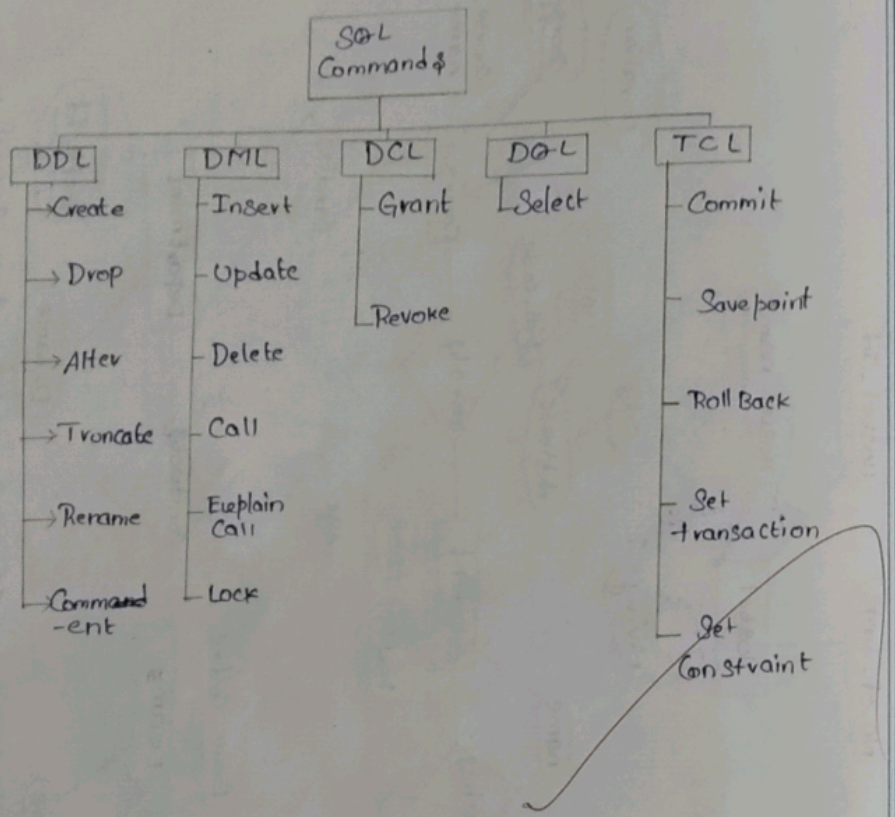
SQL Commands:



TASK-2.1

Date: 4-8-2025

Study of SQL Commands

SQL Commands:

SQL Commands are the fundamental building blocks for Communicating with a database management System (DBMS).

- SQL Commands are mainly Categorized into five Categories:
- DDL - Data Definition language
- DQL - Data Query language
- DML - Data Manipulation language
- DCL - Data Control language
- TCL - Transaction Control language.

DDL : Data Definition Language:

DDL actualy Consists of the SQL Commands that Can be used for defining, altering and deleting database structures.

(1) Create: It Create database (or) its objects (table, index, function, views, store procedure and triggers).

Syntax: CREATE TABLE table_name (Column1 data-type, Column2 data-type, ...);

Example: ~~CREATE TABLE Students~~

CREATE TABLE employees (id INT, name VARCHAR(50), age INT, Salary DECIMAL (10,2));

(2) Drop: Employed to remove objects from a database. Deleting tables, views, indexes, (or) other database Structures when they are no longer needed.

Syntax:- DROP TABLE table_name;

Example:- DROP TABLE employees;

③ ALTER:- used to modify the design (or) structure of an existing data base. changing attributes of database elements like altering table schemas (or) modifying column properties.

Syntax:- ALTER TABLE table_name ACTION;

Example:- ALTER TABLE employees ADD email VARCHAR(100);

④ Truncate:- used to delete all data inside a table without deleting the table itself.

Syntax:- 'TRUNCATE TABLE table_name;

Example:- 'TRUNCATE TABLE employees';

⑤ RENAME:- used to remove an existing table.

Syntax:- 'ALTER TABLE table_name RENAME TO new_table_name;

Example:- 'ALTER TABLE employee RENAME TO staff;

⑥ CREATE:- used to create an index on one (or) more columns.

Syntax:- 'CREATE INDEX index_name ON table_name(column1, column2, ...);

Example:- 'CREATE INDEX idx_name ON employee (name);

2. DATA MANIPULATION LANGUAGE:- It focus on modifying (or) handling data within a database fall under the category a significant portion of SQL operations.

1. INSERT:- Used to add new rows (records) to a table.

Syntax:-

INSERT INTO table_name(column1, column2.) values (value1, value2)

Example:-

INSERT INTO employees (name, department) VALUES ('John Doe', 'Marketing');

2. UPDATE:- Modifies existing data within a table. It can update one (or) more columns for all rows that meet the condition specified.

Syntax:-

UPDATE table_name SET column1 = value1, column2 = value 2 WHERE condition;

Example:-

UPDATE employees SET department = 'HR' WHERE id = 123;

3. DELETE:- Remove one (or) more rows from a table. without a specific condition, it will delete all rows in the table.

Syntax:-

DELETE FROM table_name WHERE condition;

Example:-

DELETE FROM employees WHERE department = 'Intern';

4. Call:- Call a PL/SQL (or) JAVA subprogram

Syntax:- Call procedure_name(arguments);

Example:-

5. Explain plan:- Describes the access path to data

Syntax:- ~~Explain~~
EXPLAIN PLAN FOR SELECT * FROM table_name.

Example:-

6. Lock:- Table Control Currency.

Syntax:- LOCK TABLE table_name IN lock_mode;

Example:-

DCL:- Data Control language.
DCL includes commands such as GRANT and REVOKE which mainly deals with the rights permissions and other Controls of the database system.

GRANT:- Assigns new privileges to a user account, allowing access to specific database objects, actions (or) functions.

Syntax:- GRANT privilege_type [(column_list)] ON [object_type] object_name TO user [WITH GRANT OPTION];

Example:- GRANT SELECT, INSERT ON students TO user1;

REVOKE:- Removes previously granted privileges from a user account, taking away their access to certain database objects (or) actions.

Syntax:- REVOKE [GRANT OPTION FOR] privilege_type [(column_list)] ON [object_type] object_name FROM user [CASCADE];

TCL: Transaction Control language.
Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed.

Commit:- Saves all changes made during the transaction.

Syntax:- COMMIT;

Example:- BEING TRANSACTION;
INSERT INTO students (ID, Name, Age) VALUES (3,
COMMIT;

Savepoint:- Creates a savepoint within the current transaction.

Syntax:- ~~Sa~~ SAVEPOINT savepoint_name;

Example:- SAVEPOINT SP1;
UPDATE students SET Age = 25 WHERE ID = 3;
ROLLBACK TO SP1;

ROLLBACK:- Undoes all # changes made during the transaction

Syntax:- ROLLBACK;

Example:- BEGIN TRANSACTION; SET Age = 30;
INSERT INTO students ~~(ID, Name~~
ROLL BACK;

SET TRANSACTION:- It defines the transaction's parameters.

Syntax:-
SET TRANSACTION [transaction_characteristics];

Example:-
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

(5) DQL:- DATA QUERY LANGUAGE:

DQL statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it.

SELECT:- It is used to retrieve data from the database.

Syntax:- SELECT Column1, Column2, ... FROM table_name WHERE condition;

Example:

GroupBy:- It is used with aggregate functions to group the result according to one (or) more columns;

Syntax:- SELECT Column1, aggregate_function (Column2) FROM table_name GROUP BY Column1;

Example:- SELECT region SUM (amount)
FROM Sales
GROUP BY region;

ORDER By:- It is used to sort the results in accending (or) descending order.

Syntax:- SELECT Column1, Column2 FROM table_name ORDER BY Column1 [ASC|DESC], Column2 [ASC|DESC];

Example:- SELECT first_name, last_name
FROM Students
ORDER BY age Asc, last_name DESC;

HAVING:- Filters the result of GROUP BY

Syntax:- SELECT Column1, AGE-FUNCTION (Column2)
FROM table_name
GROUP BY Column1
HAVING Condition;

DISTINCT:-
Removes dupilcate rows from the result set.

Syntax:- SELECT DISTINCT Column1, Column2, ...
FROM table_name;

Example:- SELECT DISTINCT

LIMIT:- By default, it sorts in ascending order unless specified as DESC

Syntax:- SELECT * FROM table_name LIMIT number;

Example:- SELECT * FROM

TASK-2.2 Design And Implementation Hierarchical/ Network Model:-

(1) Identify the specificity of each relationship in university management System.

Ans:
1. Student - Enroll - Courses — Many to Many
2. Department - offers - Courses — one to Many
3. Professor - Teaches - course — one to Many
4. Course - Has - schedule — one to one
5. Student - Makes - Payment — one to Many
6. Student - Allocated to - Hostel Room — one to one
7. Student - Has - Attendence — one-to-Many

(2) Find the domain of attributes and apply suitable Constraints.

* Student Entity:-

Attribute	Domain	Constraints
1. Student_id	VARCHAR(10)	Primary Key, NOT NULL, Unique
2. first_name	VARCHAR(50)	NOT NULL
3. Last_name	VARCHAR(50)	NOT NULL
4. email	VARCHAR(100)	Unique, Valid, Not NULL
5. ~~dead~~ dob	DATE	NOT NULL, Must be >age 16
6. depa_id	VARCHAR(10)	Foreign key to Department (depa_id)

* Professor Entity:-

Attribute	Domain	Constraints
Prof_id	VARCHAR(10)	Primary Key, Not Null, Unique.
first_name	VARCHAR(50)	Not NULL
last_name	VARCHAR(50)	Not NULL
email	VARCHAR(100)	Unique, Not Null
depa_id	VARCHAR(10)	Foreign key to Department (dept_id)

* Enrollment:-

Attribute	Domain	Constraints
enroll_id	INT	Primary Key
Student_id	VARCHAR(10)	Foreign key to Student (student_id)
Course_id	VARCHAR(10)	Foreign key to Course (Course_id)
enroll_date	DATE	NOT NULL

* Payment Entity:-

Attribute	Domain	Constraints
Payment_id	INT	Primary key
Student_id	VARCHAR(10)	Foreign key to student (student_id)
amount	DECIMAL(10,2)	Not Null, Valuog

Payment_date	DATE	Valid, NOT NULL

* Department Entity:-

Attribute	Domain	Constraints.
Department_ID	INT	Primary key.
Dept_Name	VARCHAR (100)	unique, NOT NULL
HOD	INT	Forigen key, → (Faculty (Faculty_ID))

* Schedule Entity:-

Attribute	Domain	Constraints.
Schedule_ID	INT	Primary Key
Course_ID	INT	Forigen key → Course(Course_ID)
Faculty_ID	INT	Forigen key → Faculty (Faculty_ID)

* Attendence Entity:-

Attributes	Domain	Constraints.
Attendence_ID	INT	Primary key
Student_ID	INT	Forigen key → Cstudent (student_ID)
course_ID	INT	Forigen key → Ccourse (Course_ID)
Status	VARCHAR (10)	check (status IN ('Present', 'Absent')).

* Hostel Entity:-

Attributes	Domain	Constraints.
Hostel_Name	VARCHAR(100)	unique, NOT NULL
Hostel_ID	INT	Primary key
Warden_Name	VARCHAR (100)	NOT NULL ~~unique~~ .
Location	VARCHAR (200)	NOT NULL

* Perform SQL operations using DDL And DML Commands:-

DDL Commands:-

CREATE Student (student_ID INT Primary key, First_name char (100) NOT NULL, LAST_Name char (100) NOT NULL, email varchar(100) NOT NULL valid);

CREATE TABLE Faculty (Prof_id VARCHAR (100) primary key, first_name char (100) NOT NULL, last_name char (100) NOT NULL, email char (100) valid NOT NULL).

CREATE TABLE Department (Department_ID INT Primary key, Dept_Name VARCHAR (100) unique NOT NULL, HOD INT Forigen Key → (Faculty (Faculty_ID)));

CREATE TABLE Enrollment ~~Courses~~ (enroll_Id INT Primary key, Student_Id INT Forigen key to Student (Student_ID), Course_ID INT Forigen key to Coorse (Coorse_ID));

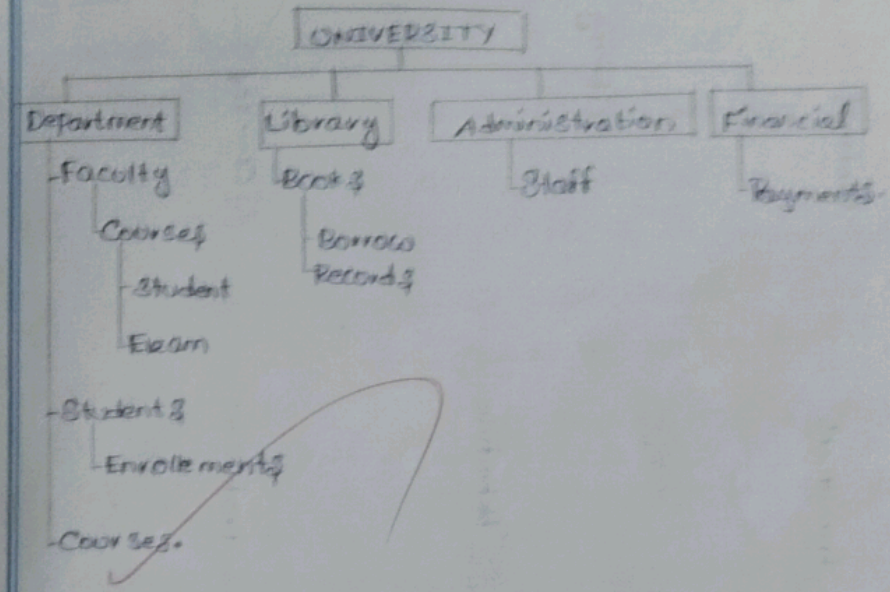
```
CREATE TABLE Department (Dept_Name char(100) Unique, NOT
NULL, Dept_ID INT unique, HOD INT Foriger Key to
Faculty(Faculty_ID))

CREATE TABLE Schedule (schedule_ID INT Unique,
Course_ID INT Foriger key to Course(Course_ID), Faculty_ID
INT Foriger key to Faculty(Faculty_ID));

CREATE TABLE Hostel (Hostel_ID INT unique, Primary key,
HostelName VARCHAR(100) unique NOT NULL, Warden_Name
VARCHAR(100) NOT NULL, Location VARCHAR(100) NOT NULL);

ALTER TABLE Student ADD Blood_Group VARCHAR(5);

ALTER TABLE Hostel DROP Column Contact_NO;

DROP ATABLE Attendence;

TRUNCATE TABLE Hostel; ~~Allocate~~

ALTER TABLE Course RENAME TO Course_MASTER;
```

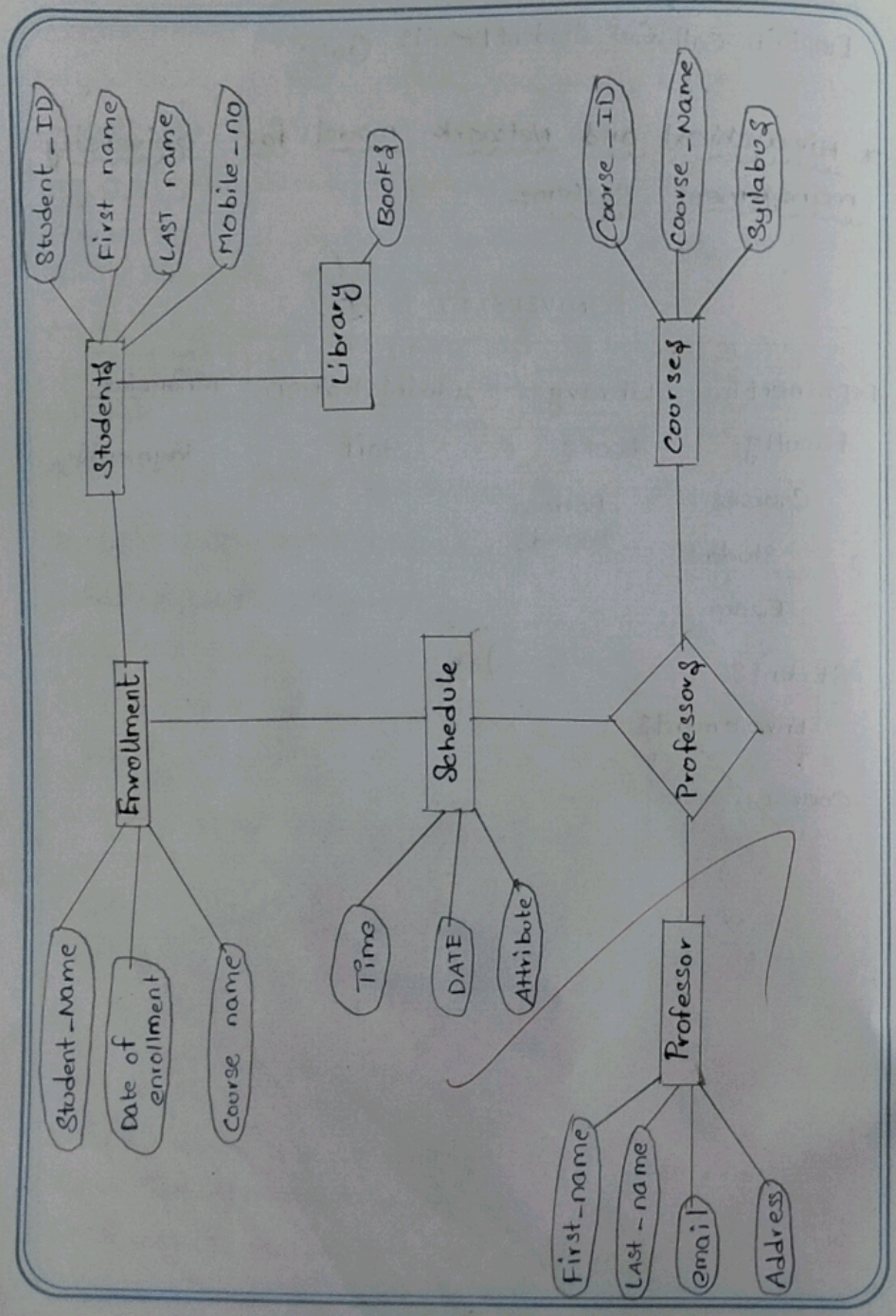
DML Commands:-

```
INSERT INTO Department Values (1,'Computer Science',
NULL);

INSERT INTO Faculty values (100, 'Dr. Kumar', 'Professor',
Kumar@uni.edu', '9998887771', 1);

Update Fee
SET Status = 'Paid'
WHERE Fee_ID = 6001;

DELETE FROM Student WHERE Student_ID = 100;

CALL Attendence(101);

LOCK TABLE Student IN EXCLUSIVE MODE;
```

Explain Call Get Student Details (101);

* Hierarchical and Network model for university management Systems:-





TASK 2.3 - Different types of Select Queries:

(1) Basic SELECT:- Fetches all columns and rows from the student table.

```
SELECT * FROM Students;
```

(2) Select specific columns:- Fetches only selected Columns.

```
SELECT Stu_name, Stu_age, FROM Students;
```

(3) WHERE Clause (Filtering data):- Fetches records of Students older than 20.

```
SELECT * FROM Students WHERE Stu_age >20;
```

(4) DISTINCT:- Fetches only unique Course IDs (remove duplica-te).

```
SELECT DISTINCT Course_id FROM Student_Course;
```

(5) ORDER BY (Sorting):- Sorts Students by age in descending Order.

```
SELECT Stu_name, Stu_age FROM Students ORDER BY Stu_age DESC;
```

(6) LIMIT (TOP N Results):- Returns only 5 rows.

```
SELECT * FROM Students LIMIT 5;
```

(7) LIKE (Pattern Matching):- Finds Students whose names start with "A".

```
SELECT * FROM Students WHERE Stu_name LIKE 'A%';
```

(8) BETWEEN:- Fetches Students whose age is between 18 and 22.

```
SELECT * FROM Students WHERE Stu_age BETWEEN 18 AND 22.
```

(9) IN operator:- Fetches Students aged 18, 20 (or) 22.

```
SELECT * FROM Students WHERE Stu_age IN (18,20,22);
```

(10) Aggregate Functions:- Returns total number, average age, maximum and minimum.

```
SELECT COUNT(*) As total_Students FROM Students;
SELECT AVG (Stu_age) As average_age FROM Students;
SELECT MAX (Stu_age), MIN (Stu_age) FROM Students;
```

(11) GROUP BY :- Groups students by course and counts them.

```
SELECT Course_id, COUNT(Sto_id) AS total_students
FROM Student_Course
GROUP BY Course_id;
```

(12) HAVING (with Groups) :- Fetches only courses where more than 2 students enrolled.

```
SELECT Course_id, COUNT(Stuid) AS total_students
FROM Student_Course
GROUP BY Course_id
HAVING COUNT(Stuid)>2;
```

(13) JOIN queries :-

→ INNER JOIN :- Shows student names with their course names. SELECT S. Sto_name, C. name.

```
SELECT FROM Students
INNER JOIN Student_Course SC ON S.Sto_id = SC.Stuid.
INNER JOIN Course C ON SC.Course_id = C.Course_id;
```

→ LEFT JOIN :- Shows all students (even if they didn't take a course).

```
SELECT S.Sto_name, Sto_age.
FROM Students
WHERE Sto_age> (SELECT AVG (Sto_age) FROM Students);
```

(14) Sub query (Nested SELECT) :- Finds student older than the average age.

```
SELECT Sto_name, Sto_age
FROM Students
WHERE Sto_age> (SELECT AVG (Sto_age) FROM Students);
```

(15) UNION (Combine Results) :- Combines names of students and course names (unique values only).

```
SELECT Sto_name FROM Students
UNION
SELECT name FROM Course;
```

VEL TECH - OS Lab	
EX NO.	2
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	-
TOTAL (20)	15
SIGN WITH DATE	15/9/25

RESULT :- Thus, the study of SQL Commands is done successfully.

15/9/25