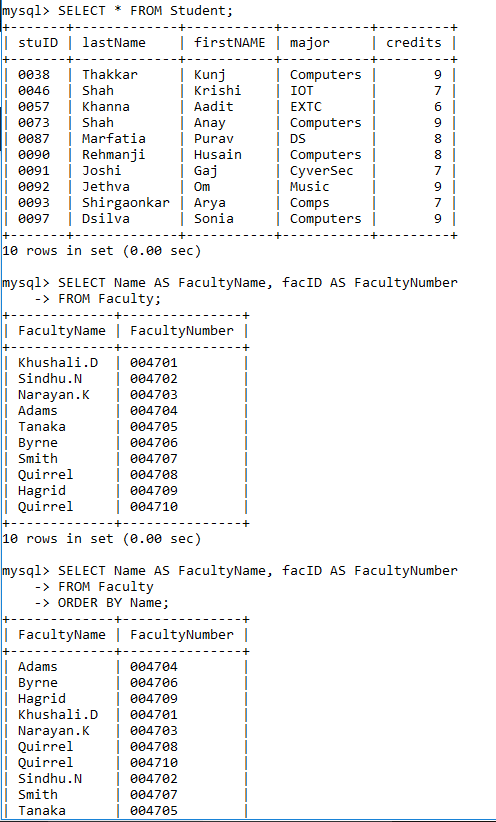
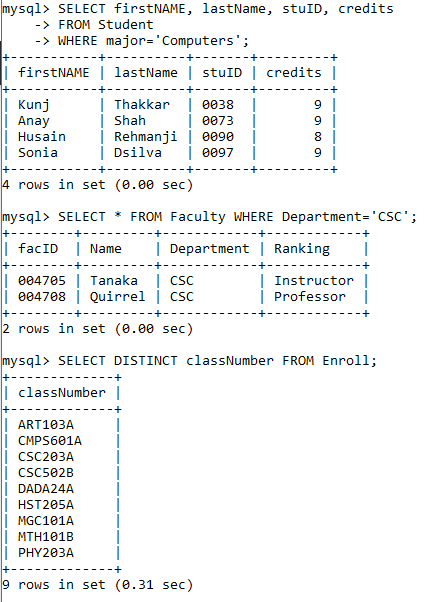
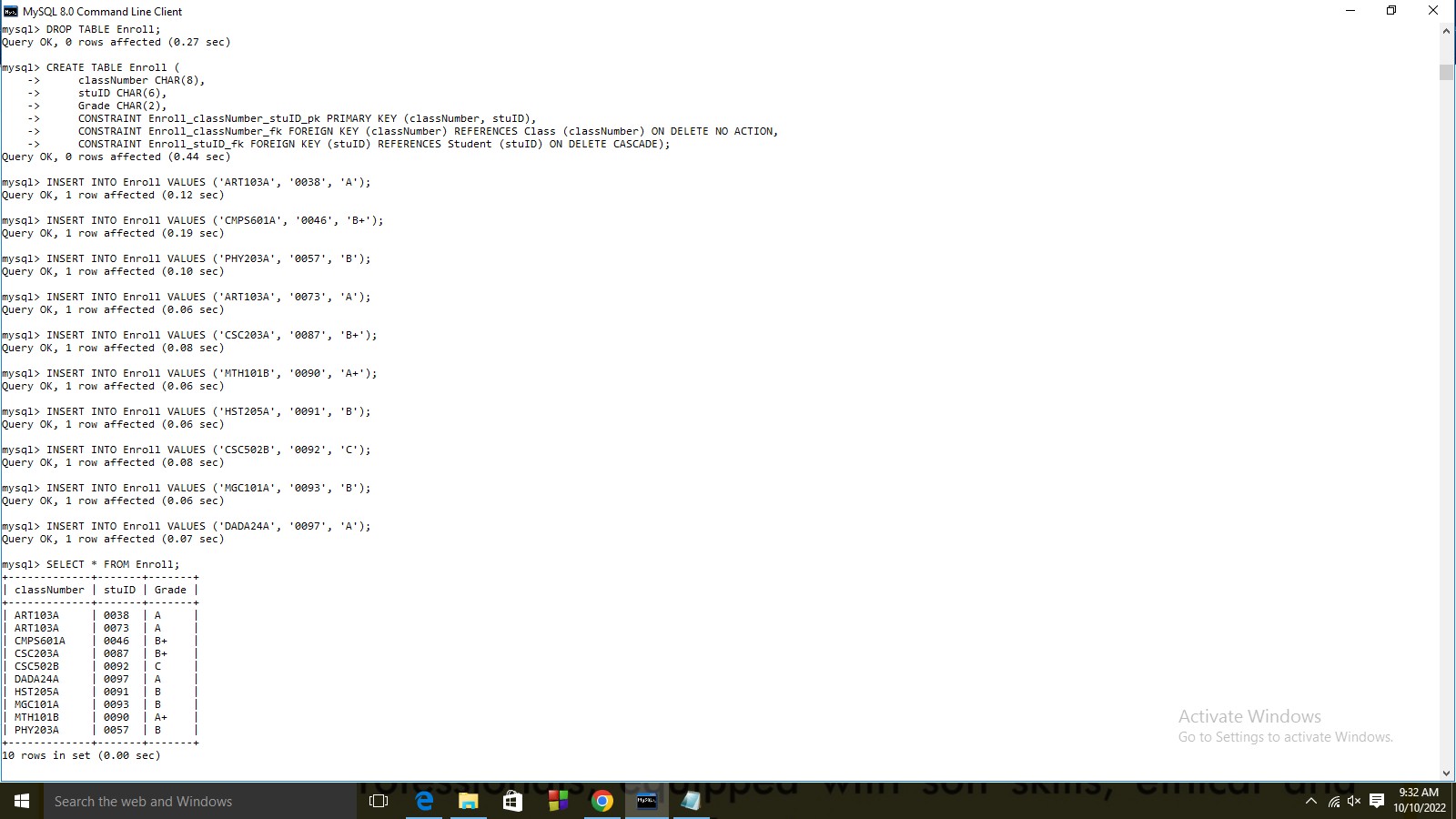
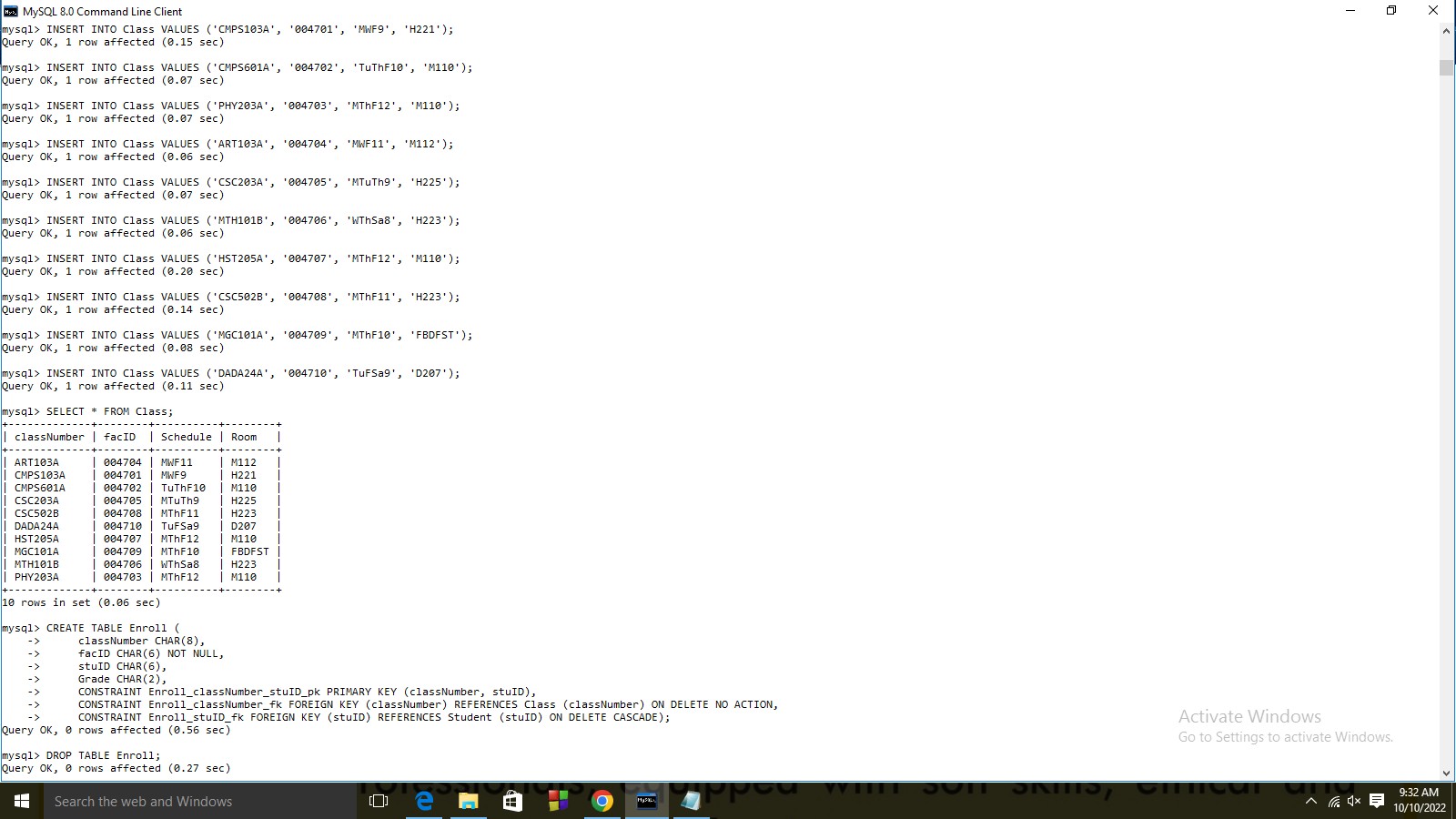
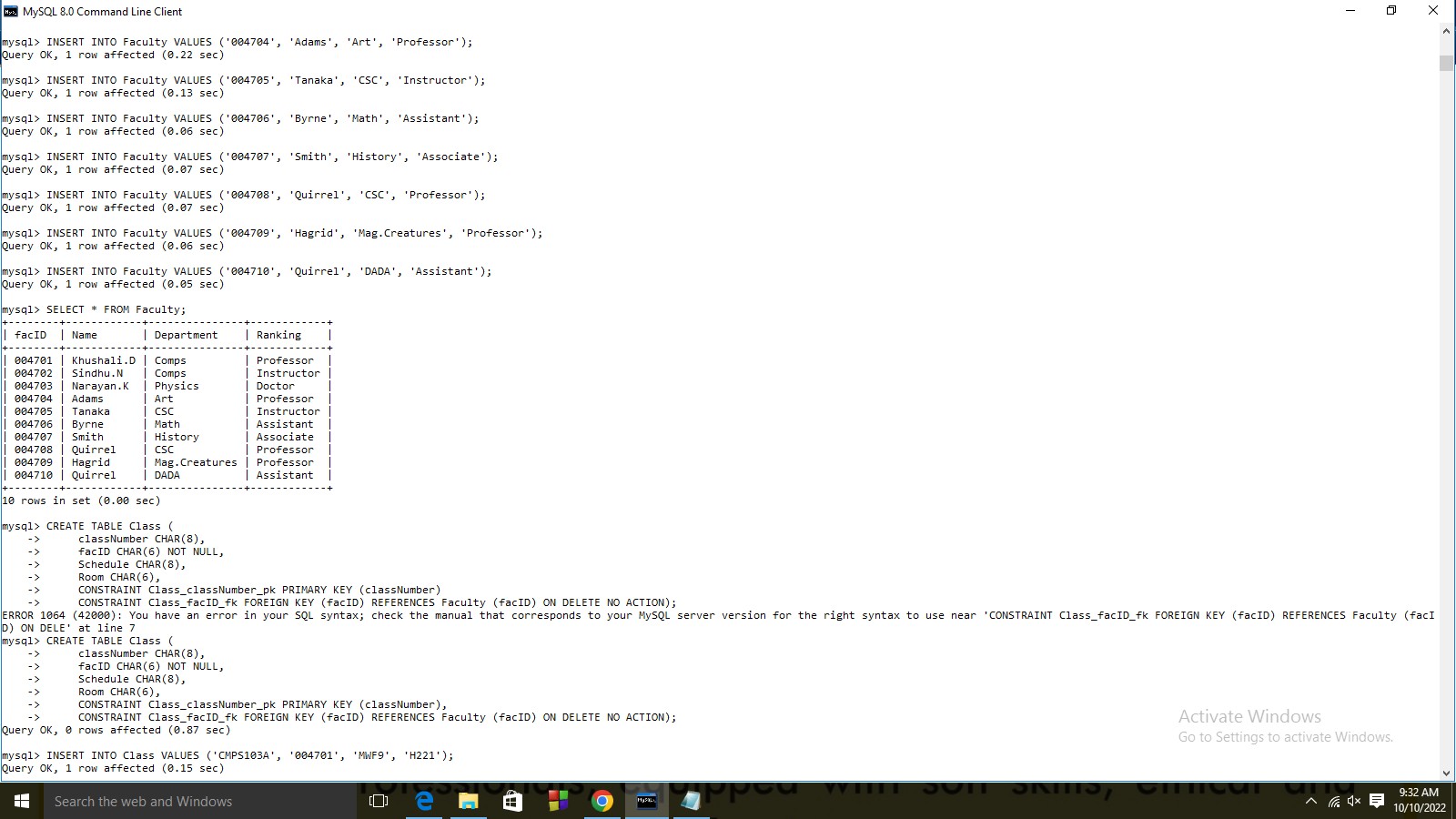
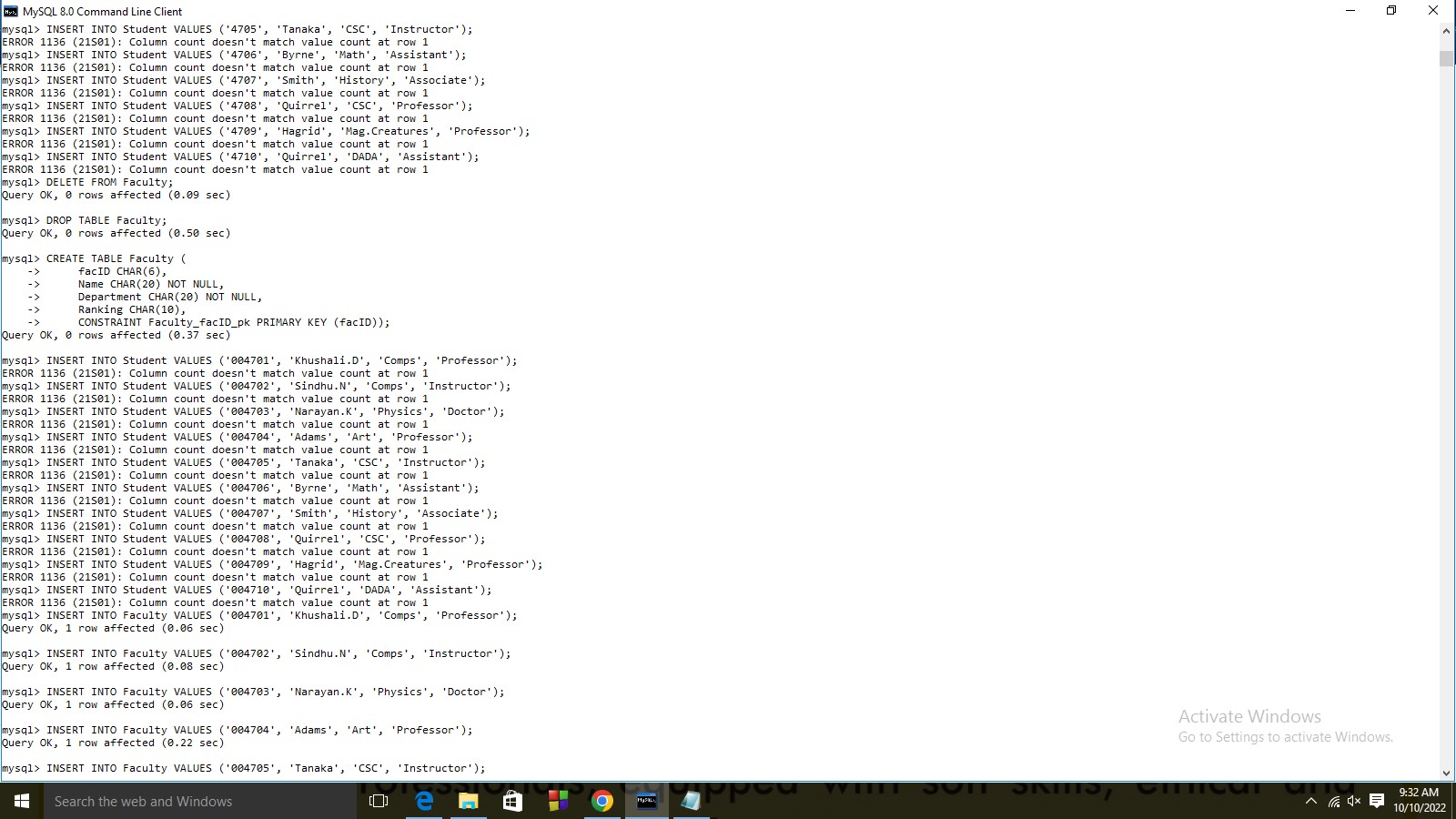
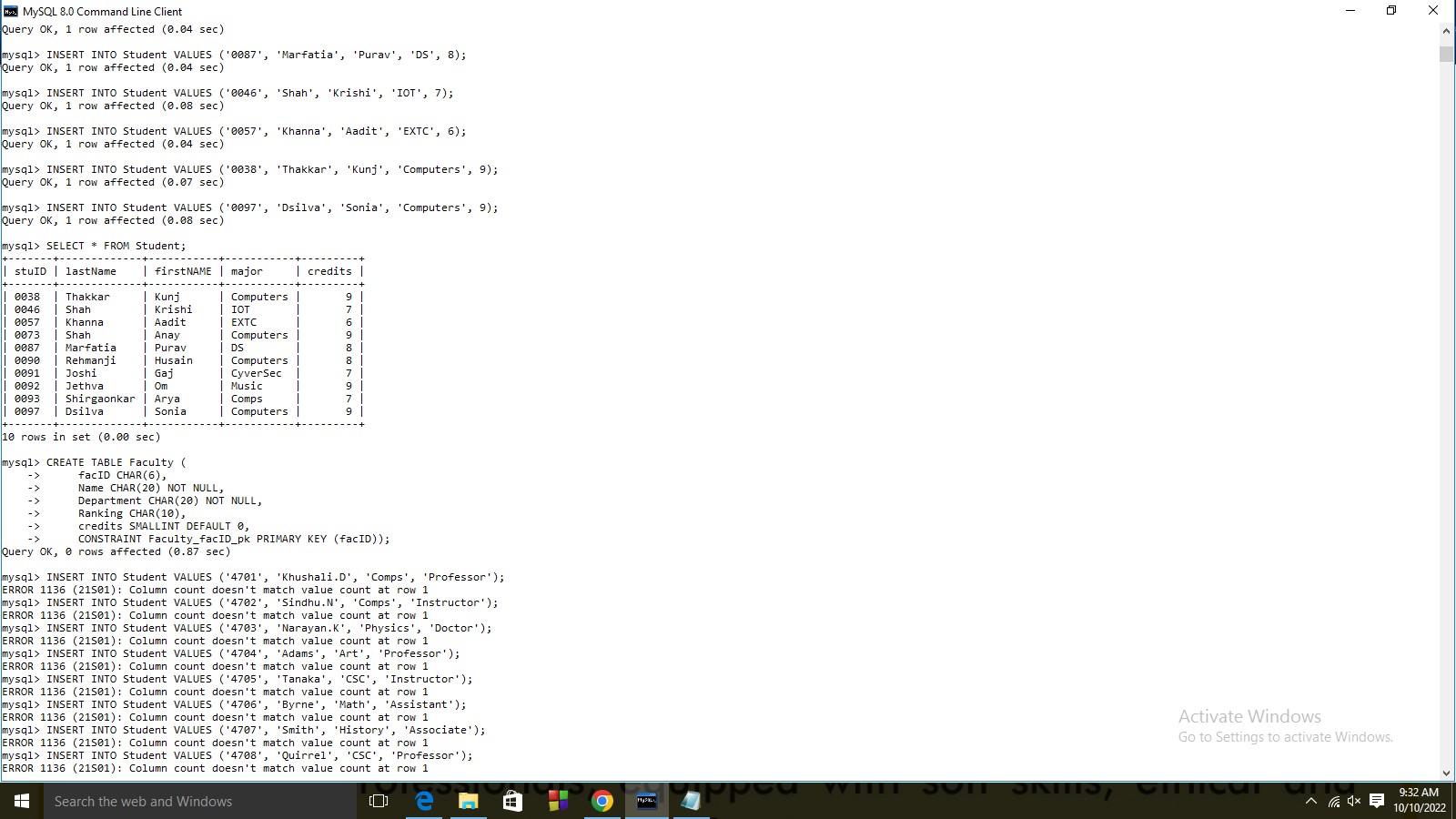
Select \* from tablename where col2=Value1 and/or col3=value2

1. Select …. where clause with string matching
   * Starts with any character
   * ends with any character
   * have a specific substring
   * character at specific position
   * starts with a specific character and having specifically n no of characters
   * starts and ends with different character
2. Query your database by applying aggregate function MIN, MAX, COUNT, AVG, SUM (create alias also)
3. Query your database by applying group by clause using one column and multiple column.
4. Query your database by applying order by clause using one column and multiple column
5. Query your database by applying group by, order by and having clause simultaneously
6. Select clause with various date functions:

CURDATE(),CURRENT\_TIME(),CURRENT\_TIMESTAMP(),DATE(),DATEDIF F(),DAY(),DAYNAME(),EXTRACT(),MINUTE(),MONTH(),WEEK(),NOW(),YE AR()

1. Perform set operation on multiple select queries(UNION)

**Screenshots on MYSQL:**



## -Mihir Vora - 60004220115

NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 5

Submission date : 28-11-2022

### Experiment No-5

**AIM:** Perform Nested queries and Complex queries.

#### THEORY:

A Subquery or Inner query or 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.

SELECT column\_name [, column\_name ] FROM table1 [, table2 ]

WHERE column\_name OPERATOR ( SELECT column\_name [, column\_name ] FROM table1 [, table2 ]

[WHERE])

#### Example: Using a Subquery with Equality

**Question:Find the numbers of all the courses taught by Byrne of the math department.**

**Solution**:We already know how to do this by using a natural join, but there is another way of finding the solution. Instead of imagining a join from which we choose records with the same facId, we could visualize this as two separate queries. For the first one, we would go to the Faculty table and find the record with name of Byrne and department of Math.

We could make a note of the corresponding facId. Then we could take the result of that query, namely Fll0, and search the Class table for records with that value in facId. Once we found them, we would display the classNumber. SQL allows us to sequence these queries so that the result of the first can be used in the second.

A subquery can be used in place of a join, provided the result to be displayed is contained in a single table and the data retrieved from the subquery con-sists of only one column. When you write a subquery involving two tables, you name only one table in each SELECT. The query to be done first, the subquery, is the one in parentheses, following the first WHERE line. The main query is performed using the result of the subquery. Normally you want the value of some field in the table mentioned in the main query to match the value of some field from the table in the subquery. In this example, we knew we would get only one value from the subquery, since facId is the key of Faculty, so a unique value would be produced. Therefore, we were able to use equality as the operator.

Since the sub-query is performed first, the SELECT . . . FROM . . . WHERE of the subquery is actually replaced by the value retrieved.

#### Example: Subquery Using ‘IN’

**Question:Find the names and IDs of all Faculty members who teach a class in Room H221.**

**Solution**:We need two tables, Class and Faculty, to answer this question. We also see that the names and IDs both appear on the Faculty table, so we have a choice of a join or a subquery. If we use a subquery, we begin with the Class table to find facId values for any courses that meet in Room H221. We find two such entries, so we make a note of those values. Then we go to the Facultytable and compare the facId value of each record on that table with the two ID values from Class, and display the corresponding facId and name.

In the WHERE line of the main query we used IN, rather than =, because the result of the subquery is a set of values rather than a single value. We are saying we want the facId in Faculty to match any member of the set of values we obtain from the subquery.

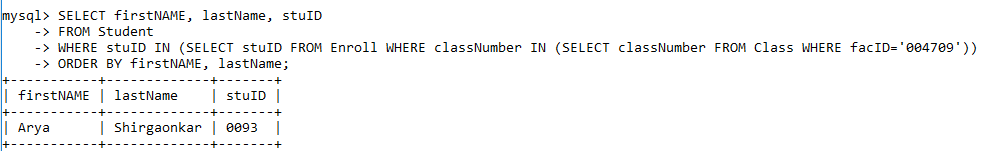
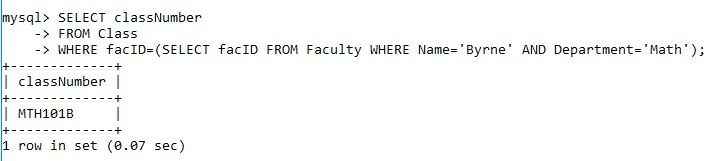
The IN is a more general form of subquery than the comparison operator, which is restricted to the case where a single value is produced. We can also use the negative form ‘NOT IN’, which will evaluate to true if the record has a field value which is not in the set of values retrieved by the subquery.

#### Example: Nested Subqueries

**Question: Get an alphabetical list of names and IDs of all students in any class taught by F110.**

**Solution:** We need three tables, Student, Enroll, and Class, to answer this question. However, the values to be displayed appear on one table, Student, so we can use a subquery. First we check the Class table to find the classNumber of all courses taught byF110. We find two values, MTH101B and MTH103C. Next we go to the Enroll table to find the stuId of all students in either of these courses. We find three values, S1020, S1010, and S1002. We nowlook at the Studenttable to find the records with matching stuId values, and display the stuId, lastName, and firstName, in alphabetical order by name.

**Screenshots on MYSQL:**



## -Mihir Vora 60004220115



NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 6

Submission date : 30-11-2022

### Experiment No-6

**AIM:** Perform Join operations.

#### THEORY:

1. **UNION**

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of

columns. The columns must also have similar data types. Also, the columns in each SELECT

statement must be in the same order. SQL UNION Syntax

SELECT column\_name(s) FROM table1 UNION

SELECT column\_name(s) FROM table2;

Note: The UNION operator selects only distinct values by default. To allow duplicate values,

use the ALL keyword with UNION. SQL UNION ALL Syntax

SELECT column\_name(s) FROM table1 UNION ALL

SELECT column\_name(s) FROM table2;

#### INTERSECT

The SQL INTERSECT clause/operator is used to combine two SELECT statements, but

returns rows only from the first SELECT statement that are identical to a row in the second

SELECT statement. This means INTERSECT returns only common rows returned by the two

SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT

operator. MySQL does not support INTERSECT operator

#### EXCEPT

The SQL EXCEPT clause/operator is used to combine two SELECT statements and returns

rows from the first SELECT statement that are not returned by the second SELECT

statement. This means EXCEPT returns only rows, which are not available in second

SELECT statement.

Just as with the UNION operator, the same rules apply when using the EXCEPT operator.

MySQL does not support EXCEPT operator.

#### JOINS INNER JOIN

The INNER JOIN keyword selects all rows from both tables as long as there is a match

between the columns in both tables. SQL INNER JOIN Syntax

SELECT column\_name(s) FROM table1

INNER JOIN table2

ON table1.column\_name=table2.column\_name; LEFT JOIN

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows

in the right table (table2). The result is NULL in the right side when there is no match.

SQL LEFT JOIN Syntax

SELECT column\_name(s) FROM table1

LEFT JOIN table2

ON table1.column\_name=table2.column\_name;

#### RIGHT JOIN

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching

rows in the left table (table1). The result is NULL in the left side when there is no match.

SQL RIGHT JOIN Syntax

SELECT column\_name(s) FROM table1

RIGHT JOIN table2

ON table1.column\_name=table2.column\_name;

#### Examples-

1. Retrieve customer names having loan or account or both by eliminating duplicates

select cust\_name from borrower union

select cust\_name from depositor;

1. Retrieve customer names having loan or account or both without eliminating

duplicates

select cust\_name from borrower union all

select cust\_name from depositor;

1. List customers who have placed order. select cname, order\_number

from customer join order1 on order1.id=customer.customer\_id;

1. List all customers whether they placed order or not. select cname, order\_number

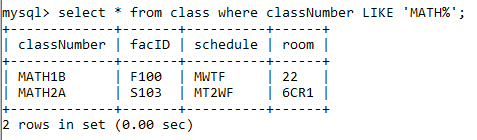
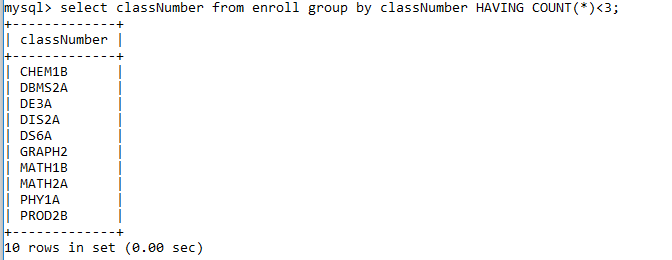
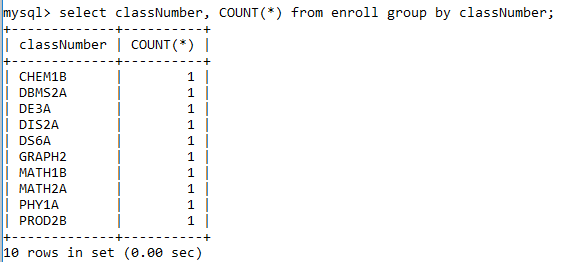
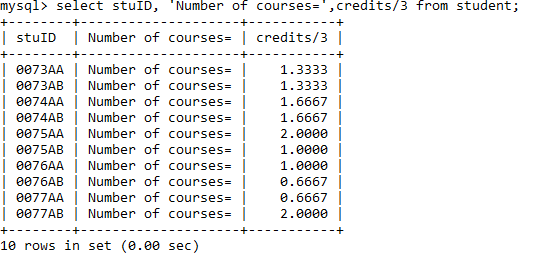
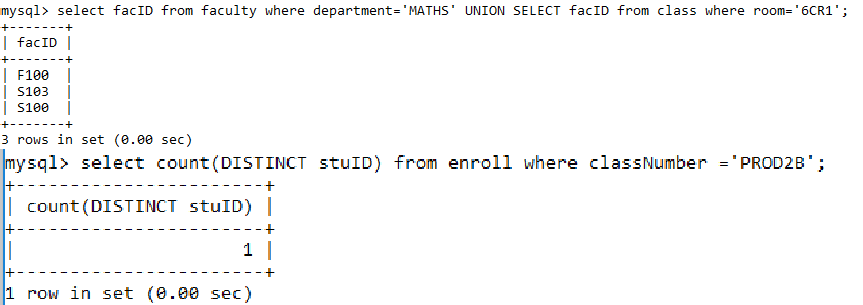
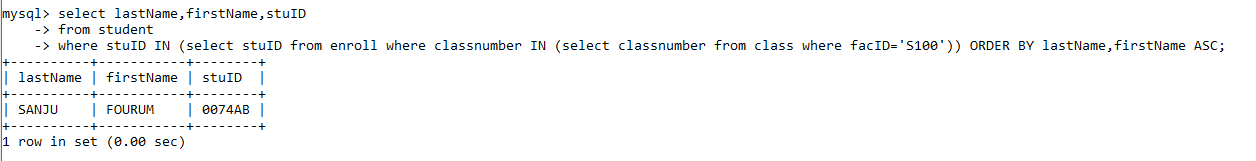
from customer left outer join order1 on order1.id=customer.customer\_id;

1. List all order numbers whether they are ordered by customers or not. select cname, order\_number

from customer right outer join order1 on order1.id=customer.customer\_id;

**Conclusion**: Various queries are executed with set operations and all types of joins.

**Screenshots on MYSQL:**



# -Mihir Vora 60004220115

NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 7

Submission date : 01-12-2022

### Experiment No-7

**Aim:** Create Views and Triggers.

**Theory:** A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view. Views, which are kind of virtual tables, allow users to do the following:

* + Structure data in a way that users or classes of users find natural or intuitive.
  + Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
  + Summarize data from various tables which can be used to generate reports.

Creating Views:

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

#### CREATE VIEW

**view\_name AS SELECT column1, column2.....**

**FROM table\_name WHERE [condition];**

1. **create a view to select employee name and address as well as the department details in which he is working group by department number.**

create view emp\_temp

as select ename, address, salary,

dno,dname from employee, department

where dno=dnumber group by dno;

1. Display all records of a view.

Select \* from emp\_temp;

1. Delete the view emp\_temp.

Drop view emp\_temp

#### Trigger:

A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table. Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

A trigger is a named database object that is associated with a table, and it activates when a particular event (e.g. an insert, update or delete) occurs for the table. The statement CREATE TRIGGER creates a new trigger in MySQL.

Syntax

CREATE

[DEFINER = { user | CURRENT\_USER }] TRIGGER

trigger\_name trigger\_time trigger\_event

ON tbl\_name FOR EACH ROW trigger\_body

trigger\_time: { BEFORE | AFTER } trigger\_event: { INSERT | UPDATE

| DELETE }

Example

* 1. delimiter //

create trigger depocheck before insert on depositor FOR EACH ROW

IF NEW.salary is null THEN

SET NEW.salary =

5000; END IF;

//

* 1. delimiter //

create trigger feed\_depositor\_bkp after insert on depositor

FOR EACH ROW

insert into depositor\_bkp(cust\_name,salary,branch) values (NEW.cust\_name, new.salary,new.branch);

//

* 1. delimiter //

create trigger employeetrig before update on employee for each row

if new.dno is null then set new.dno= 01;

end if;

//

* 1. delimiter //

create trigger total\_sal1 after insert on

employee for each row

if new.dno is not null then update department

set total\_sal=total\_sal+new. salary where dnumber=new.dno;

end if;

//

* 1. drop trigger feed\_depositor\_bkp;

#### CONCLUSION:

Hence, we have executed views and triggers for a college database.

# -Mihir Vora 60004220115

NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 8

Submission date : 01-12-2022

### Experiment No-8

**Aim:** Implementations locks in various modes for concurrency control.

**Theory:** A transaction is a logical unit of processing in a DBMS which entails one or more database access operation. In a nutshell, database transactions represent real-world events of any enterprise.

All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical database, would have a mix of reading and WRITE operations and hence the concurrency is a challenge.

Concurrency control is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases.

Therefore, concurrency control is a most important element for the proper functioning of a system where two or multiple database transactions that require access to the same data, are executed simultaneously.

#### Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.

* Lock-Based Protocols
* Two Phase
* Timestamp-Based Protocols
* Validation-Based Protocols

#### Concurrency control using Mysql Locks:

Deadlock is a state of a database system having two or more transactions, when each transaction is waiting for a data item that is being locked by some other transaction. A deadlock can be indicated by a cycle in the wait-for-graph. This is a directed graph in which the vertices denote transactions and the edges denote waits for data items.

A database locks serve to protect shared resources or objects. These protected resources could be:

* Tables
* Data Rows
* Data blocks
* Cached Items
* Connections
* Entire Systems

The deadlock detection and removal approach runs a deadlock detection algorithm periodically and removes deadlock in case there is one. It does not check for deadlock when a transaction places a request for a lock. When a transaction requests a lock, the lock manager checks whether it is available. If it is available, the transaction is allowed to lock the data item; otherwise the transaction is allowed to wait.

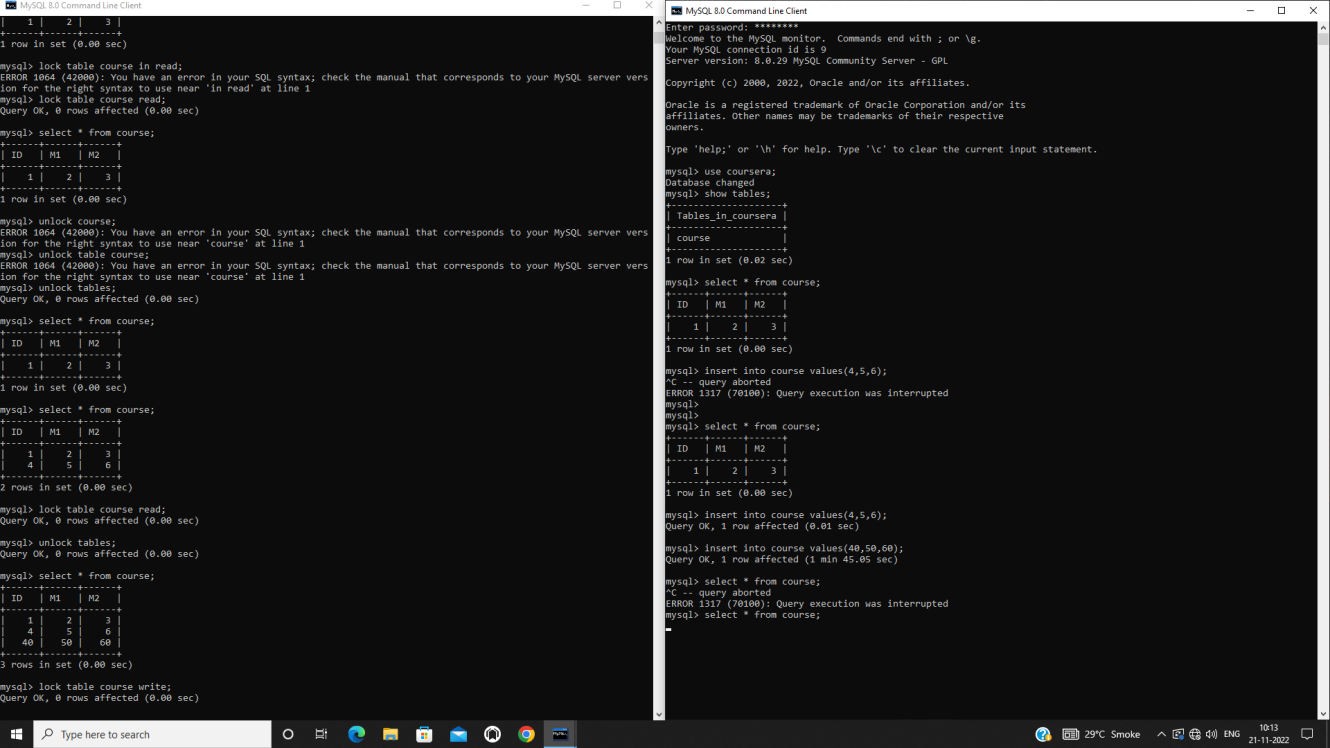
Since there are no precautions while granting lock requests, some of the transactions may be deadlocked. To detect deadlocks, the lock manager periodically checks if the wait-for graph has cycles.

If the system is deadlocked, the lock manager chooses a victim transaction from each cycle. The victim is aborted and rolled back; and then restarted later. Some of the methods used for victim selection are –

* Choose the youngest transaction.
* Choose the transaction with fewest data items.
* Choose the transaction that has performed least number of updates.
* Choose the transaction having least restart overhead.
* Choose the transaction which is common to two or more cycles.

This approach is primarily suited for systems having transactions low and where fast response to lock requests is needed.

#### Sreenshots on MYSQL:



***-Mihir Vora 60004220115***

NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 9

Submission date : 09-12-2022

### Experiment No-9

**Aim:** Case study on Fragmentation (PHf, DHF, Vf and Hf)

**Theory:** Networks of computers are found in everywhere. The Internet is one, as are the many networks of which it is composed. Mobile phone networks, corporate networks, factory networks, campus networks, home networks, in-car networks \_ all of these type of networks, both separately and in combination, share the essential characteristics that make them relevant subjects for study under the heading distributed systems.

Distributed systems are an important development in computing technology, which is concerned with the delivery of constantly expanding data to points of query. Collections of data in the forms of partitions or fragments can be distributed or replicated over multiple physical locations. When an organization is geographically dispersed, it may choose to store its databases on a central database server or to distribute them to local servers (or a combination of both).

A **distributed database** is a single logical database that is spread physically across computers in multiple locations that are connected by a data communications network. A distributed database (DDB) is a collection of data that logically belongs to the same system but is spread over the sites of a computer network. Distribution of data is a collection of fragmentation, replication and allocation processes. The sites of the distributed database may be allocated in the same space and have the same network address but the communication between them is done over a network instead of sharing memory.

As communication technology, software and hardware, advance rapidly and the equipment prices falls every day, developing DDBS become more and more feasible. Design of efficient distributed database is one of the hot topics in database and information technology areas

**Fragmentations**

Distributed database system design primary concern is to make fragmentation of classes in case of object-oriented databases, or the relations in case of relational database, replication and allocation of the fragments in different sites of the distributed system, and local optimization in each site. Fragmentation is a technique to split a single

class or relation of a database into two or more partitions, also; the combination of the partitions supports the original database without any loss of information. That means, it reduces the amount of irrelevant data accessed by the applications of the database, thus reducing the number of disk accesses. Each fragment may be stored at any site over a computer network.

Fragmentation aims to improve:

1. Reliability.
2. Performance.
3. Balanced storage capacity and costs.
4. Communication costs.
5. Security.

The following information is used to decide fragmentation: Quantitative information: cardinality of relations, frequency of queries, site where query is run, selectivity of the queries, etc.

Qualitative information: predicates in queries, types of access of data, read/write, etc.

Fragmentation Types

Fragmentation can be horizontal, vertical or mixed.

##### 1. Horizontal Fragmentation

Horizontal fragmentation (HF) allows a relation or class to be partitioned into disjoint tuples or instances. Each fragment is stored at a different node, and each fragment has unique rows. However, the unique rows all have the same attributes (columns). Intuition behind horizontal fragmentation is that Every site should hold all information that is used to query at the site and the information at the site should be fragmented so the queries of the site run faster.

***2. Vertical Fragmentation(VF)***

A class or relation in VF will be partitioned into separate sets of columns or attributes except the primary key. Each set must include the primary key attribute(s) of the table. This arrangement can make sense when different sites are responsible for processing different functions involving an entity.

The main target of vertical fragmentation is to divide a relation into a set of smaller relations, so that in only one fragment many of the applications will run.

##### 3. Mixed Fragmentation

Combination of Horizontal and vertical fragmentations give the mixed fragmentations (MF). The table in this type of fragmentation scheme is divided into blocks that based on the needed requirements. Each fragmentation may be allocated on a specific site. Mixed fragmentation is the most complex one; it needs more management. Mixed fragmentation contains a vertical fragment followed schema will not be sufficient to satisfy the requirements by a vertical fragmentation, or a horizontal fragmentation followed by a vertical fragmentation.

**Correctness Rules of Fragmentation**

1. Completeness

Decomposition of relation R into fragments R1,R2,…,Rn is complete

iff each data item in R can also be found in some Ri.

1. Reconstruction

If relation R is decomposed into fragments R1, R2,…, Rn, then there

should exist some relational operator ∇that reconstructs R from its fragments, i.e., R = R1∇…∇Rn:

Union to combines horizontal fragments.

Join to combine vertical fragments.

1. Disjointness

If relation R is decomposed into fragments R1, R2,…, Rn and data item di appears in fragment Rj, then di should not appear in any other fragment Rk, k = j (exception: primary key attribute for vertical fragmentation

for horizontal fragmentation, data item is a tuple.

for vertical fragmentation, data item is an attribute.

For each fragment of a relation R, the comparison of fragmentation strategies is shown.

1. Condition C = True (all tuples are selected).
2. List (L = ATTRS(R)) = True (all attributes are included in the list).

# -Mihir Vora 60004220115

NAME : Mihir Ashish Vora

SAP ID : 60004220115

TOPIC : DBMS PRACTICAL 10

Submission date : 08-12-2022

### Experiment No-10

**Aim:** Case study on Recent Databases and applications.

**Theory:** Historically, Database Management systems (DBMS) were simple software programs and associated hardware that allowed users to access data from different geographical locations. The system offers its users the ability to store data without concerns about structural changes, or the data’s physical location. Additionally, a Database Management system (DBMS) can set restrictions on the data being used, and the services available to each user.

DBMSs are changing, however. They are expanding, taking on more responsibilities, and providing smarter answers. As new goals and problems present themselves, the desire to find new ways to use Database Management systems prompt unique solutions. Many of these innovations are available only in cloud-based DBMSs. As Database Management systems develop new features and new options, it makes sense to re- examine the organization’s current system, and consider all new options.

The coronavirus pandemic, with its emphasis on isolation, has accelerated the acceptance of online shopping and working remotely. Many small businesses have made the decision to digitize and are shifting to the

cloud at an accelerated rate.

The market for Database Management systems is growing fast and, according to Research and Markets, the global DBMS market was estimated to have reached $63.9 trillion in 2020, and is projected to reach

$142.7 trillion by 2027.

Increasingly, organizations are merging their data warehouses and data lakes into cloud storage systems. Shifting to the cloud requires a Database Management system (DBMS) for working with a broad range of new data formats.

Database Management trends in 2022 include:

* + Cloud-based DBMS
  + Automation and DBMS
  + Augmented DBMS
  + Increased security
  + In-memory databases
  + Graph databases
  + Open source DBMSs
  + Databases-as-a-service

These trends are based, to a large extent, on businesses wanting to provide access to their products and services over the internet, with the goal of maintaining (or increasing) profits during the pandemic.

#### Netflix Database Study:

In 2019 Netflix began to run into problems with Cassandra for certain use cases. At that time Netflix was seeing an increase in demand and an increase in their database requirements. For example, the studio side of Netflix was producing more shows and movies which added new data needs. Generally speaking, there was an increase in the need for consistent data, specifically for these three use cases:

* Cloud drive service - a file system-like service for media assets which was needed by the Netflix studio side of the business.
* Content delivery - Netflix built its own CDN called Open Connect, and they needed a control plane service to manage network devices around the world.
* Spinnaker - a continuous delivery platform on the cloud.

These are all global services that need to support consistent transactions at times. Supporting consistent transactions is problematic with Cassandra, because in Cassandra you don’t get rich transactions. Instead, you have lightweight transactions, which are extremely performant, but limited. Also, the secondary indices in Cassandra are unreliable and often don’t work.

The other options Netflix considered for these use cases were AWS Aurora as well as DynamoDB but they ran into some limitations with scalability there.

Netflix decided that they needed a scalable SQL database for those three use cases and established a set of requirements: Multi-active topology, Global consistent secondary indices, global transactions, open source, and SQL.

In 2020 Netflix deployed their first CockroachDB cluster in production. Today they have 100 production clusters and 150+ test clusters. At this time most of the clusters are deployed in a single region with three availability zones. The biggest CockroachDB cluster at Netflix is a 60-node, single-region cluster with 26.5 terabytes of data.

In this section of the presentation, Shengwei Wang explains why Netflix does not use the CockroachDB binary, how Netflix navigates upgrades to new versions of CockroachDB, and how Netflix deploys/manages CockroachDB.

**Conclusion:** Hence it is evident from the above case study the importance of Database Management Systems in the modern world along with the developments that have taken place in the recent years.

# -Mihir Vora 60004220115

***THANK YOU***