**Learning Management System**

**Database Design Document**

**V 2.0**

**By**

|  |  |  |
| --- | --- | --- |
| 1 | Hasnat Akram | NUM-BSCS-2023-01 |



**Department of Computer Sciences**

**Namal University**

**Mianwali, Pakistan**

**Submission Date: 2nd June 2024**

**TABLE OF CONTENTS**

[CHAPTER 1: PROJECT OVERVIEW 5](#_Toc199759588)

[1.1. INTRODUCTION: 5](#_Toc199759589)

[1.2. PROBLEM STATEMENT: 5](#_Toc199759590)

[1.3. PROJECT OBJECTIVES: 5](#_Toc199759591)

[1.4. DOCUMENT OBJECTIVES: 6](#_Toc199759592)

[CHAPTER 2: DETAILED DATABASE DESIGN 7](#_Toc199759593)

[2.1. ENTITY: 7](#_Toc199759595)

[2.2. DATA DICTIONARY: 8](#_Toc199759596)

[2.2.1. Person 8](#_Toc199759597)

[2.2.2. Admin 9](#_Toc199759598)

[2.2.3. Faculty 9](#_Toc199759599)

[2.2.4. Student 9](#_Toc199759600)

[2.2.5. Department 10](#_Toc199759601)

[2.2.6. Program 10](#_Toc199759602)

[2.2.7. Semester 10](#_Toc199759603)

[2.2.8. SemesterDetails 10](#_Toc199759604)

[2.2.9. Course 11](#_Toc199759605)

[2.2.10. Class 11](#_Toc199759606)

[2.2.11. CourseAllocation 11](#_Toc199759607)

[2.2.12. Enrollment 12](#_Toc199759608)

[2.2.13. Lecture 12](#_Toc199759609)

[2.2.14. Attendance 12](#_Toc199759610)

[2.2.15. Assessment 13](#_Toc199759611)

[2.2.16. AssessmentChecked 13](#_Toc199759612)

[2.2.17. Result 13](#_Toc199759613)

[2.2.18. Transcipt 14](#_Toc199759614)

[2.2.19. Reviews 14](#_Toc199759615)

[2.2.20. Alumni 14](#_Toc199759616)

[2.2.21. LoginCredentials 14](#_Toc199759617)

[2.2.22. Salary 15](#_Toc199759618)

[2.2.23. Qualification 15](#_Toc199759619)

[2.2.24. Address 16](#_Toc199759620)

[2.2.25. AuditTrail 16](#_Toc199759621)

[2.3. RELATIONSHIPS: 16](#_Toc199759622)

[2.4. ENTITY RELATIONSHIP DIAGRAM: 20](#_Toc199759623)

[Chapter 3 : Logical DATABASE DESIGN 21](#_Toc199759624)

[**3.1.** **RELATIONAL SCHEMA:** 21](#_Toc199759625)

[**3.2.** **FUNCTIONAL DEPENDENCIES:** 31](#_Toc199759626)

[**3.3.** **NORMALIZATION:** 31](#_Toc199759627)

[Chapter 4 : Physical DATABASE DESIGN 33](#_Toc199759628)

[**4.1.** **STRUCTURE OF THE TABLES:** 33](#_Toc199759629)

[**4.2.** **DATA SAMPLES INSIDE TABLES:***.* 40](#_Toc199759630)

[**4.3.** **QUERIES RESULTS:** 51](#_Toc199759631)

# PROJECT OVERVIEW

## INTRODUCTION:

Education has undergone a revolution with the continuous growth of recent technologies. Most recently, online learning is more efficient and available than ever before. Learning Management Systems (LMS) are an integral part of education today because they offer a systematic way of managing courses, students, and teachers. This proposal is about developing a learning management system (LMS) as the semester project for the course of database systems. The LMS will be designed in such a way that it will allow effortless interaction between the administrators, teachers, and students.

LMS are helpful in managing educational processes efficiently at institutions and organizations. The system will be developed with the intention of offering a wide range of features user experience, where students have easy access to learning materials, teachers are allowed to manage course content, and administrators are able to manage users effectively.

## PROBLEM STATEMENT:

The classic learning management systems are often not simple, responsive, and user-friendly. Most LMS today provide a platform for teachers and students to interact. What they lack is the involvement of the admin at the same time. This project aims to develop a web-based management software where are the three pillars of academia can interact with one another simultaneously, where students will be able to visualize their progress across their courses, teachers will be able to manage their classes and admin will be allowed to update and modify the database of student and teachers as well as interact with faculty and students directly by staying on the same platform.

## PROJECT OBJECTIVES:

The primary goal of this project is to develop a robust, web-based Learning Management System (LMS) integrated with an underlying database for NAMAL University, Mianwali.

**Key Objectives:**

* The system will be an integrated cohesive platform for the three main academic roles, administrators, teachers, and students.
* The system will allow the creation and management of a detailed database containing records from admission through graduation, including course, program, and semester information as well as login credentials.
* The system will enable the LMS to operate seamlessly on top of this database, allowing for accurate data retrieval and dynamic modification.
* Efficient management, real-time interactions, and continuous academic progress monitoring for all users will be provided by the system.

## DOCUMENT OBJECTIVES:

The objectives of this document are to:

1. Provide a comprehensive description of the database design for the Learning Management System, spanning conceptual, logical, and physical design phases.
2. Outline the structure and organization of each section to ensure clarity and ease of navigation for developers, stakeholders, and future maintainers.
3. Detail the entities, relationships, and constraints to be implemented, serving as a reference for accurate database implementation.
4. Present the relational schema, functional dependencies, and normalization steps to demonstrate systematic design decisions that prevent data anomalies and ensure integrity.
5. Serve as a practical guide for developers during database creation, ensuring consistency with project requirements and facilitating future enhancements and maintenance.

# DETAILED DATABASE DESIGN



## ENTITY:

In this section, we identify and define the key entities that form the core of the LMS database. Each entity represents a distinct object or concept that the system needs to track.

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Entity Name** | **Description** |
| 01 | Person | Represents the parent entity for admin, faculty and student representing their common attributes |
| 02 | Admin | Represents administrative staff responsible for managing overall system data and configurations. |
| 03 | Faculty | Represents faculty members responsible for delivering courses, managing class attendance, and grading. |
| 04 | Student | Represents learners enrolled in courses, tracking their academic progress and records. |
| 05 | Department | Representing the department of the academic institution |
| 06 | Program | Represents academic degree programs (e.g., BS Computer Science, BS Electrical Engineering) offered by the institution. |
| 07 | Semester | Represents the semester that takes place |
| 08 | SemesterDetails | The associative entity between course and semester to reflect the courses in a semester |
| 09 | Course | Represents an academic course offered by the institution, including course description, credits, and associated program/semester. |
| 10 | Class | Represents the class of students i.e. the student admitted in the same batch in the same program |
| 11 | CourseAllocation | The associative entity between teacher and course which shows the relation between teacher and the course he/she teaches. |
| 12 | Enrollment | Maps students to enroll in courses and semesters, tracking academic progress and registration data. |
| 13 | Lecture | Stores the record of each lecture held |
| 14 | Attendance | Stores records of attendance for students in specific classes, capturing daily or session-based participation. |
| 15 | Assessment | Represents the assessment taken by an instructor in a course allocation |
| 16 | AssessmentChecked | The associative entity between assessment and enrollment to capture the marks obtained by a student in a particular assessment |
| 17 | Result | Represent the result of specific enrollment i.e. result of student in a specific course in a semester |
| 18 | Transcript | Represents the result of student in particular semester |
| 19 | Reviews | Represents the reviews of students regarding an allocation in which they are enrolled |
| 20 | Alumni | Represents records of students who have graduated, including graduation data and post-graduation tracking. |
| 21 | LoginCredentials | Stores authentication information such as username, hashed password, and security tokens separately from user profile data. |
| 22 | Salary | Stores salary record of admin and teachers. |
| 23 | Qualification | Stores the qualification records of admin, teachers and students. |
| 24 | Address | Stores the address of the person. |
| 25 | AuditTrail | Stores a secure log of who did what, when, where, and how in the application. |

*.*

## DATA DICTIONARY:

In the "Attributes" section, identify and define the specific characteristics or properties of each of the entities in the database. Provide a list of attributes associated with the chosen entity. Each attribute should represent a specific piece of information or characteristic that must be stored in the database. Specify the data type for each attribute to indicate the kind of data it will store. Common data types include text, numbers, dates, Boolean values. Identify any constraints or rules that apply to the attribute values. This may include constraints such as required fields, unique values, minimum or maximum lengths, or format requirements. For each attribute, provide a clear and concise definition or description. Explain what the attribute represents and its significance within the context of the entity.

* + 1. Person**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | personID (primary key) | VARCHAR(20) | NOT NULL, Unique | A unique identifier for each person who is associated with the system |
| 02 | fname | VARCHAR(50) | NOT NULL | First name of the person |
| 03 | lname | VARCHAR(50) | NOT NULL | Last name of the person |
| 04 | personalEmail | VARCHAR(100) |  | Email address for personal communication |
| 05 | institutionalEmail | VARCHAR(100) | NOT NULL, unique | Email address given by institution for communication and administrative purposes |
| 06 | CNIC | VARCHAR(15) | NOT NULL, | CNIC number of the person |
| 07 | gender | ENUM | NOT NULL,  (M, F ,O) | Gender of the person |
| 08 | DOB | DATE | NOT NULL | Date of birth according to the CNIC of the person |
| 09 | cNumber | VARCHAR(15) | NOT NULL, valid number | Optional number for contact purposes |
| 10 | type | ENUM ('Admin', 'Faculty', 'Student') | NOT NULL | determinative of the person table |

* + 1. Admin**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | employeeID (primary key) | VARCHAR(20) | NOT NULL, Unique | A unique identifier for admin |
| 03 | joiningDate | DATE | NOT NULL | The date when the admin joined or was registered in the system. |
| 04 | leavingDate | DATE |  | The date when admin left the system (in other words his designation ) |
| 05 | officeLocation | VARCHAR(100) |  | Location of admin office in university building |

* + 1. Faculty**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | employeeID (Primary Key) | VARCHAR(20) | NOT NULL | A unique identifier of each teacher |
| 02 | designation | VARCHAR(50) | NOT NULL | Job title (e.g., Lecturer, Assistant Professor) |
| 03 | departmentID (foreign key) | INT | NOT NULL | Department teacher belongs to |
| 04 | joiningDate | DATE | NOT NULL | Date of joining the institute |

* + 1. Student**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | studentID (Primary Key) | VARCHAR(20) | NOT NULL, unique | Unique identifier for each student |
| 02 | classID (foreign key) | INT | NOT NULL | Student’s department at university |
| 03 | programID (foreign key) | VARCHAR(10) | NOT NULL | Degree program student is enrolled in |
| 04 | status | ENUM | NOT NULL, ( enrolled, graduated, dropped) | Academic status |

* + 1. Department**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | departmentID (Primary Key) | INT | NOT NULL, unique | Unique ID for each department |
| 02 | departmentName | VARCHAR(50) | NOT NULL | Full name of the department |
| 03 | HOD (foreign key) | VARCHAR(50) | NOT NULL | The faculty member who is the head of department |

* + 1. Program**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | programID (Primary Key) | VARCHAR(10) | NOT NULL | ProgramID |
| 02 | programName | VARCHAR(50) | NOT NULL | Name of the program |
| 03 | totalSemesters | INT |  | Total semesters in which program is completed |
| 04 | departmentID (foreign key) | INT | NOT NULL | Offering department |

* + 1. Semester**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | semesterID (Primary Key) | INT | NOT NULL | Semester ID unique for each semester |
| 02 | programID (foreign key) | VARCHAR(20) | NOT NULL | Name of the program to which the semester belongs |
| 03 | fee | Integer |  | Fee of the semester |
| 04 | semesterNo | Integer | NOT NULL | Number of the semester ( 1 to 8/ 10) |

* + 1. SemesterDetails**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | semesterID (Primary Key) | INT | NOT NULL, unique | Semester ID from semester table |
| 02 | courseID (Primary key) | VARCHAR(10) | NOT NULL | Course ID from course table |
| 03 | classID (primary key) | VARCHAR(20) | NOT NULL | class for which the semester is scheduled |
| 04 | session | VARCHAR(10) |  | Session in which the semester takes place |

* + 1. Course**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | courseCode (Primary Key) | VARCHAR(20) | NOT NULL | Official course code, unique identifier for each course |
| 02 | courseName | VARCHAR(50) | NOT NULL | Full name of the course |
| 03 | creditHours | INT | NOT NULL | Number of credit hours assigned to the course |
| 04 | pre-requisite (Foreign key) | VARCHAR(20) |  | Course code of the pre-requisite course |
| 05 | programID (foreign key) | VARCHAR(10) | NOT NULL | Degree program for which the course is offered |
| 06 | description | TEXT |  | Short summary of the course content or objectives |

* + 1. Class**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | classID (Primary Key) | INT | NOT NULL, | Unique identifier for the class |
| 02 | programID (foreign key) | VARCHAR(10) | NOT NULL | Name of the academic program |
| 03 | batchNo | YEAR | NOT NULL | Graduation year of the class |

* + 1. CourseAllocation**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | allocationID (Primary key) | INT | NOT NULL | Unique ID for each allocation |
| 02 | teacherID (Foreign Key) | VARCHAR(20) | NOT NULL | ID of the assigned teacher |
| 03 | courseCode (Foreign Key) | VARCHAR(20) | NOT NULL | Course code |
| 04 | session | VARCHAR(20) |  | Spring or fall |
| 05 | status | ENUM | DEFAULT = ongoing | Status of allocation (completed/ongoing/ cancelled) |

* + 1. Enrollment**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | erollmentID (Primary Key) | INT | NOT NULL | unique ID for each enrollment |
| 02 | studentID (Foreign key) | VARCHAR(20) | NOT NULL | ID of the student who is enrolled |
| 03 | allocationID(Foreign key) | INT | NOT NULL | ID of the course allocation in which student is enrolled |
| 04 | enrollmentDate | DATETIME |  | Date on which enrollment is done |
| 05 | status | ENUM | DEFAULT = ACTIVE | Status of enrollment (active, dropped, completed) |

* + 1. Lecture**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | lectureID (Primary Key) | VARCHAR (10) | NOT NULL | Unique ID for each record |
| 02 | allocationID (foreign Key) | INT | NOT NULL | ID of the allocation to which this lecture belongs |
| 04 | lectureNo | INT | NOT NULL | Lecture no of a particular allocation |
| 03 | venue | VARCHAR(50) | NOT NULL | Physical location of the lecture |
| 04 | startingTime | DATETIME | NOT NULL | Starting time of the lecture |
| 05 | endingTime | DATETIME | NOT NULL | Ending time of the lecture |
| 06 | topic | Text |  | Topic of the lecture delivered |

* + 1. Attendance**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | date (primary key) | DATETIME | NOT NULL | Date of the attendance |
| 02 | studentID (primary key) | VARCHAR(20) | NOT NULL | ID of the student |
| 03 | lectureID (primary key) | VARCHAR(10) | NOT NULL | ID of the lecture |

* + 1. Assessment**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | assessmentID (Primary Key) | INT | NOT NULL | Unique ID for each record |
| 02 | allocationID (Foreign key) | INT | NOT NULL | ID of the allocation for which the quiz is scheduled |
| 03 | assessmentType | ENUM | NOT NULL | type of assessment (quiz, assignment , mid or finals ) |
| 05 | assessmentName | VARCHAR(20) | NOT NULL | Name of the assessment |
| 04 | weightage | INT | NOT NULL | Weightage of assessment in final result |
| 05 | date | DATE |  | Date on which assessment is held |
| 06 | totalMarks | INT | NOT NULL | Total marks of the assessment |

* + 1. AssessmentChecked**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | assessmentID (Primary Key) | INT | NOT NULL | Unique ID for each record |
| 02 | enrollmentID (Primary key) | INT | NOT NULL | ID of the enrollment (student) for whom assessment was scheduled |
| 03 | resultID (Foreign key) | INT | NOT NULL | The result ID of which this assessment is a part of |
| 04 | obtained | DECIMAL(10,2) | NOT NULL | Marks obtained by the student |

* + 1. Result**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | resultID (Primary Key) | INT | NOT NULL | Unique ID for each result |
| 02 | enrollmentID (Foreign key) | INT | NOT NULL | ID of the enrollment of which the result is |
| 03 | courseGPA | DECIMAL(4,2) | NOT NULL | GPA obtained by student in the enrollment |

* + 1. Transcipt**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | studentID (Primary Key) | VARCHAR(20) | NOT NULL | ID of the student to which the transcript belongs |
| 02 | semesterID (Primary key) | INT | NOT NULL | ID of the semester of the transcripts is |
| 03 | totalCredits | INT | NOT NULL | Total credits passed by the student in the semester |
| 04 | semesterGPA | DECIMAL(4,2) | NOT NULL | GPA obtained by the student in the semester. |

* + 1. Reviews**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | reviewID (Primary Key) | INT | NOT NULL | Unique ID for each record |
| 02 | enrollmentID (Foreign key) | INT | NOT NULL | ID of the enrollment about which the review record is |
| 04 | createdAt | DATE | NOT NULL | Time at which the review was written |
| 03 | reviews | TEXT |  | The actual text of reviews |

* + 1. Alumni**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | alumniID (Primary Key) | INT | NOT NULL | Alumni ID of the student |
| 02 | graduationDate | DATE | NOT NULL | the date of graduation |
| 03 | email | VARCHAR(100) |  | Email address for contact |
| 04 | employmentInfo | Text |  | Employment info of the alumni |

* + 1. LoginCredentials**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | personID (Primary Key) | VARCHAR(20) | NOT NULL | ID of teacher or student |
| 02 | username | VARCHAR(20) | NOT NULL, UNIQUE | Username of the person |
| 03 | password | VARCHAR(16) | NOT NULL | Password |
| 04 | recoveryEmail | VARCHAR(20) | NOT NULL | Recovery email |

* + 1. Salary**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | salaryID (Primary Key) | INT | NOT NULL | Unique identifier for each salary record |
| 02 | employeeID (Foreign Key) | VARCHAR(20) | NOT NULL | Employee’s ID (either admin or teacher) to whom the record belongs |
| 03 | month | VARCHAR(10) | NOT NULL | Month for which salary was paid |
| 04 | amount | DECIMAL (10,2) | NOT NULL | Salary amount paid in that month |
| 05 | paymentDate | DATETIME | NOT NULL | Actual date salary was disbursed |

* + 1. Qualification**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | qualificationID (Primary Key) | INT | NOT NULL | Unique identifier for each qualification record |
| 02 | personID (Foreign Key) | VARCHAR(20) | NOT NULL | Person’s ID (either admin, teacher or student) to whom the record belongs |
| 03 | degreeTitle | VARCHAR(50) | NOT NULL | Title of the degree |
| 04 | educationBoard | VARCHAR(20) |  | Board of education from where the degree is issued |
| 05 | institution | VARCHAR(50) | NOT NULL | Name of the university or institution |
| 06 | passingYear | YEAR |  | Year of graduation or expected completion |
| 07 | totalMarks | INT |  | Total Marks or CGPA |
| 08 | obtainedMarks | INT |  | Obtained Marks or CGPA |
| 09 | isCurrent | Boolean | Default: False | TRUE if the degree is ongoing; FALSE if completed |

* + 1. Address**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 02 | personID (Primary Key) | VARCHAR(20) | NOT NULL | Person’s ID (either admin, teacher or student) to whom the address belongs |
| 03 | country | VARCHAR(50) | NOT NULL | country in the address |
| 04 | province | VARCHAR(50) |  | name of the province |
| 05 | city | VARCHAR(50) | NOT NULL | city name |
| 06 | zipCode | INT | NOT NULL | zip code or postal code of city |
| 07 | streetAddress | VARCHAR(100) |  | street address of home |

* + 1. AuditTrail**:**

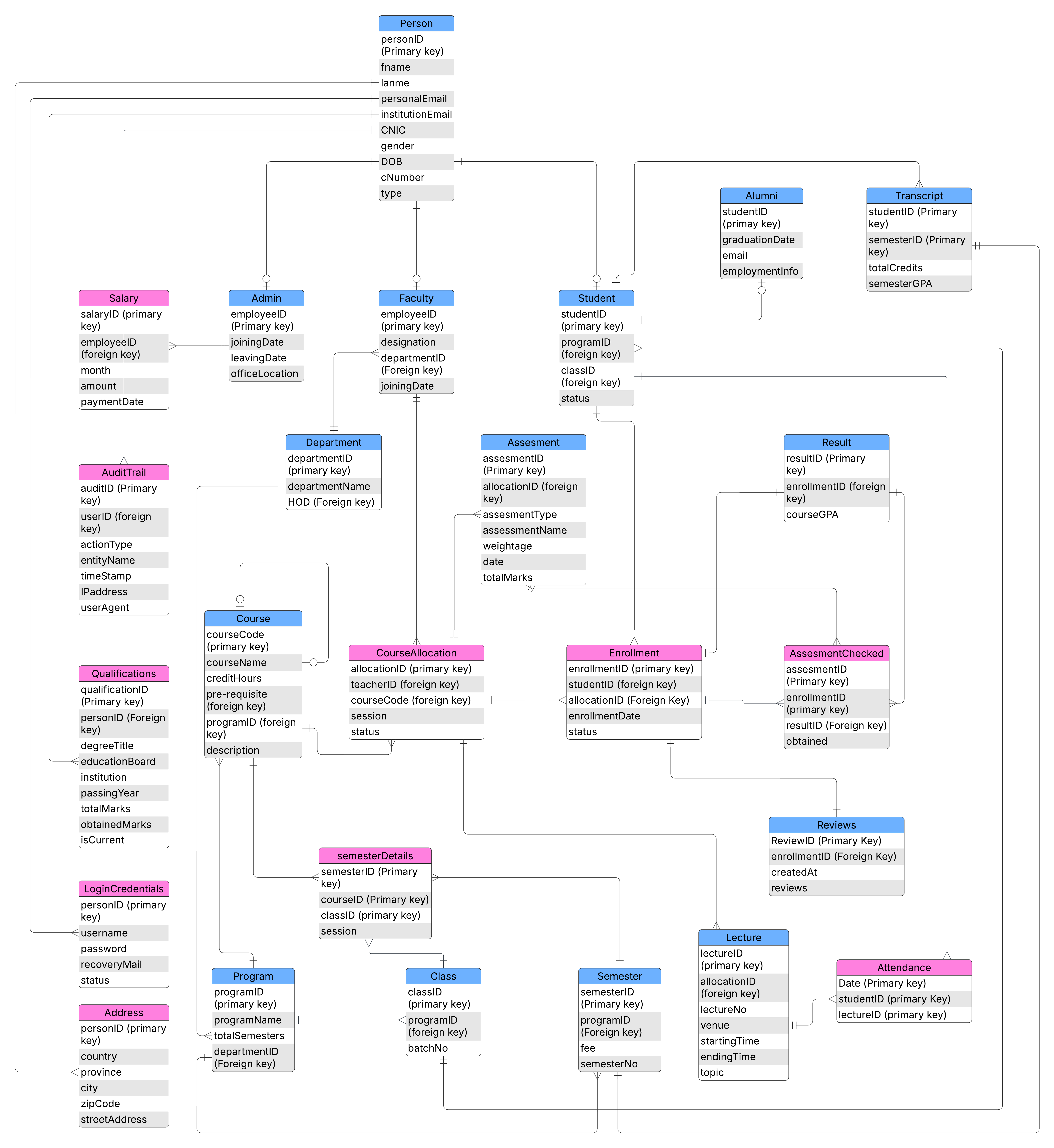
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | auditID (primary key) | INT | NOT NULL | Unique ID for each audit record |
| 02 | userID (foreign key) | VARCHAR(20) | NOT NULL | ID of the person who has done the audit |
| 03 | actionType | ENUM | NOT NULL | Type of action which is performed (create, read, update, delete, login, logout) |
| 04 | entityName | VARCHAR(50) | NOT NULL | Name of the entity affected |
| 05 | timeStamp | DATETIME | NOT NULL | Time at which the audit was done |
| 06 | IPaddress | VARCHAR(45) | NOT NULL | IP address of the user |
| 07 | UserAgent | VARCHAR(225) | NOT NULL | Details about the machine of the user |

## RELATIONSHIPS:

Defining the relationships between entities is crucial for maintaining data integrity and reflecting real-world associations. Below are some example relationships:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Participating Entities** | **Relation** | **Business Rule** |
| 1 | Program, Course | Course offered in Program | A Program may offer many Courses; each Course is offered by exactly one Program. |
| 2 | Program, Class | Class scheduled under Program | A Program may comprise multiple Classes; each Class is associated with exactly one Program. |
| 3 | Teacher, CourseAllocation | Teacher assigned via CourseAllocation | A Teacher may have many CourseAllocations; each CourseAllocation references exactly one Teacher. |
| 4 | Course, CourseAllocation | Course taught via CourseAllocation | A Course may be allocated in many CourseAllocations; each CourseAllocation refers to exactly one Course. |
| 5 | Class, CourseAllocation | CourseAllocation scheduled for Class | A Class may host multiple CourseAllocations; each CourseAllocation is tied to exactly one Class. |
| 6 | CourseAllocation, Lecture | Lecture scheduled for CourseAllocation | Each CourseAllocation schedules many Lectures; each Lecture belongs to exactly one CourseAllocation. |
| 7 | Student, Enrollment | Student enrolled via Enrollment | A Student may have many Enrollments; each Enrollment record refers to exactly one Student. |
| 8 | CourseAllocation, Enrollment | Enrollment recorded in CourseAllocation | A CourseAllocation may include many Enrollments; each Enrollment is for exactly one CourseAllocation. |
| 9 | Enrollment, Assessment | Assessment recorded for Enrollment | Each Enrollment may generate multiple Assessment records; each assessment belongs to exactly one Enrollment. |
| 10 | Lecture, Attendance | Attendance taken at Lecture | Each Lecture produces many Attendance records; each Attendance record is for exactly one Lecture. |
| 11 | Student, Attendance | Attendance entry for Student | A Student may have many Attendance records; each Attendance entry refers to exactly one Student. |
| 12 | Student, Alumni | Student becomes Alumni | A Student may become an Alumni (one Alumni record) once graduated; not all Students are Alumni. |
| 13 | Student, LoginCredentials | LoginCredentials issued to Student | Each Student has exactly one LoginCredentials record; each LoginCredentials entry belongs to one Student. |
| 14 | Teacher, LoginCredentials | LoginCredentials issued to Teacher | Each Teacher has exactly one LoginCredentials record; each LoginCredentials entry belongs to exactly one Teacher. |
| 15 | Employee (Admin/Teacher), Salary | Salary paid to Employee | Each Admin or Teacher (Employee) may have multiple Salary records; each Salary entry refers to exactly one Employee. |
| 16 | Person (Admin/Teacher/Student), Qualification | Qualification held by Person | Any Admin, Teacher, or Student (Person) may hold multiple Qualifications; each Qualification belongs to one Person. |
| 17 | Course, Course (Prerequisite) | Course requires prerequisite Courses | A Course may require multiple prerequisite Courses; a Course may be a prerequisite for multiple other Courses. |
| 18 | Department, Faculty | Department headed by Faculty | Each Department has one Head of Department (Faculty); each Faculty may be Head of at most one Department. |
| 19 | Department, Program | Program offered by Department | A Department may offer many Programs; each Program belongs to exactly one Department. |
| 20 | Program, Semester | Semester part of Program | A Program comprises many Semesters; each Semester belongs to exactly one Program. |
| 21 | Semester, SemesterDetails | SemesterDetails detail for Semester | A Semester may include many SemesterDetails; each SemesterDetails pertains to exactly one Semester. |
| 22 | Course, SemesterDetails | Course part of SemesterDetails | A Course may appear in many SemesterDetails; each SemesterDetails includes exactly one Course. |
| 23 | Class, SemesterDetails | SemesterDetails for Class | A Class may have many SemesterDetails; each SemesterDetails applies to exactly one Class. |
| 24 | CourseAllocation, Assessment | Assessment for CourseAllocation | A CourseAllocation may have many Assessments; each Assessment is for exactly one CourseAllocation. |
| 25 | Enrollment, AssessmentChecked | AssessmentChecked for Enrollment | An Enrollment may have many AssessmentChecked entries; each AssessmentChecked refers to exactly one Enrollment. |
| 26 | Assessment, AssessmentChecked | AssessmentChecked linked to Assessment | An Assessment may be checked for many Enrollments; each AssessmentChecked references exactly one Assessment. |
| 27 | Result, AssessmentChecked | AssessmentChecked part of Result | A Result may include many AssessmentChecked entries; each AssessmentChecked is included in exactly one Result. |
| 28 | Enrollment, Result | Result for Enrollment | An Enrollment may have one Result; each Result corresponds to exactly one Enrollment. |
| 29 | Student, Transcript | Transcript record for Student | A Student may have many Transcripts; each Transcript belongs to exactly one Student. |
| 30 | Semester, Transcript | Transcript for Semester | A Semester may have many Transcripts; each Transcript pertains to exactly one Semester. |
| 31 | Enrollment, Reviews | Reviews for Enrollment | An Enrollment may have many Reviews; each Review refers to exactly one Enrollment. |
| 32 | Person, Address | Address of Person | A Person may have many Addresses; each Address belongs to exactly one Person. |
| 33 | Person, AuditTrail | AuditTrail entries by Person | A Person may perform many actions recorded in AuditTrail; each AuditTrail entry is performed by exactly one Person. |
| 34 | Student, Class | Student assigned to Class | A Class may include many Students; each Student belongs to exactly one Class. |
| 35 | Admin, LoginCredentials | LoginCredentials for Admin | Each Admin has exactly one LoginCredentials record; each LoginCredentials entry belongs to one Admin. |

## ENTITY RELATIONSHIP DIAGRAM:

**

# : Logical DATABASE DESIGN

* 1. **RELATIONAL SCHEMA:**

Below is a concise description of each relation (table) in the schema. For each table, we list its primary key (PK), all non‐key attributes (columns), and any foreign‐key (FK) references that enforce relationships. Wherever a many‐to‐many relationship exists, an associative (bridge) table has been introduced with a composite primary key.

* + 1. **Relations (Tables)**
       1. **Person**

**Table Name :** Person

**Primary Key :** personID

**Columns:**

* personID; Surrogate key, UNIQUE
* firstName
* lastName
* personalEmail
* institutionalEmail
* CNIC; National ID (UNIQUE).
* gender
* dateOfBirth
* contactNumber
* personType; determinator

**Relationships:**

* Any row in Admin, Faculty, or Student must have a corresponding personID here.
* One‐to‐one “is-a” subtyping is enforced by having personID as a FK (and PK) in each subtype table.
  + - 1. **Admin**

**Table Name:** Admin

**Primary Key:** employeeID (FK → Person(personID))

**Columns:**

* employeeID
* joiningDate
* leavingDate
* officeLocation

**Relationships:**

* The one‐to‐one link is enforced by Admin.personID referencing Person(personID); each Admin row corresponds to exactly one Person.
  + - 1. **Faculty**

**Table Name:** Faculty

**Primary Key:** employeeID (FK → Person(personID))

**Columns:**

* employeeID
* designation
* departmentID; FK → Department(departmentID) (one‐to‐many).
* joiningDate

**Relationships:**

* Each Faculty row links back to exactly one Person (personID).
* departmentID enforces that a faculty member belongs to one Department.
  + - 1. **Student**

**Table Name:** Student

**Primary Key:** (FK → Person(personID))

**Columns:**

* studentID
* classID; FK → Class(classID) (one‐to‐many).
* programID; FK → Program(programID) (one‐to‐many).
* status

**Relationship:**

* Each Student row corresponds to exactly one Person.
* classID indicates the cohort (Class) in which the student is enrolled.
* programID indicates the academic Program (degree) to which the student belongs.
  + - 1. **Department**

**Table Name:** Department

**Primary Key:** departmentID

**Columns:**

* departmentID
* departmentName
* HOD; FK → Faculty(employeeID) (optional, one‐to‐zero/one).

**Relationship:**

* If a department has a Head of Department (HOD), the HOD column points to the corresponding Faculty record.
* One Department can have many Faculty (inverse of HOD).
  + - 1. **Program**

**Table Name:** Program

**Primary Key:** programID (surrogate)

**Columns:**

* programID
* programName
* totalSemesters
* departmentID; FK → Department(departmentID) (one‐to‐many).

**Relationship:**

* Each Program belongs to exactly one Department (via departmentID).
* One Department can offer multiple Programs.
  + - 1. **Class**

**Table Name:** Class

**Primary Key:** classID (surrogate)

**Columns:**

* classID
* programID; FK → Program(programID) (one‐to‐many).
* batchNumber

**Relationship:**

* Each Class is linked to exactly one Program.
* One Program can have multiple Classes (e.g., different cohorts, different years).
  + - 1. **Semester**

**Table Name:** Semester

**Primary Key:** semesterID (surrogate)

**Columns:**

* semesterID
* programID; FK → Program(programID) (one‐to‐many).
* fee
* semesterNumber

**Relationship:**

* Each Semester belongs to exactly one Program.
* One Program can define multiple Semester entries.
  + - 1. **Course**

**Table Name:** Course

**Primary Key:** courseCode (Natural key, e.g., ‘CS101’)

**Columns:**

* courseCode
* courseName
* creditHours
* prerequisite; FK → Course(courseCode) (self-reference, nullable).
* programID; FK → Program(programID) (one-to-many).
* description

**Relationship:**

* Each Course is assigned to exactly one Program.
* If a course requires a prerequisite, preRequisite points to another courseCode in this same table.
* One Program can have multiple Courses.
  + - 1. **SemesterDetails**

**Table Name:** SemesterDetails

**Primary Key (Composite):** (semesterID, courseCode, classID)

**Columns:**

* semesterID; FK → Semester(semesterID) (part of composite PK).
* courseCode; FK → Course(courseCode) (part of composite PK).
* classID; FK → Class(classID) (part of composite PK).
* session

**Relationship:**

* Bridge table for the M:N relationship between Course, Class, and Semester
* A given Course can be offered to multiple Classes across different Semesters.
* A given Class can take multiple Courses in a given Semester.
* Each row means “Course X is offered to Class Y in Semester Z (Session W).”
  + - 1. **CourseAllocation**

**Table Name:** CourseAllocation

**Primary Key:** allocationID (surrogate)

**Columns:**

* allocationID
* teacherID; FK → Faculty(employeeID) (one-to-many).
* courseCode; FK → Course(courseCode) (one-to-many).
* classID; FK → Class(classID) (one-to-many).
* session
* status

**Relationship:**

* Each CourseAllocation ties one Faculty member to one Course for one Class in a particular session.
* The session field should match an existing triple in SemesterDetails (though referential enforcement can be handled at the application level or via a CHECK constraint).
  + - 1. **Enrollment**

**Table Name:** Enrollment

**Primary Key:** enrollmentID (surrogate)

**Columns:**

* enrollmentID.
* studentID; FK → Student(studentID) (one-to-many).
* allocationID; FK → CourseAllocation(allocationID) (one-to-many).
* enrollmentDate
* status.

**Relationship:**

* Each row represents one student registering for one course allocation.
* One CourseAllocation can have many Enrollment rows (many students).
* One Student can appear in many Enrollment rows (across different course allocations)
  + - 1. **Lecture**

**Table Name:** Lecture

**Primary Key:** lectureID (derived key ( alloactionID + lectureNo ))

**Columns:**

* lectureID
* allocationID; FK → CourseAllocation(allocationID) (one-to-many).
* lectureNumber
* venue
* startTime
* endTime
* topic **Relationship:**
* Each CourseAllocation can have multiple Lecture rows (i.e., multiple sessions).
  + - 1. **Attendance**

**Table Name:** Attendance

**Primary Key (Composite):** (attendanceDate, studentID, lectureID)

**Columns:**

* attendanceDate
* studentID; FK → Student(studentID) (part of composite PK).
* lectureID; FK → Lecture(lectureID) (part of composite PK).

**Relationship:**

* Bridge table for the M:N relationship between Student and Lecture
* A given student may attend multiple lectures on the same or different dates.
* A given lecture may have many students attending on a given date.
* Each attendance record states: “On Date D, Student S attended Lecture L.”
  + - 1. **Assessment**

**Table Name:** Assessment

**Primary Key:** assessmentID (surrogate)

**Columns:**

* assessmentID
* allocationID; FK → CourseAllocation(allocationID) (one-to-many).
* assessmentType
* assessmentName
* weightage
* assessmentDate
* totalMarks

**Relationship:**

* Each CourseAllocation can have multiple Assessment records (different quizzes, exams, etc.)
  + - 1. **AssessmentChecked**

**Table Name:** AssessmentChecked

**Primary Key (Composite):** (assessmentID, enrollmentID)

**Columns:**

* assessmentID; FK → Assessment(assessmentID) (part of composite PK).
* enrollmentID; FK → Enrollment(enrollmentID) (part of composite PK).
* resultID; FK → Result(resultID) (optional, if results are stored separately).
* marksObtained

**Relationship:**

* Bridge table for the M:N relationship between Assessment and Enrollment
* Each row captures one student’s marks on one specific assessment.
* A given Assessment can have many AssessmentChecked rows (many students).
* A given Enrollment can appear in many AssessmentChecked rows (one per assessment)
  + - 1. **Result**

**Table Name:** Result

**Primary Key:** resultID (surrogate)

**Columns:**

* resultID
* enrollmentID; FK → Enrollment(enrollmentID) (one-to-one).
* courseGPA

**Relationship:**

* Each Enrollment can have exactly one Result.
* The one‐to‐one relationship is enforced by making resultID reference a unique enrollmentID.
  + - 1. **Transcript**

**Table Name:** Transcript

**Primary Key (Composite):** (studentID, semesterID)

**Columns:**

* studentID; FK → Student(studentID) (part of composite PK).
* semesterID; FK → Semester(semesterID) (part of composite PK).
* totalCredits
* semesterGPA

**Relationship:**

* Bridge table for the M:N relationship between Student and Semester
* Indicates which student‐semester pairs exist and stores their aggregate performance.
* A given Student can have multiple Transcript rows (one per semester).
* A given Semester can have multiple Transcript rows (one per student).
  + - 1. **Reviews**

**Table Name:** Reviews

**Primary Key:** reviewID (surrogate)

**Columns:**

* reviewID
* enrollmentID; FK → Enrollment(enrollmentID) (one-to-many).
* createdAt
* reviewText

**Relationship:**

* Each Enrollment can generate zero or more Reviews (one‐to‐many).
* A given review always corresponds to exactly one student‐course enrollment.
  + - 1. **Alumni**

**Table Name:** Alumni

**Primary Key:** alumniID (FK → Student(studentID))

**Columns:**

* alumniID
* graduationDate
* email
* employmentInfo

**Relationship:**

* One‐to‐one: once a Student’s status becomes ‘Graduated’, they may have a corresponding Alumni row.
  + - 1. **LoginCredentials**

**Table Name:** LoginCredentials

**Primary Key:** personID (FK → Person(personID))

**Columns:**

* PersonID; PK and FK referencing Person(personID) (one-to-one).
* Username; UNIQUE.
* password
* recoveryEmail

**Relationship:**

* Each Person has exactly one set of credentials.
* One‐to‐one: prevents multiple credentials per person.
  + - 1. **Salary**

**Table Name:** Salary

**Primary Key:** salaryID (surrogate)

**Columns:**

* salaryID
* employeeID; FK → Person(personID)
* month
* amount
* paymentDate

**Relationship:**

* Each salary record ties back to exactly one employee (who must be in either Faculty or Admin)
* One‐to‐many: an employee can have many salary records over different months.
  + - 1. **Qualification**

**Table Name:** Qualification

**Primary Key:** qualificationID (surrogate)

**Columns:**

* qualificationID.
* personID; FK → Person(personID) (one-to-many).
* degreeTitle
* educationBoard
* institution
* passingYear
* totalMarks
* marksObtained
* isCurrent

**Relationship:**

* Each Person can have multiple Qualification rows (one-to-many).
  + - 1. **Address**

**Table Name:** Address

**Primary Key:** personID (FK → Person(personID))

**Columns:**

* personID; FK→ referencing Person(personID) (one-to-one).
* country
* province
* city
* postalCode
* streetAddress

**Relationship:**

* One‐to‐one: each Person may have exactly one Address (design choice).
* If multiple addresses were required, Address would need its own surrogate key and a composite PK; here, we assume only one.
  + - 1. **AuditTrail**

**Table Name:** AuditTrail

**Primary Key:** auditID (surrogate)

**Columns:**

* auditID
* userID; FK → Person(personID) (one-to-many).
* actionType
* entityName
* timeStamp
* ipAddress
* userAgent.

**Relationship:**

* Each action in the system is logged with the userID who performed it.
* One‐to‐many: one Person can appear in many AuditTrail rows over time.
  + 1. **Relationship Cardinalities**
       1. **One-to-One (1:1)**
* Person → Admin: Admin.personID is both PK and FK to Person(personID).
* Person → Faculty: Faculty.personID is both PK and FK to Person(personID).
* Person → Student: Student.personID is both PK and FK to Person(personID).
* Student → Alumni: Alumni.alumniID is both PK and FK to Student(studentID).
* Person → LoginCredentials: LoginCredentials.personID is both PK and FK to Person(personID).
* Person → Address: Address.personID is both PK and FK to Person(personID).
* Enrollment → Result: Result.enrollmentID is a FK (and can be declared UNIQUE to enforce 1:1).
  + - 1. **One-to-Many (1:N)**
* Department → Faculty: Faculty.departmentID FK → Department.departmentID.
* Department → Program: Program.departmentID FK → Department.departmentID.
* Program → Class: Class.programID FK → Program.programID.
* Program → Semester: Semester.programID FK → Program.programID.
* Program → Course: Course.programID FK → Program.programID.
* Class → Student: Student.classID FK → Class.classID.
* Class → CourseAllocation: CourseAllocation.classID FK → Class.classID.
* Course → CourseAllocation: CourseAllocation.courseCode FK → Course.courseCode.
* CourseAllocation → Lecture: Lecture.allocationID FK → CourseAllocation.allocationID.
* CourseAllocation → Assessment: Assessment.allocationID FK → CourseAllocation.allocationID.
* CourseAllocation → Enrollment: Enrollment.allocationID FK → CourseAllocation.allocationID.
* Assessment → AssessmentChecked: AssessmentChecked.assessmentID FK → Assessment.assessmentID.
* Enrollment → AssessmentChecked: AssessmentChecked.enrollmentID FK → Enrollment.enrollmentID.
* Student → Enrollment: Enrollment.studentID FK → Student.studentID.
* Student → Transcript: Transcript.studentID FK → Student.studentID.
* Semester → Transcript: Transcript.semesterID FK → Semester.semesterID.
* Enrollment → Reviews: Reviews.enrollmentID FK → Enrollment.enrollmentID.
* Person → Qualification: Qualification.personID FK → Person.personID.
* Person → Salary: Salary.employeeID FK → (Faculty.employeeID or Admin.employeeID).
* Person → AuditTrail: AuditTrail.userID FK → Person.personID.
  + - 1. **Many-to-Many (M:N)** – Broken into associative tables:
* Course ⊕ Class ⊕ Semester:

Bridge: SemesterDetails(semesterID PK, courseCode PK, classID PK, session)

Each row indicates “Course \* is offered to Class \* in Semester \*.” Triple‐FK combination enforces the M:N.

* Student ⊕ Lecture (Attendance):

Bridge: Attendance(attendanceDate PK, studentID PK, lectureID PK)

Indicates “On Date , Student attended Lecture .”

* Assessment ⊕ Enrollment:

Bridge: AssessmentChecked(assessmentID PK, enrollmentID PK, resultID, marksObtained)

Indicates “Student (via Enrollment \*) took Assessment \* and either references a Result or records marks directly.”

* Student ⊕ Semester (Transcript):
* Bridge: Transcript(studentID PK, semesterID PK, totalCredits, semesterGPA)

Indicates “Student \* completed Semester \* with X credits and GPA.”

* 1. **FUNCTIONAL DEPENDENCIES:**

For each table, the following functional dependencies (FDs) have been identified:

1. Person: personID → fname, lname, personalEmail, institutionalEmail, CNIC, gender, DOB, cNumber, type
2. Admin: employeeID → personID, joiningDate, leavingDate, officeLocation
3. Faculty: employeeID → personID, designation, departmentID, joiningDate
4. Student: studentID → personID, classID, programID, status
5. Department: departmentID → departmentName, HOD
6. Program: programID → programName, totalSemesters, departmentID
7. Class: classID → programID, batchNo
8. Semester: semesterID → programID, fee, semesterNo
9. Course: courseCode → courseName, creditHours, preRequisite, programID, description
10. SemesterDetails: (semesterID, courseID, classID) → session
11. CourseAllocation: allocationID → teacherID, courseCode, classID, session, status
12. Enrollment: enrollmentID → studentID, allocationID, enrollmentDate, status
13. Lecture: lectureID → allocationID, lectureNo, venue, startingTime, endingTime, topic
14. Attendance: (date, studentID, lectureID) → – (composite PK; no additional attributes)
15. Assessment: assessmentID → allocationID, assessmentType, assessmentName, weightage, date, totalMarks
16. AssessmentChecked: (assessmentID, enrollmentID) → resultID, obtained
17. Result: resultID → enrollmentID, courseGPA
18. Transcript: (studentID, semesterID) → totalCredits, semesterGPA
19. Reviews: reviewID → enrollmentID, createdAt, reviews
20. Alumni: alumniID → graduationDate, email, employmentInfo
21. LoginCredentials: personID → username, password, recoveryEmail
22. Salary: salaryID → employeeID, month, amount, paymentDate
23. Qualification: qualificationID → personID, degreeTitle, educationBoard, institution, passingYear, totalMarks, obtainedMarks, isCurrent
24. Address: personID → country, province, city, zipCode, streetAddress
25. AuditTrail: auditID → userID, actionType, entityName, timeStamp, IPaddress, UserAgent
    1. **NORMALIZATION:**

The normalization process ensures that the database schema adheres to the principles of 1NF, 2NF, and 3NF to eliminate data anomalies and redundancies.

1. **First Normal Form (1NF):**  
   All relations have atomic attribute values, and there are no repeating groups. Composite and multi-valued attributes have been moved to separate tables where necessary (e.g., Attendance as a bridge table).
2. **Second Normal Form (2NF):**

Every non-key attribute is fully functionally dependent on the entire primary key. Tables with composite primary keys (e.g., SemesterDetails, Attendance, AssessmentChecked, Transcript) store attributes that depend on the full composite key.

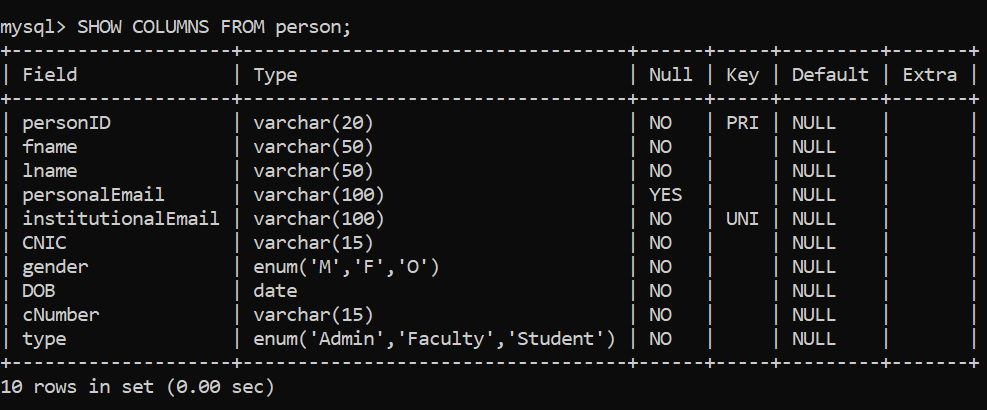
1. **Third Normal Form (3NF):**  
   All non-key attributes are non-transitively dependent on the primary key. Any transitive dependencies identified during the FD analysis have been resolved by creating separate relations. For instance, in Course, courseName, creditHours, preRequisite, and description depend directly on courseCode, and there is no indirect dependency through another attribute.

# : Physical DATABASE DESIGN

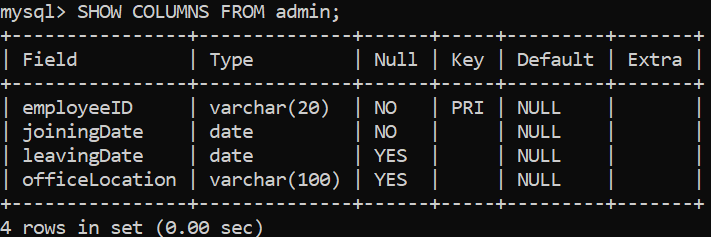
* 1. **STRUCTURE OF THE TABLES:**

Below are screenshots of each table showing their structure as well as the subsequent query that is used to show structure of that table. The query can be seen at the top of the screenshot.

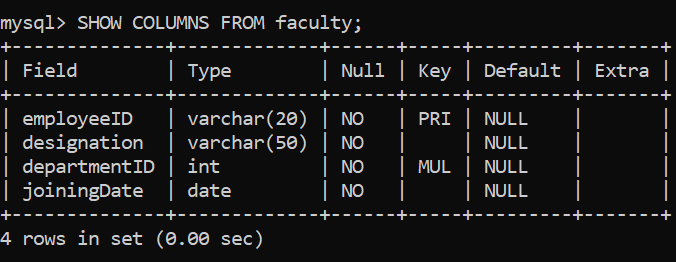
1. **Person**

****

1. **Admin**

**

1. **Faculty**

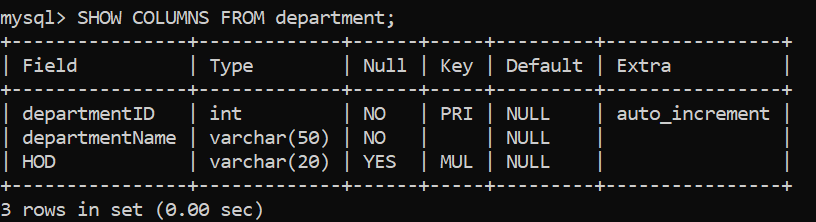
****

1. **Student**

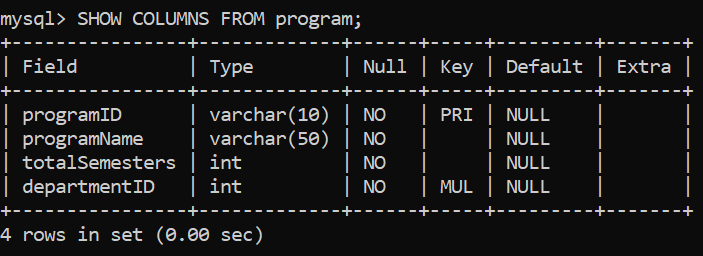
**A black screen with white text

AI-generated content may be incorrect.**

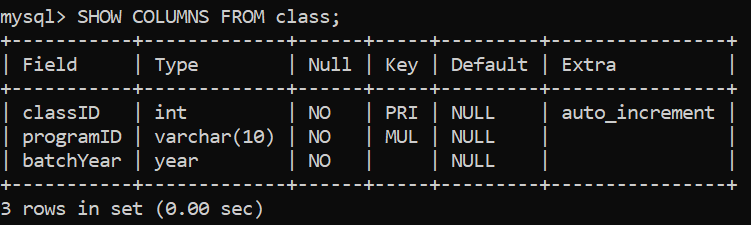
1. **Department**

****

1. **Program**

****

1. **Class**

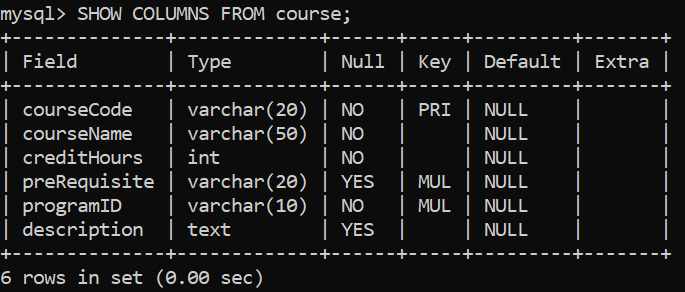
****

1. **Semester**

**A black screen with white text

AI-generated content may be incorrect.**

1. **Course**

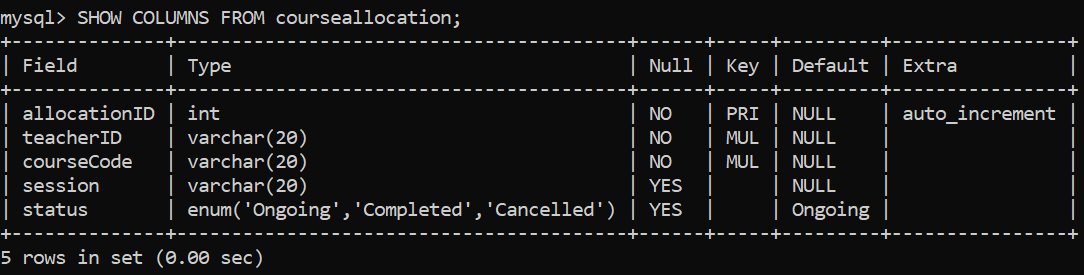
****

1. **SemesterDetails**

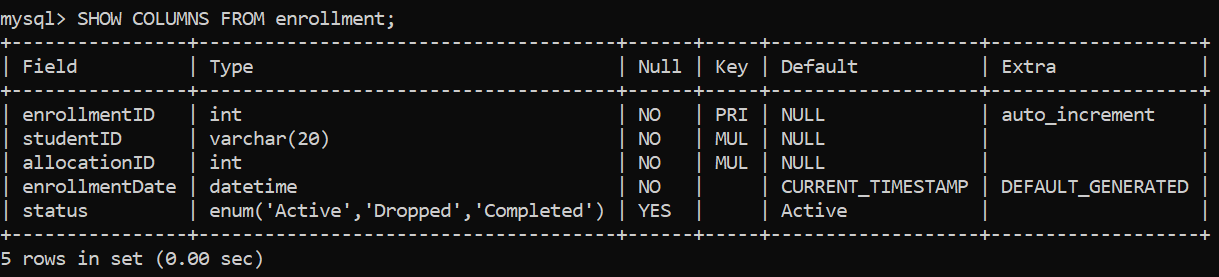
**A black screen with white text

AI-generated content may be incorrect.**

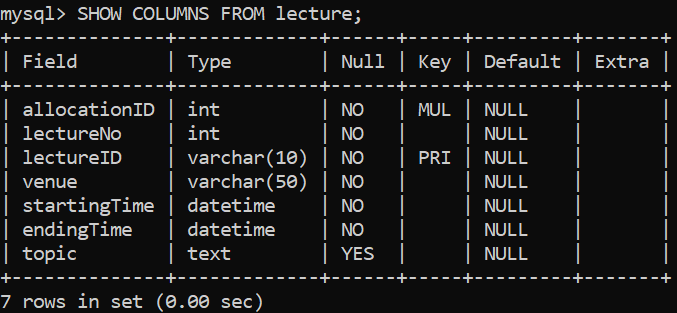
1. **CourseAllocation**

****

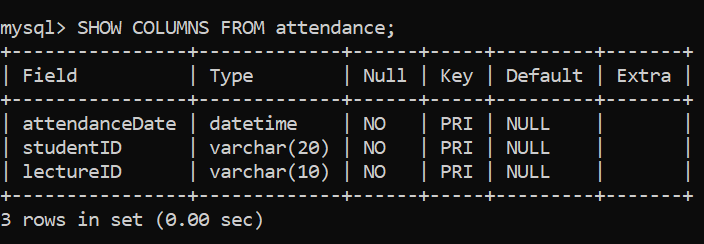
1. **Enrollment**

****

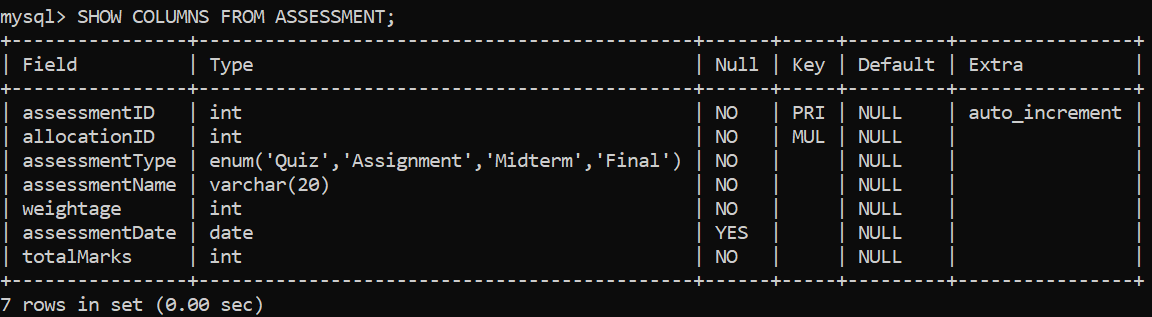
1. **Lecture**

****

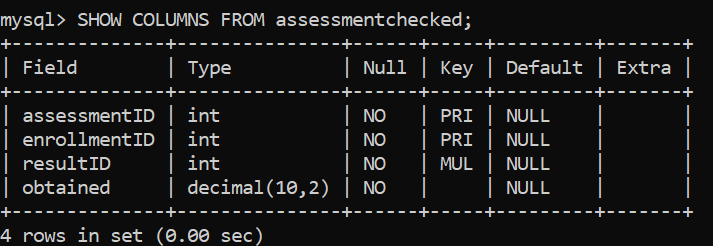
1. **Attendance**

****

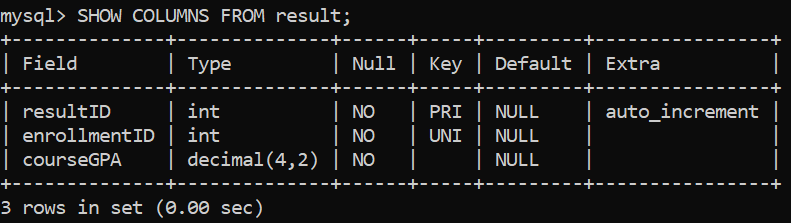
1. **Assessment**

****

1. **AssessmentChecked**

****

1. **Result**

****

1. **Transcipt**

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **Reviews**

**A screen shot of a computer program

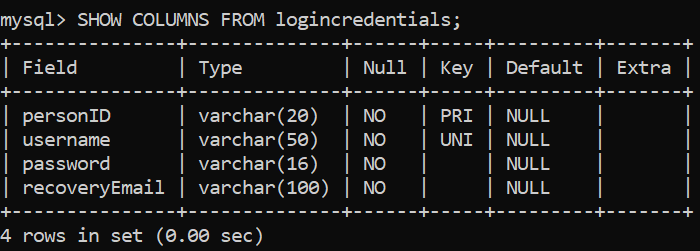
AI-generated content may be incorrect.**

1. **Alumni**

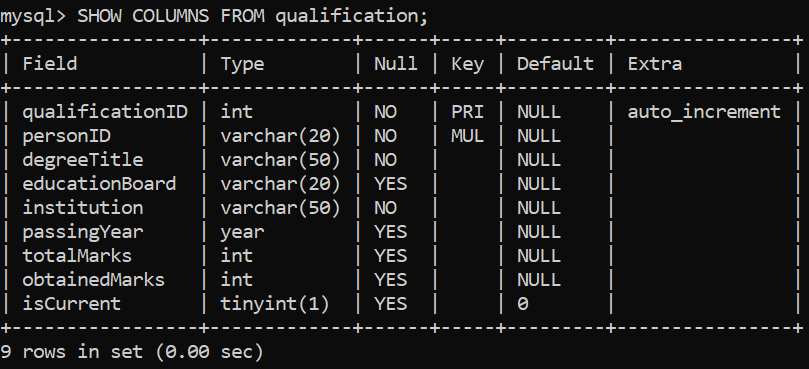
**A black screen with white text

AI-generated content may be incorrect.**

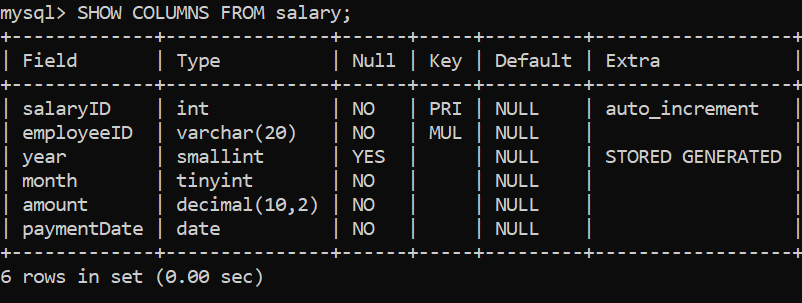
1. **LoginCredentials**

****

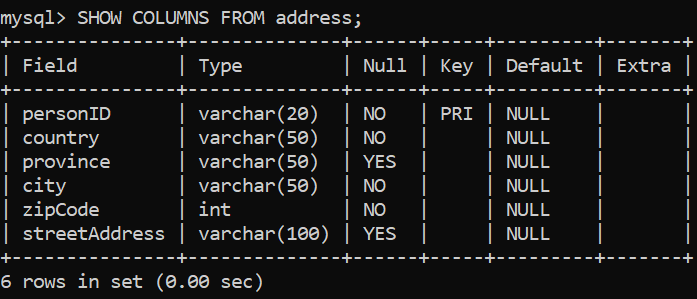
1. **Qualifications**



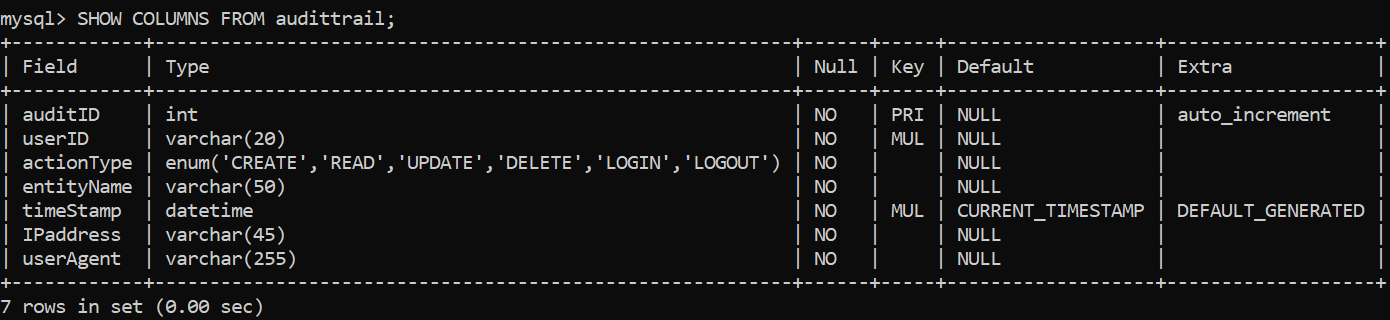
1. **Salary**

****

1. **Address**

****

1. **AuditTrail**

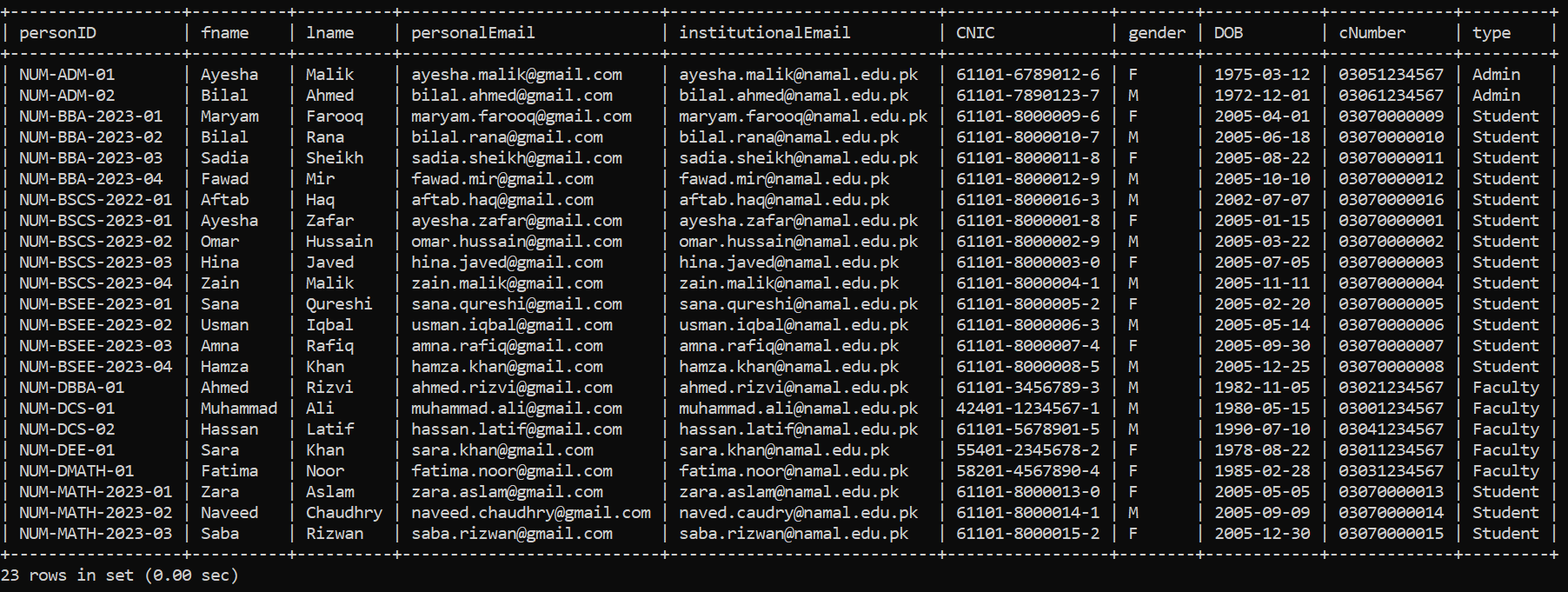
****

* 1. **DATA SAMPLES INSIDE TABLES:***.*

Below are the queries used to extract data from each table, followed by the screenshots of sample data from each table that was retrieved using these queries.

|  |
| --- |
| **Queries:**  SELECT \* FROM person;  SELECT \* FROM admin;  SELECT \* FROM faculty;  SELECT \* FROM student;  SELECT \* FROM department;  SELECT \* FROM program;  SELECT \* FROM class;  SELECT \* FROM semester;  SELECT \* FROM course;  SELECT \* FROM semesterDetail;  SELECT \* FROM courseAllocation;  SELECT \* FROM enrollment;  SELECT lectureID, venue, startingTime, endingTime, topic FROM lecture limit 20;  SELECT \* FROM attendance limit 20;  SELECT \* FROM assessment;  SELECT \* FROM assessmentChecked limit 20;  SELECT \* FROM result limit 20;  SELECT \* FROM transcript;  SELECT \* FROM reviews;  SELECT \* FROM alumni;  SELECT \* FROM LoginCredentials;  SELECT personID, degreeTitle, educationBoard, institution, passingYear, totalMarks, obtainedMarks, Iscurrent from qualification LIMIT 20;  SELECT \* FROM salary limit 20;  SELECT \* FROM address;  SELECT userID,actionType,entityName,timeStamp,IPaddress,userAgent FROM auditTrail limit 20; |

1. **Person**

****

1. **Admin**

*A screen shot of a computer code

AI-generated content may be incorrect.*

1. **Faculty**

**A computer screen shot of a black background

AI-generated content may be incorrect.**

1. **Student**

**A black screen with white text

AI-generated content may be incorrect.**

1. **Department**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

1. **Program**

**A black screen with white text

AI-generated content may be incorrect.**

1. **Class**

**A screen shot of a computer

AI-generated content may be incorrect.**

1. **Semester**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

1. **Course**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

1. **SemesterDetails**

**A black screen with white text

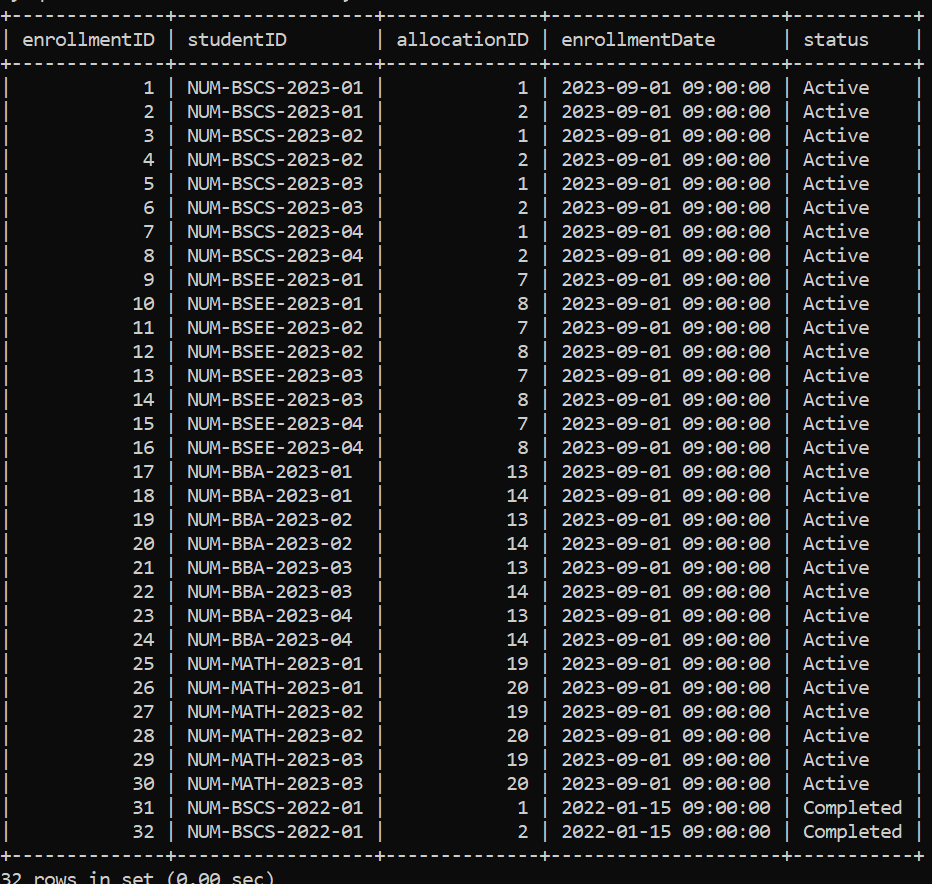
AI-generated content may be incorrect.**

1. **CourseAllocation**

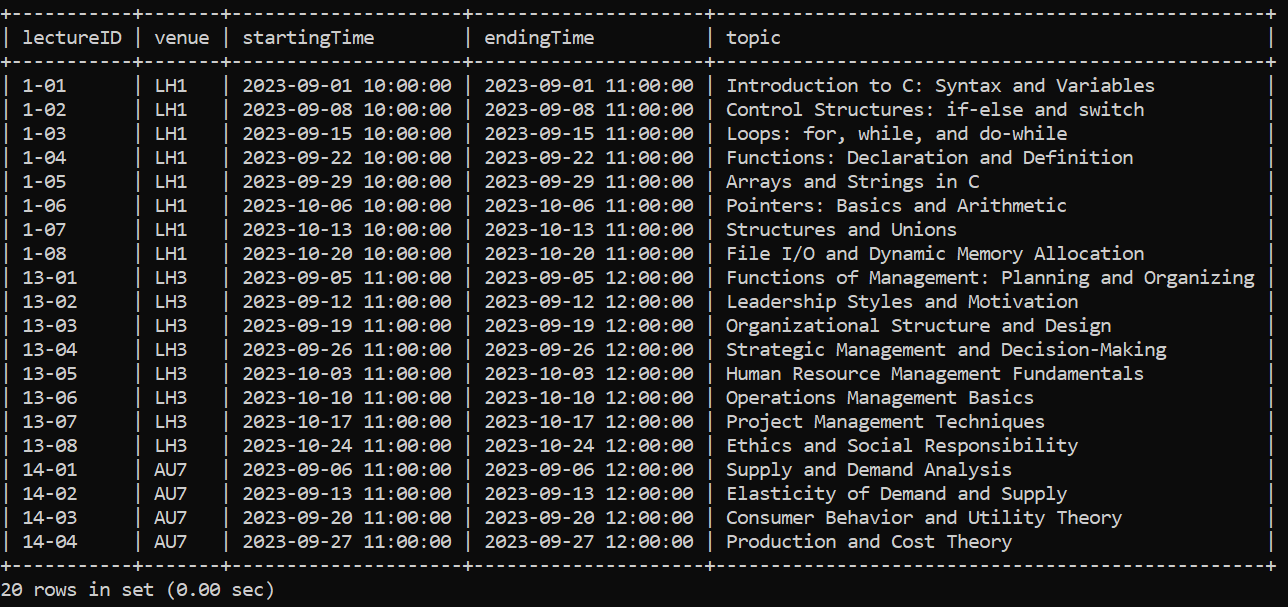
**A screen shot of a computer

AI-generated content may be incorrect.**

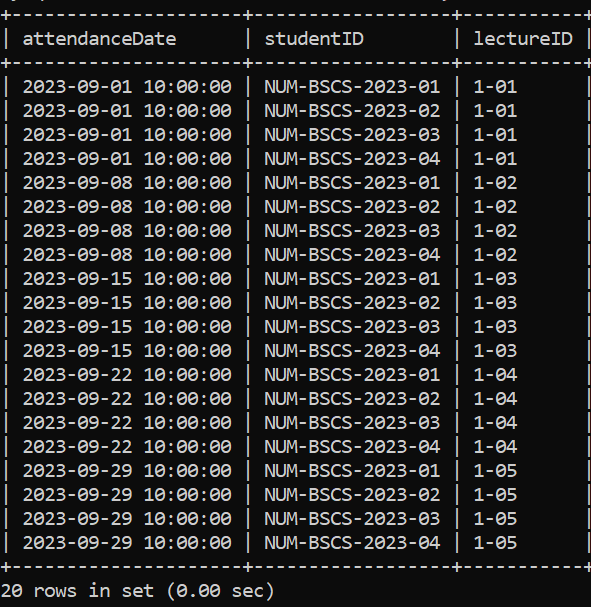
1. **Enrollment**

****

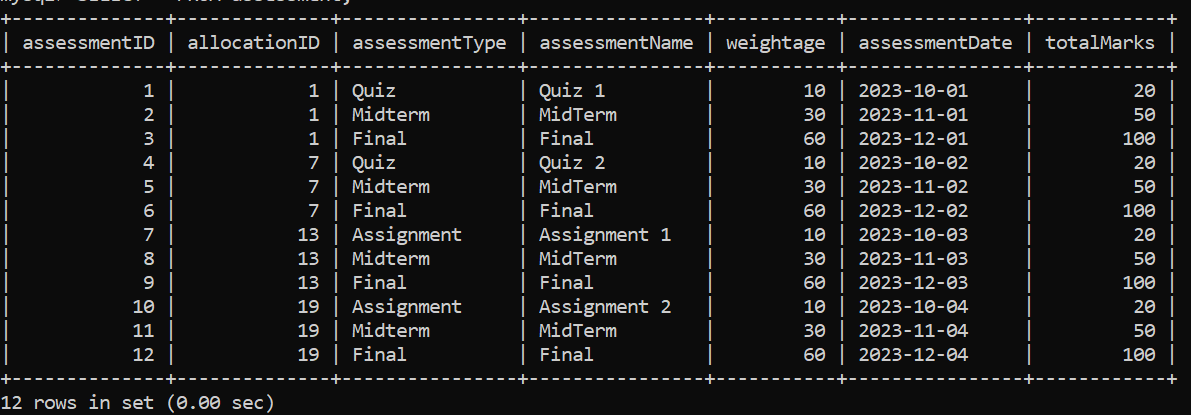
1. **Lecture**

****

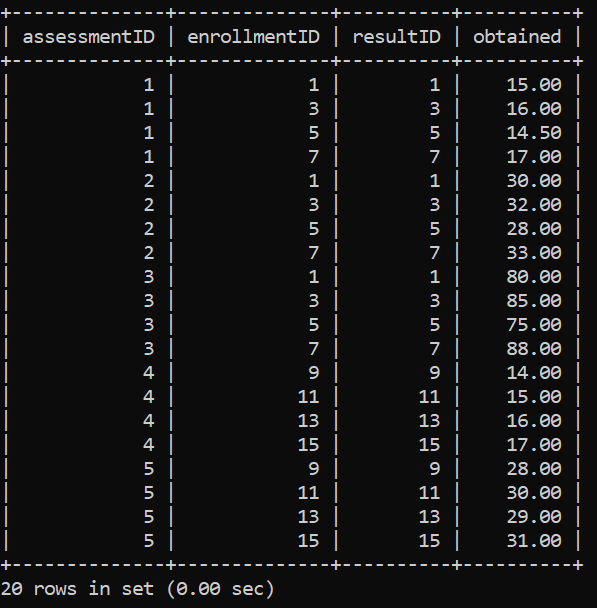
1. **Attendance**

****

1. **Assessment**

****

1. **AssessmentChecked**

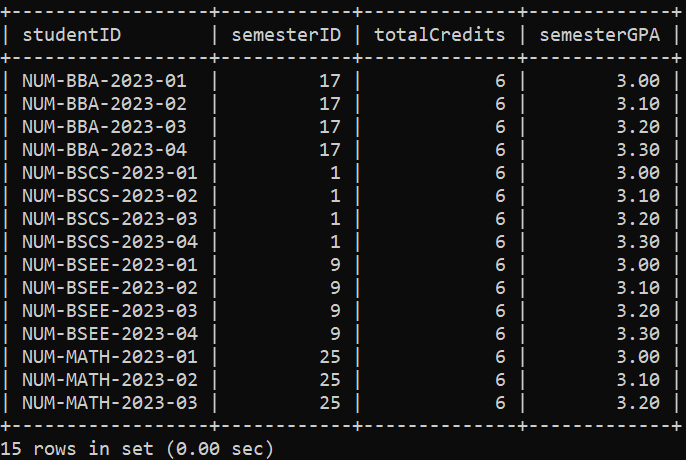
****

1. **Result**

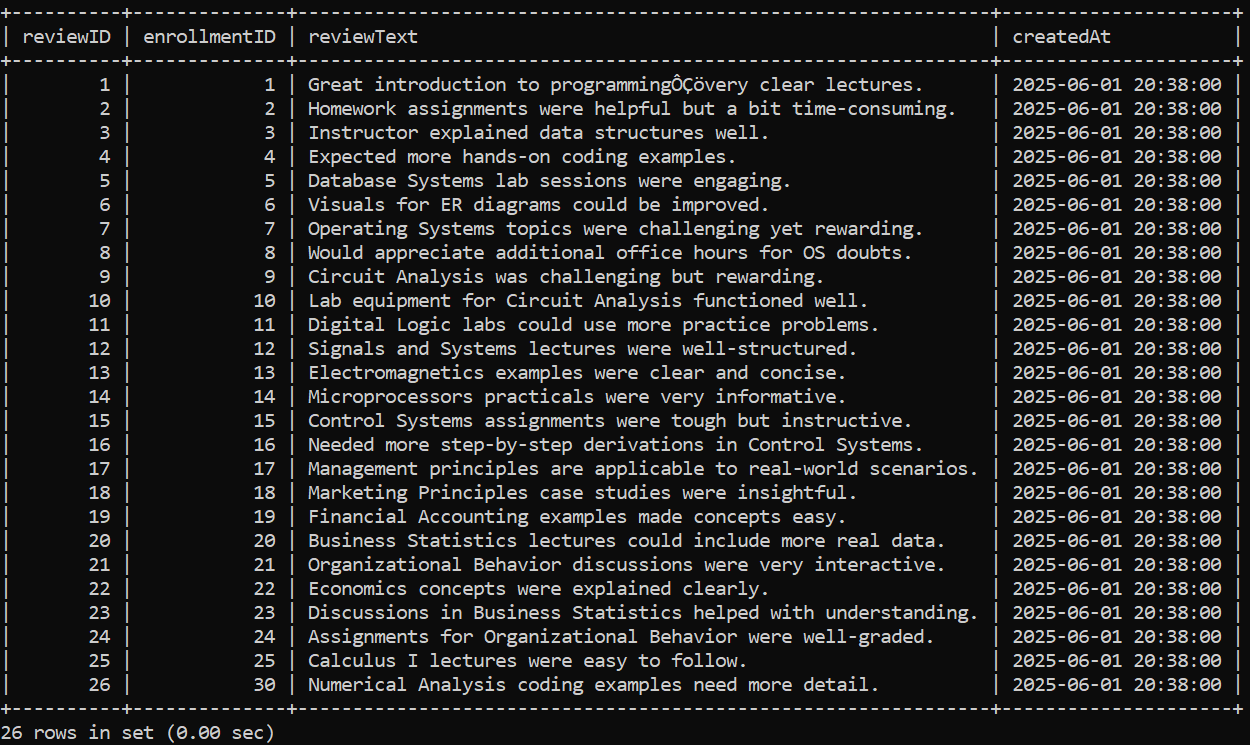
**A black and white screen with numbers

AI-generated content may be incorrect.**

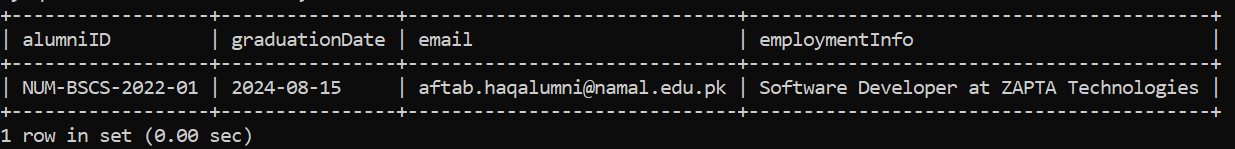
1. **Transcipt**

****

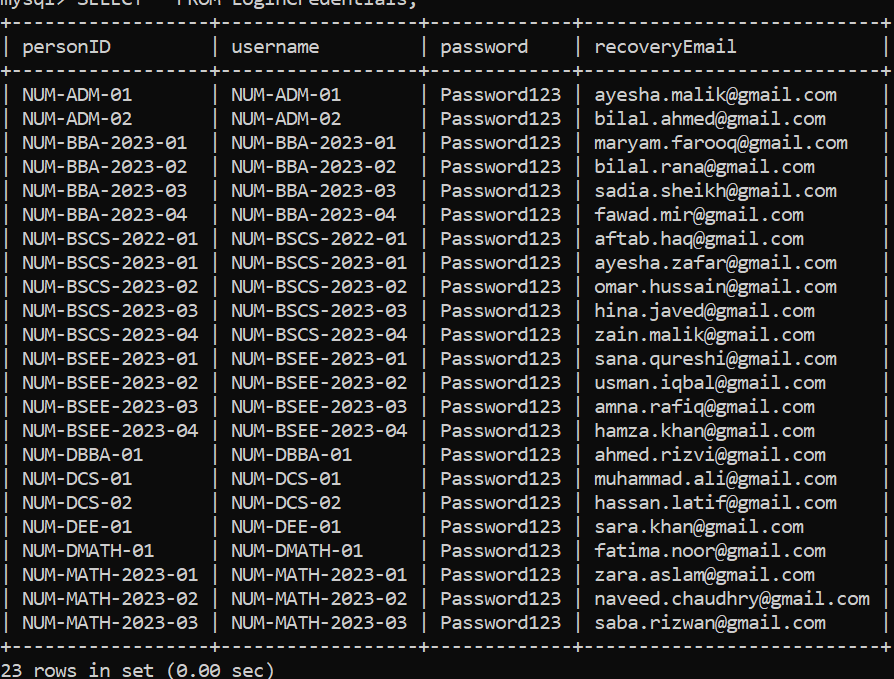
1. **Reviews**

****

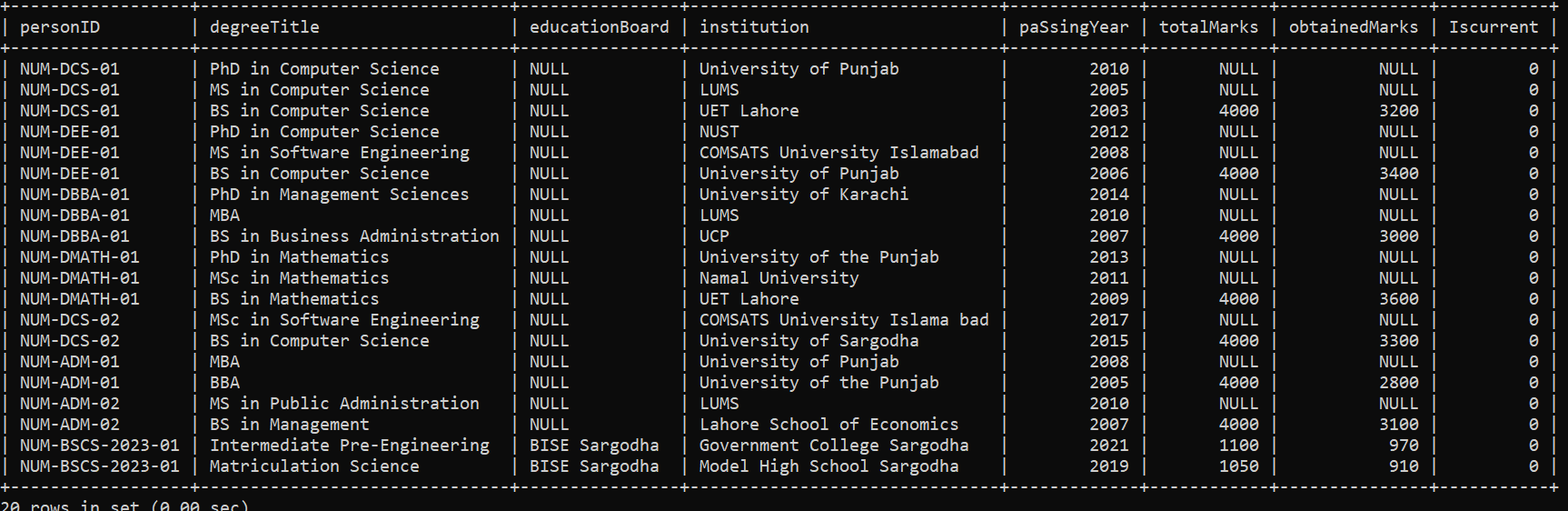
1. **Alumni**

****

1. **LoginCredentials**

****

1. **Qualifications**

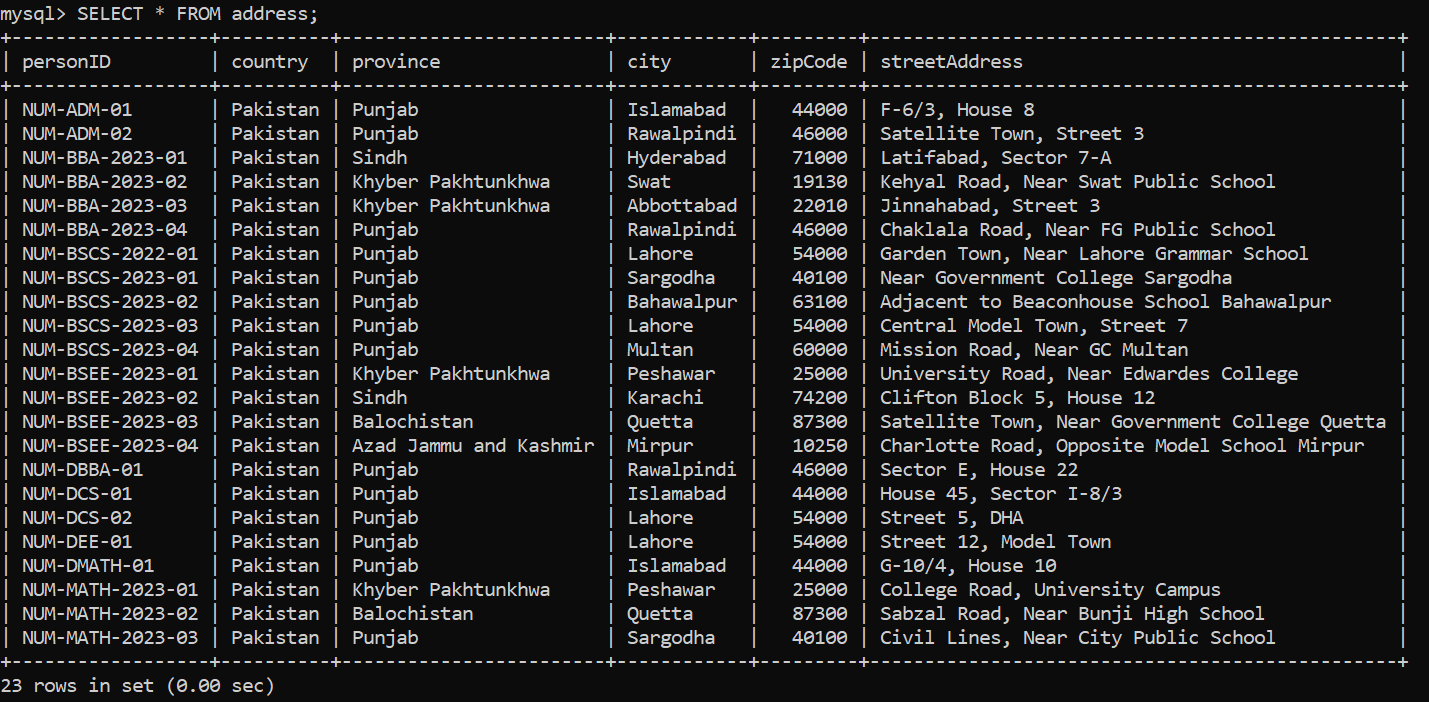


1. **Salary**

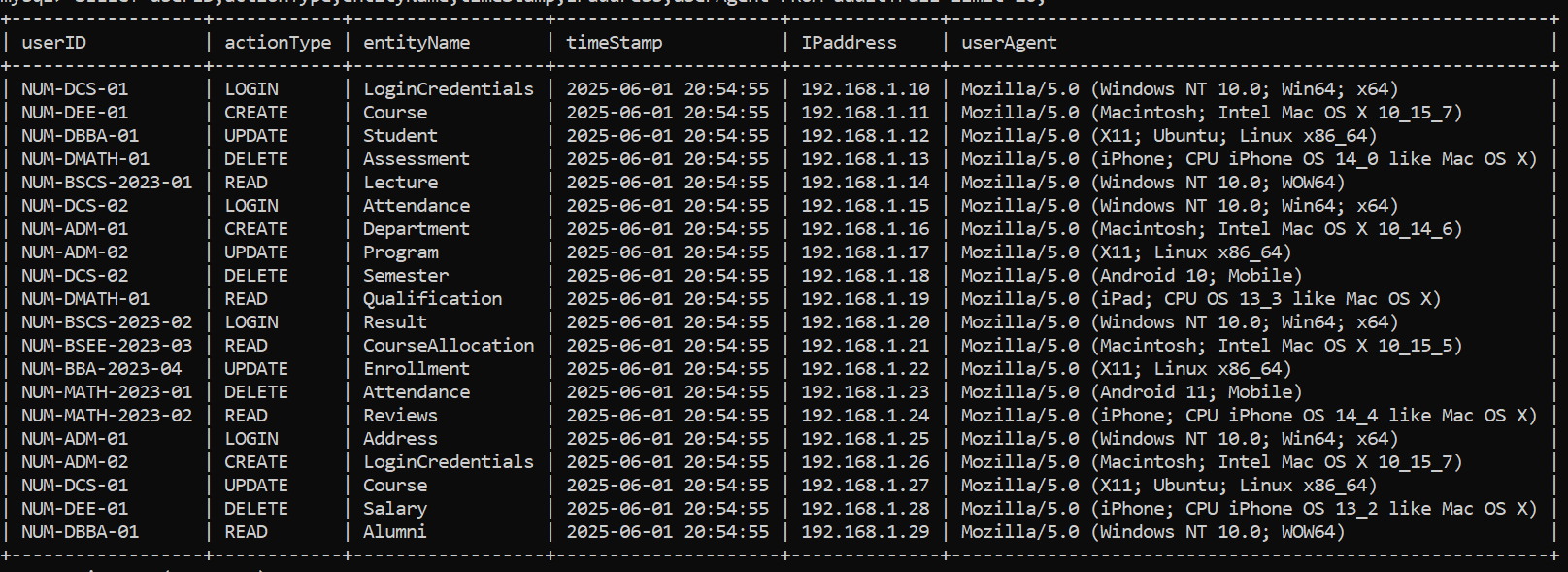
**A black screen with numbers and lines

AI-generated content may be incorrect.**

1. **Address**

****

1. **AuditTrail**

****

* 1. **QUERIES RESULTS:**

|  |
| --- |
| 1. Query:   CREATE VIEW AssessmentResultView AS  SELECT  ac.assessmentID,  a.assessmentName,  a.weightage,  a.totalMarks AS maxMarks,  ac.obtained AS marksObtained,  ac.enrollmentID,  s.studentID,  CONCAT(p.fname, ' ', p.lname) AS studentName,  ca.courseCode,  c.courseName,  ca.session AS courseSession,  r.courseGPA AS finalGPA  FROM AssessmentChecked ac  JOIN Assessment a  ON ac.assessmentID = a.assessmentID  JOIN Enrollment e  ON ac.enrollmentID = e.enrollmentID  JOIN Student s  ON e.studentID = s.studentID  JOIN Person p  ON s.studentID = p.personID  JOIN CourseAllocation ca  ON a.allocationID = ca.allocationID  JOIN Course c  ON ca.courseCode = c.courseCode  LEFT JOIN Result r  ON ac.enrollmentID = r.enrollmentID;  SELECT StudentID, StudentName, assessmentName, Weightage, maxMarks, Marksobtained, courseName, finalGPA FROM ASSESSMENTResultView limit 15; |
| Output: |
| 1. Query:   SELECT s.studentID, CONCAT(p.fname, ' ', p.lname) AS studentName, ca.allocationID, ca.courseCode, c.courseName, ca.session AS courseSession, COUNT(DISTINCT l.lectureID) AS totalLecturesHeld, COUNT(a.attendanceDate) AS lecturesAttended FROM Attendance a  JOIN Lecture l ON a.lectureID = l.lectureID JOIN Enrollment e ON a.studentID = e.studentID AND l.allocationID = e.allocationID JOIN Student s ON e.studentID = s.studentID JOIN Person p ON s.studentID = p.personID JOIN CourseAllocation ca ON l.allocationID = ca.allocationID JOIN Course c ON ca.courseCode = c.courseCode GROUP BY ca.allocationID, s.studentID; |
| Output: |
| 1. Query:   SELECT ca.allocationID, ca.courseCode, c.courseName, ca.session AS courseSession, p.personID AS studentID, CONCAT(p.fname, ' ', p.lname) AS studentName, e.enrollmentDate, e.status AS enrollmentStatus FROM Enrollment e JOIN CourseAllocation ca ON e.allocationID = ca.allocationID JOIN Course c ON ca.courseCode = c.courseCode JOIN Student s ON e.studentID = s.studentID JOIN Person p ON s.studentID = p.personID WHERE s.studentID='NUM-BSCS-2023-01'; |
| Output: |
| 1. Query:   SELECT CONCAT(p.fname,' ',p.lname) AS studentName,r.courseGPA FROM Result r JOIN Enrollment e ON r.enrollmentID=e.enrollmentID JOIN CourseAllocation ca ON e.allocationID=ca.allocationID JOIN Person p ON e.studentID=p.personID WHERE ca.courseCode='CS102' AND r.courseGPA>(SELECT AVG(r2.courseGPA) FROM Result r2 JOIN Enrollment e2 ON r2.enrollmentID=e2.enrollmentID JOIN CourseAllocation ca2 ON e2.allocationID=ca2.allocationID WHERE ca2.courseCode='CS102'); |
| Output: |
| 1. Query:   SELECT CONCAT(p.fname,' ',p.lname) AS Instructor,ca.session,COUNT(DISTINCT ca.courseCode) AS totalCourses FROM Faculty f JOIN CourseAllocation ca ON f.employeeID=ca.teacherID JOIN Person p ON f.employeeID=p.personID GROUP BY Instructor ,ca.session HAVING totalCourses>1; |
| Output: |
| 1. Query:   SELECT c.courseName,AVG(ac.obtained) AS averageMarks FROM AssessmentChecked ac JOIN Assessment a ON ac.assessmentID=a.assessmentID JOIN CourseAllocation ca ON a.allocationID=ca.allocationID JOIN Course c ON ca.courseCode=c.courseCode GROUP BY c.courseName ORDER BY averageMarks DESC; |
| Output: |
| 1. Query:   SELECT t.semesterID, AVG(t.semesterGPA) AS averageGPA FROM Transcript t WHERE t.semesterID IN ( SELECT semesterID FROM SemesterDetails GROUP BY semesterID HAVING COUNT(DISTINCT courseCode) >= 2 ) GROUP BY t.semesterID; |
| Output: |
| 1. Query:   SELECT c.courseName,ca.session,CONCAT(p.fname,' ',p.lname) AS studentName,r.courseGPA,t.topGPA,ROUND((r.courseGPA/t.topGPA)\*100,2) AS relativePercentage,(SELECT COUNT(\*)+1 FROM Result r2 JOIN Enrollment e2 ON r2.enrollmentID=e2.enrollmentID JOIN CourseAllocation ca2 ON e2.allocationID=ca2.allocationID WHERE ca2.courseCode=ca.courseCode AND r2.courseGPA>r.courseGPA) AS courseRank FROM Result r JOIN Enrollment e ON r.enrollmentID=e.enrollmentID JOIN CourseAllocation ca ON e.allocationID=ca.allocationID JOIN Course c ON ca.courseCode=c.courseCode JOIN Student s ON e.studentID=s.studentID JOIN Person p ON s.studentID=p.personID JOIN (SELECT ca.courseCode,MAX(r.courseGPA) AS topGPA FROM Result r JOIN Enrollment e ON r.enrollmentID=e.enrollmentID JOIN CourseAllocation ca ON e.allocationID=ca.allocationID GROUP BY ca.courseCode) AS t ON ca.courseCode=t.courseCode ORDER BY c.courseCode,courseRank; |
| Output: |
| 1. Query:   SELECT d.departmentName,COUNT(DISTINCT s.studentID) AS totalStudents FROM Department d JOIN Program p ON d.departmentID=p.departmentID JOIN Student s ON p.programID=s.programID GROUP BY d.departmentName HAVING totalStudents=(SELECT MAX(student\_count) FROM(SELECT COUNT(DISTINCT s2.studentID) AS student\_count FROM Program p2 JOIN Student s2 ON p2.programID=s2.programID GROUP BY p2.departmentID) AS counts); |
| Output: |
| 1. Query:   SELECT DISTINCT p.programName,s.semesterNo AS 'Ongoing Semesters', sd.semesterID FROM Program p JOIN Semester s ON s.programID=p.programID JOIN semesterDetails sd ON s.semesterID=sd.semesterID JOIN Course c on sd.courseCode=c.courseCode JOIN courseAllocation ca ON ca.courseCode=c.courseCode; |
| Output: |

# : Interface Design

## 5.1. LANGUAGE/FRAMEWORK:

### **5.1.1. Technology Stack Overview**

The Learning Management System was developed using Django 5.2.1, a high-level Python web framework that follows the Model-View-Template (MVT) architectural pattern. Django was specifically chosen for this educational platform due to its robust feature set, security considerations, and suitability for complex database-driven applications.

### **5.1.2. Primary Framework: Django 5.2.1**

**Django** is a free and open-source web framework written in Python that encourages rapid development and clean, pragmatic design. It was created to handle the intensive requirements of newsroom environments and has evolved to become one of the most popular web frameworks for building scalable web applications.

### **5.1.3. Core Architecture - Model-View-Template (MVT):**

1. **Model Layer**: Handles data logic and database interactions through Django's Object-Relational Mapping (ORM)
2. **Form layer:** Data validation and form handling (crucial layer often overlooked)
3. **View Layer**: Contains the business logic and processes HTTP requests
4. **Template Layer**: Manages the presentation logic and user interface rendering
5. **URL Layer** - URL routing and mapping

**Built-in Administrative Interface**

Django provides an automatic admin interface that perfectly aligns with our LMS requirements. This feature was crucial for our three-tier user system (Admin, Faculty, Student) as it provides:

* Automatic CRUD operations for all database models
* User management with role-based permissions
* Content management capabilities
* Audit trail functionality through Django's built-in logging

### **5.1.4. Key Advantages and Rationale for Django Selection:**

**Built-in Administrative Interface**

Django provides an automatic admin interface that perfectly aligns with our LMS requirements. This feature was crucial for our three-tier user system (Admin, Faculty, Student) as it provides:

* Automatic CRUD operations for all database models
* User management with role-based permissions
* Content management capabilities
* Audit trail functionality through Django's built-in logging

**Object-Relational Mapping (ORM)**

Django's ORM eliminates the need for writing raw SQL queries while maintaining database independence. This was particularly beneficial for our complex relational schema with 25+ tables:

* Automatic SQL generation from Python code
* Database migration management
* Query optimization and caching
* Support for complex relationships (One-to-One, One-to-Many, Many-to-Many)

**Security Framework**

Django includes built-in protection against common web vulnerabilities, which is essential for an educational platform handling sensitive student data:

* **CSRF Protection**: Automatic Cross-Site Request Forgery protection
* **SQL Injection Prevention**: ORM prevents SQL injection attacks
* **XSS Protection**: Template system automatically escapes variables
* **Clickjacking Protection**: X-Frame-Options middleware
* **HTTPS/SSL Support**: Built-in security middleware for production deployment

**User Authentication and Authorization System**

Django's authentication framework provides comprehensive user management capabilities:

* User registration and login functionality
* Session management
* Password hashing and validation
* Permission and group-based access control
* Custom user models support

**Scalability and Performance**

Django's architecture supports scaling from small applications to large enterprise systems:

* Database connection pooling
* Template and query caching
* Static file optimization
* Support for multiple databases
* Asynchronous view support (Django 3.1+)

**5.2. DATABASE CONNECTIVITY:**

**Database Architecture Overview**

The Learning Management System employs MySQL as the primary relational database management system, chosen for its reliability, performance, and compatibility with Django's ORM. The application supports dual-environment configuration to seamlessly operate in both development and production environments with different database connectors optimized for each scenario.

**Environment-Specific Database Configuration**

**Local MySQL Setup:**

# Development database configuration (settings.py)

if not os.environ.get('RAILWAY\_ENVIRONMENT'):

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'LMS',

'USER': 'root',

'PASSWORD': '@databaselab123',

'HOST': 'localhost',

'PORT': '3306',

'OPTIONS': {

'sql\_mode': 'traditional',

'charset': 'utf8mb4',

'use\_unicode': True,

},

}

}

**Development Environment Characteristics:**

* **Database Engine**: django.db.backends.mysql (Django's native MySQL backend)
* **Connection Library**: mysqlclient==2.2.7 (C-based MySQL adapter)
* **Performance**: Optimized for local development with minimal latency
* **Security**: Basic authentication suitable for local environment
* **Production Environment Configuration**

**Railway Cloud MySQL Setup:**

python

# Production database configuration (Railway deployment)

if os.environ.get('RAILWAY\_ENVIRONMENT'):

DATABASES = {

'default': {

'ENGINE': 'mysql.connector.django',

'NAME': 'railway',

'USER': 'root',

'PASSWORD': 'OfIdpzYBYZLTWhASnwvGOPqpKrsNGWVU',

'HOST': 'crossover.proxy.rlwy.net',

'PORT': 58556,

'OPTIONS': {

'autocommit': True,

'charset': 'utf8mb4',

'use\_unicode': True,

'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'",

'sql\_mode': 'traditional',

},

}

}

print("Using Railway MySQL database")

**Production Environment Characteristics:**

* **Database Engine**: mysql.connector.django (Pure Python MySQL connector)
* **Connection Library**: mysql-connector-python==9.1.0
* **Cloud Infrastructure**: Railway managed MySQL service
* **High Availability**: Built-in redundancy and automatic backups
* **Security**: Encrypted connections and managed authentication
* **Database Connection Libraries and Adapters**
* **Development: mysqlclient (C-based adapter)**

**Advantages:**

* **Performance**: C-based implementation provides superior performance
* **Memory Efficiency**: Lower memory footprint compared to pure Python solutions
* **Django Integration**: Native support in Django's MySQL backend
* **Thread Safety**: Excellent multi-threading support for concurrent requests*.*