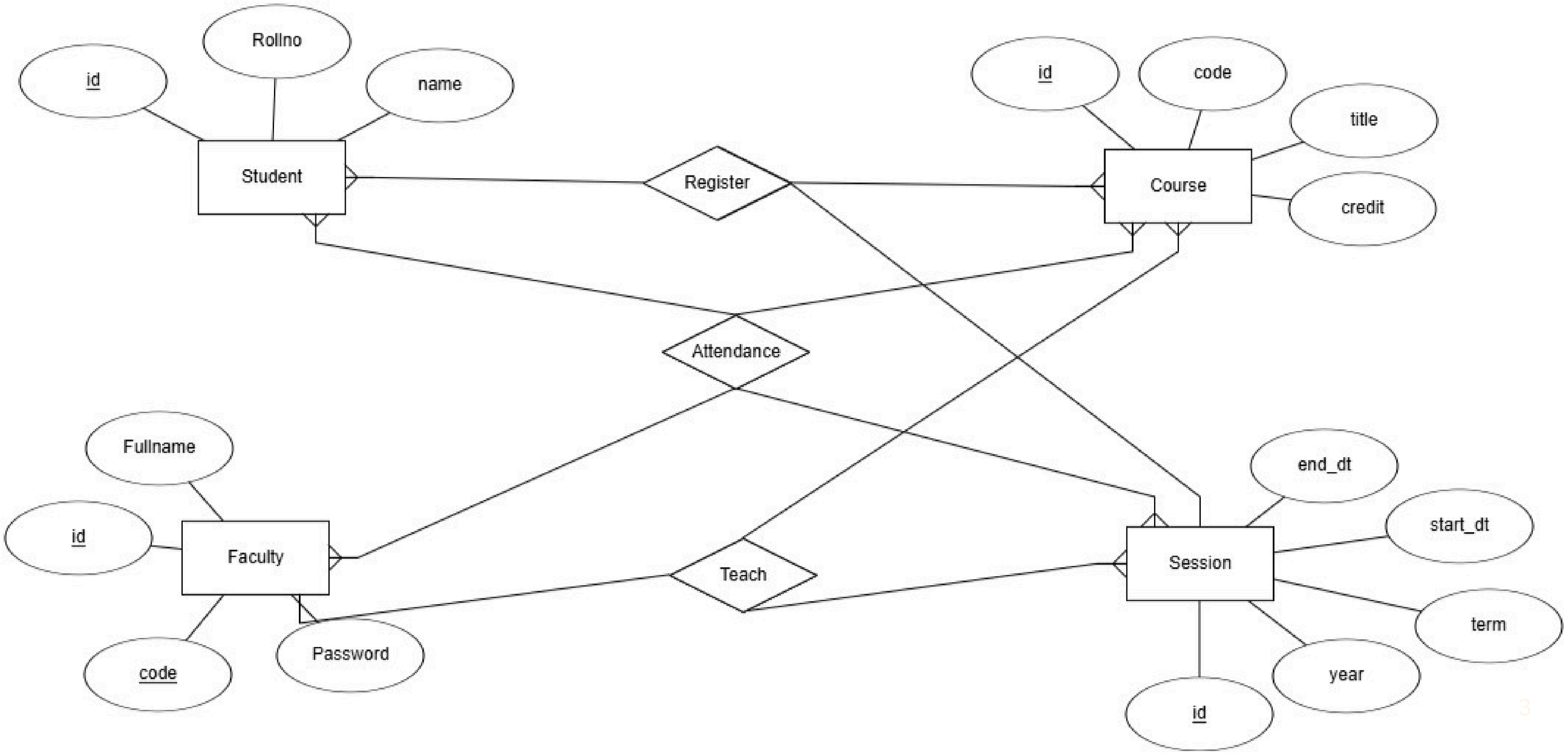
STUDENT ATTENDANCE MANAGEMENT SYSTEM

Presented by:
Laxman Patil-123B1D032
Yadnesh Patil-123B1D034
Abhay Bormale-123B1D057

Project Name: Student Attendance Management System

Objective:-To create a database system for managing student attendance records for a school or college.

ER DIAGRAM:


Rollno
id
name
Student
Register
id
code
title
credit
Course
Attendance
Fullname
id
Faculty
code
Password
Teach
end_dt
start_dt
Session
term
year
id

• Relation Tables From ER-Diagram

- **Retrieve all students:-**

```
196    -- Retrieve all students
197 •  SELECT * FROM student_details;
198
```

Result Grid | Filter Rows: | Edit:

id	roll_no	name
1	123B1D001	Devraj Bafna
2	123B1D002	Adwait Kamble
3	123B1D003	Vardhan Khinvasara
4	123B1D004	Kaiwalya Ladkhedkar
5	123B1D005	Anuj Ahire
6	123B1D006	Samir Babar

- **Retrieve all faculty members:-**

```
199    -- Retrieve all faculty members
200 •  SELECT * FROM faculty_details;
201
202    -- Retrieve all courses
```

Result Grid | Filter Rows: | Edit:

id	user_name	name	password
1	GD	Ganesh Deshmukh	123
2	MS	Minal Shahakar	123
3	SP	Sandeep Patil	123

- **Retrieve all courses:-**

```
202    -- Retrieve all courses
203 •  SELECT * FROM course_details;
204
205    -- Retrieve all sessions
```

Result Grid | Filter Rows: | Edit:

id	code	title	credit
1	CO321	Database Management System	2
2	CO215	Advanced Data Structure	3
3	CS112	Constitution Of India	4
4	CS670	Computational Techniques	4
5	CO432	Business Finance	3
6	CS673	Microprocessor Architecture	1
NULL	NULL	NULL	NULL

- **Retrieve all session:-**

```
205    -- Retrieve all sessions
206 •  SELECT * FROM session_details
207
208    -- Retrieve students register
```

Result Grid | Filter Rows: | Edit:

id	year	term
1	2025	EVEN SEMESTER
2	2025	ODD SEMESTER
NULL	NULL	NULL

- **Attendence Table:-**

```
248    -- Get all students with names starting with "A"
249 •  SELECT * FROM student_details WHERE name LIKE 'A%';
250
```

Result Grid | Filter Rows: | Edit: | Export/Imp

id	roll_no	name
2	123B1D002	Adwait Kamble
5	123B1D005	Anuj Ahire
11	123B1D011	Aarush Borkar
13	123B1D013	Aniket Damedhar
20	123B1D020	Amrita Iyer
26	123B1D026	Anushka Misal
27	123B1D027	Advait Nathe
29	123B1D029	Ajim Pathan
39	123B1D039	Atharv Sakhare
53	123B1D053	Adhya Bhagat
54	123B1D054	Abhay Bormale
56	123B1D056	Atharva Desai
63	123B1D064	Atharva Zope
NULL	NULL	NULL

- **Find courses with more than 3 credits:-**

```
245    -- Find courses with more than 3 credits
246 •  SELECT * FROM course_details WHERE credit > 3;
247
```

Result Grid | Filter Rows: | Edit: | Export/I

id	code	title	credit
3	CS112	Constitution Of India	4
4	CS670	Computational Techniques	4
NULL	NULL	NULL	NULL

- **List of all faculty names in alphabetical order:-**

```
251    -- List all faculty names in alphabetical order
252 •  SELECT * FROM faculty_details ORDER BY name ASC;
253
```

Result Grid | Filter Rows: | Edit:

id	user_name	name	password
4	AV	Ashwini Vaze	123
5	BD	Brijesh Deshmukh	123
1	GD	Ganesh Deshmukh	123
2	MS	Minal Shahakar	123
6	RS	Rucha Shinde	123
3	SP	Sandeep Patil	123
NULL	NULL	NULL	NULL

- **Retrieve student details sorted by roll number:-**

```
254    -- Retrieve student details sorted by roll number
255 •  SELECT * FROM student_details ORDER BY roll_no ASC;
256
```

Result Grid | Filter Rows: | Edit:

id	roll_no	name
1	123B1D001	Devraj Bafna
2	123B1D002	Adwait Kamble
3	123B1D003	Vardhan Khinvasara
4	123B1D004	Kaiwalya Ladkhedkar
5	123B1D005	Anuj Ahire
6	123B1D006	Samir Babar
7	123B1D007	Prasad Bange
8	123B1D008	Shreya Bhagat
9	123B1D009	Vishwajeet Bhamre
10	123B1D010	Prathamesh Bhangari
11	123B1D011	Aarush Borkar
12	123B1D012	Shreyash Chaudhari
13	123B1D013	Aniket Damedhar
14	123B1D014	Vijay Dhame
15	123B1D015	Sushant Didwagh
16	123B1D016	Tanaya Gaikwad
17	123B1D017	Virendra Gaikwad

- **Get session details sorted by year and term:-**

```
257     -- Get session details sorted by year and term
258 •  SELECT * FROM session_details ORDER BY year DESC, term ASC;
259
260     -- Count the total number of students
```

Result Grid | Filter Rows: | Edit: | Export/Import:

id	year	term
1	2025	EVEN SEMESTER
2	2025	ODD SEMESTER
NULL	NULL	NULL

- **Get attendance details for a specific student:-**

```
283     -- Get attendance details for a specific student
284 •  SELECT * FROM attendance_details WHERE student_id = 1;
285
```

Result Grid | Filter Rows: | Edit: | Export/Import:

faculty_id	course_id	session_id	student_id	on_date	status
NULL	NULL	NULL	NULL	NULL	NULL

Normalization of College Management Database to Third Normal Form (3NF)

"Improving Data Consistency, Reducing Redundancy, and Enhancing Database Efficiency"

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves decomposing larger tables into smaller, manageable ones while maintaining relationships

Normal Form	Rule
1NF	Atomic values only (no repeating groups or arrays)
2NF	1NF + No Partial Dependencies (non-key attributes depend on whole primary key)
3NF	2NF + No Transitive Dependencies (non-key attributes depend only on primary key)

Tables in Original Schema

- student_details
- faculty_details
- course_details
- session_details
- course_registration
- course_allotment
- attendance_details

1NF – Ensuring Atomic Values

- **All table fields contain atomic, indivisible values. No repeating groups or multi-valued attributes**

Example: In student_details, name is a single string, not a full name split across fields or stored with multiple values.

2NF – Removing Partial Dependencies

- **Composite keys used in junction tables.**

All non-key attributes fully depend on the entire primary key.

Example:

- course_registration uses a composite primary key (student_id, course_id, session_id), and no non-key column depends on just part of that key — all non-key fields (none here) would depend on the full composite key.

- Data like student name, course title, session year/term are not duplicated here but rather referenced by foreign keys — avoiding partial dependencies.

- **Before 2NF**

```sql
CREATE TABLE course_registration (
    student_id INT,
    course_id INT,
    session_id INT,
    student_name VARCHAR(50),  -- ✕ This is bad design!
    PRIMARY KEY (student_id, course_id, session_id)
```

- **After Modify to 2NF**

```sql
-- Course Registration
CREATE TABLE IF NOT EXISTS course_registration (
    student_id INT,
    course_id INT,
    session_id INT,
    PRIMARY KEY (student_id, course_id, session_id),
    FOREIGN KEY (student_id) REFERENCES student_details(id),
    FOREIGN KEY (course_id) REFERENCES course_details(id),
    FOREIGN KEY (session_id) REFERENCES session_details(id)
);
```

3NF – Eliminating Transitive Dependencies

- Non-key attributes depend only on primary keys.

No indirect relationships between non-key attributes.

Example:

- In faculty_details, login_id refers to login_credentials(id), and the user_name/password info is kept in a separate table, avoiding duplication or indirect dependency.
- attendance_details doesn't store student name, course title, or faculty name — it references them using IDs, thus eliminating transitive dependencies.

Changes made during normalization

- Confirmed atomicity in all columns.
- Validated functional dependencies.
- Ensured clean table relationships using junction tables.

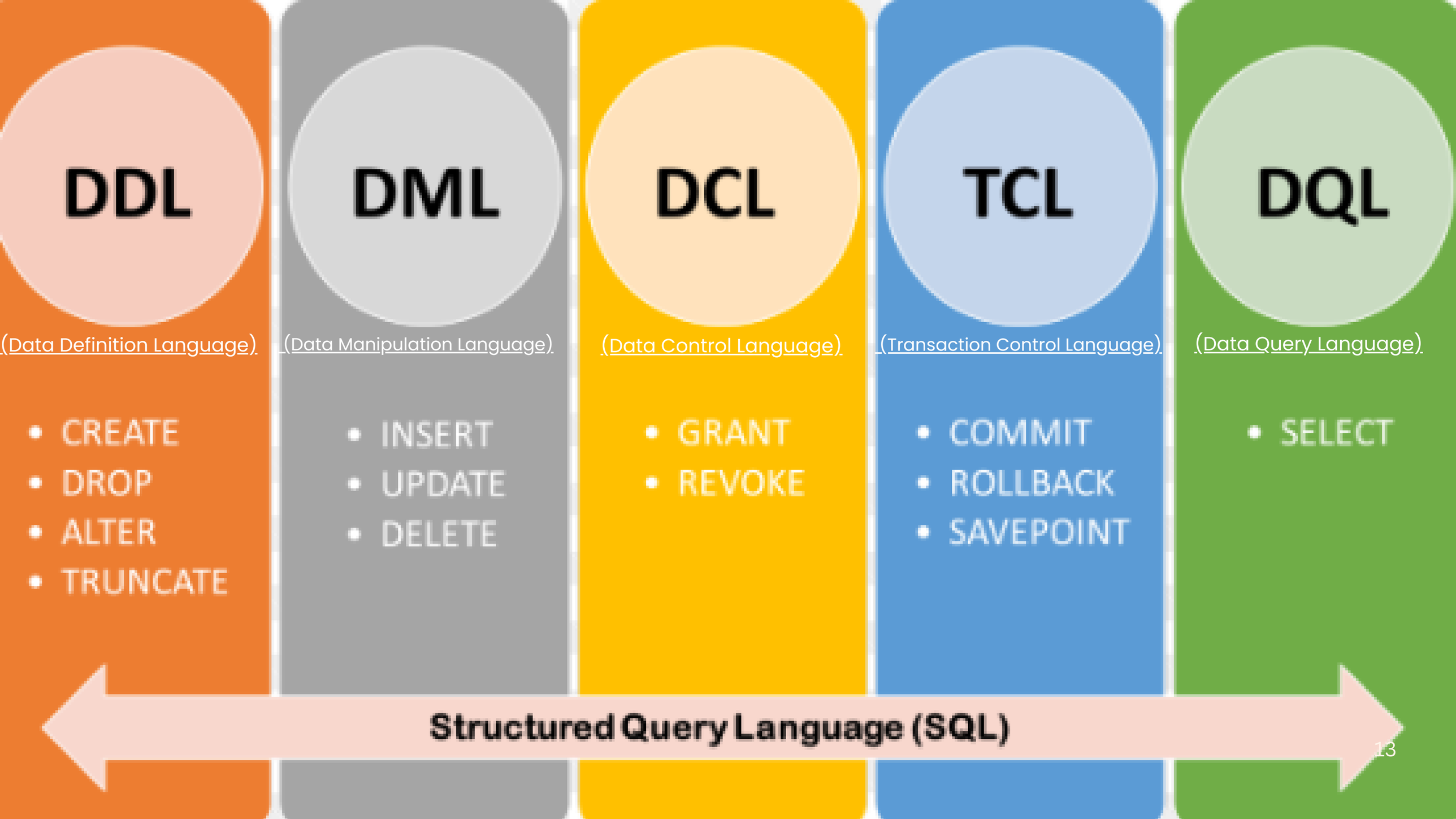
- **BEFORE 3RD NF**

```sql
CREATE TABLE faculty_details (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    user_name VARCHAR(20),       -- ✗ Shouldn't be here
    password VARCHAR(50)         -- ✗ Transitive dependency!
);
```

- **AFTER 3RD NF**

```sql
-- Login Credentials
CREATE TABLE IF NOT EXISTS login_credentials (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_name VARCHAR(20) UNIQUE,
    password VARCHAR(50)
);

-- Faculty Details
CREATE TABLE IF NOT EXISTS faculty_details (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    login_id INT,
    FOREIGN KEY (login_id) REFERENCES login_credentials(id)
);
```


DDL
(Data Definition Language)
• CREATE
• DROP
• ALTER
• TRUNCATE
DML
(Data Manipulation Language)
• INSERT
• UPDATE
• DELETE
DCL
(Data Control Language)
• GRANT
• REVOKE
TCL
(Transaction Control Language)
• COMMIT
• ROLLBACK
• SAVEPOINT
DQL
(Data Query Language)
• SELECT
Structured Query Language (SQL)

REAL-LIFE USE CASES BASED ON CODE

- MARKING ATTENDANCE – TEACHERS ENTER ATTENDANCE RECORDS IN THE ATTENDANCE TABLE
- FETCHING STUDENT ATTENDANCE – ADMINS CAN QUERY ATTENDANCE BY STUDENT OR COURSE.
- ENROLLING STUDENTS IN COURSES – STUDENTS MUST BE ADDED TO ENROLLMENT BEFORE ATTENDING.
- UPDATING STUDENT DETAILS – IF A STUDENT CHANGES THEIR EMAIL OR DEPARTMENT.
- HANDLING DATA DELETION – PREVENT ACCIDENTAL DATA LOSS USING FOREIGN KEY CONSTRAINTS.

Where Can This System Be Used?

💼 Corporate Offices & HR Training Programs

🏫 Schools, Colleges & Universities

🏥 Healthcare Training Institutes

📚 Online Learning Platforms (EdTech like Coursera, Udemy, etc.)

🏛️ Government Skill Development Programs

Conclusion:

This optimized, scalable, and secure attendance management system can be used in any educational or professional environment that requires structured attendance tracking. With indexing, security, real-time analytics, and easy scalability, it is better than traditional attendance systems and can handle high-volume data efficiently.

Advantages of AMS:-

- Faster attendance tracking
- Reduced errors
- Easy reporting

GITHUB LINK: https://github.com/Abhay-Bormale/AttendenceApp.github.io-n

THANK YOU