## **LEARNER’S GUIDE**

## A PROJECT REPORT

*Submitted by*

# Akhila Tejaswi. Y [RA2211003011417]

# Khushi [RA2211003011450]

*Under the Guidance of*

# Dr. S Shanmugam

Designation, Department of Computing Technologies

*in partial fulfillment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

# in

**COMPUTER SCIENCE AND ENGINEERING**



**DEPARTMENT OF COMPUTING TECHNOLOGIES**

# COLLEGE OF ENGINEERING AND TECHNOLOGY

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR– 603 203

**MAY 2024**



# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

# KATTANKULATHUR–603 203

## **BONAFIDE CERTIFICATE**

**RA2211003011417** , **RA2211003011450** Certified to be the bonafide work done by **Akhila Tejaswi. Y, Khushi** of II year/IV sem B.Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

**Date: 03-05-24**

### Faculty in Charge

Dr. Shanmugam S

Associate Professor

Department of Computing Technologies

SRMSIT -KTR

### HEAD OF THE DEPARTMENT

Dr. M. Pushpalatha

Head of the Department

Department of Computing Technologies

SRMIST - KTR

### 

**ABSTRACT**

Learner’s Guide have revolutionized the way people learn and search for courses. With the help of such systems, students and instructors can find and enroll in courses that align with their interests and career paths from the comfort of their own homes. These systems simplify the process of applying for courses for students and enrolling students for instructors by utilizing the Internet. Enrolling in online courses has become more accessible, faster, and simpler in recent years, thanks to the availability of a wide range of subject categories. Course seekers can easily browse the courses in the dashboard and find courses that meet their needs. Students can view their courses anytime by logging into the web application and accessing their student homepage.

Online enrollment services and Learning Management Systems make it easier and more cost-effective for instructors to host their courses. Our proposed system allows students and instructors to register and have different roles. Students can apply to courses matching their interests and instructors can add courses and track profits in the dashboard.

The online course website uses Bootstrap, jQuery, JavaScript, and HTML5 for the frontend, and Java and Spring boot MVC for the backend, with MySQL as the database. It has student and instructor roles, with students able to view and purchase courses and instructors able to create, edit, and disable courses. The platform offers flexibility to students and wider audience reach to instructors.In conclusion, Online Learning Management Systems have made education more accessible and easier for both students and instructors. With the development of technology and the internet, LMS has become an essential tool for individuals who wish to improve their knowledge and skills. The proposed LMS system offers a user-friendly interface, easy enrollment process, and flexibility that make it a perfect tool for students and instructors alike.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
|  | **ABSTRACT** | **iii** |

**Problem Statement**

|  |  |  |
| --- | --- | --- |
| **Chapter No** | **Chapter Name** | **Page No** |
|  | Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project |  |
|  | Design of Relational Schemas, Creation of Database Tables for the project. |  |
|  | Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors. |  |
|  | Analyzing the pitfalls, identifying the dependencies, and applying normalizations |  |
|  | Implementation of concurrency control and recovery mechanisms |  |
|  | Code for the project |  |
|  | Result and Discussion (Screen shots of the implementation with front end. |  |
|  | Attach the Real Time project certificate / Online course certificate |  |

### Project Description

Online Learning Management System is a web application, the platform provides an instructor with a facility to upload his/course in the portal and for students to enroll/purchase the course he/she desires after the registration. The web application will be a platform for students to view or purchase a variety of courses from a wide range of instructors. The application UI/UX is user- friendly, and the application consists of a login screen where the user can also register if not already registered.

##### Roles and Features:

The following are the roles supported in the application.

1. Instructor
2. Student

##### Instructor

Any user can register and can become an instructor. The instructor role would be to create a course and attach a file of their course. The instructor can later update the course details and can disable the course if required.

##### Student

Students can sign up for the web application and after logging into the application, he/she will be provided with a

homepage with the courses he/she purchased. The student can be directed to the courses page and browse through a variety of courses by all instructors.

### Competitive Information

Various online learning management portals are like our platform. The following are a few of the competitors of this application.

* Udemy
* Coursera
* Khan academy

### Relationship to Other Applications/Projects

This is an online learning management web application, which helps students and instructors to enroll or post courses on a web-based platform. The web application uses Spring boot and web technologies like HTML, CSS3, jQuery, and JavaScript.

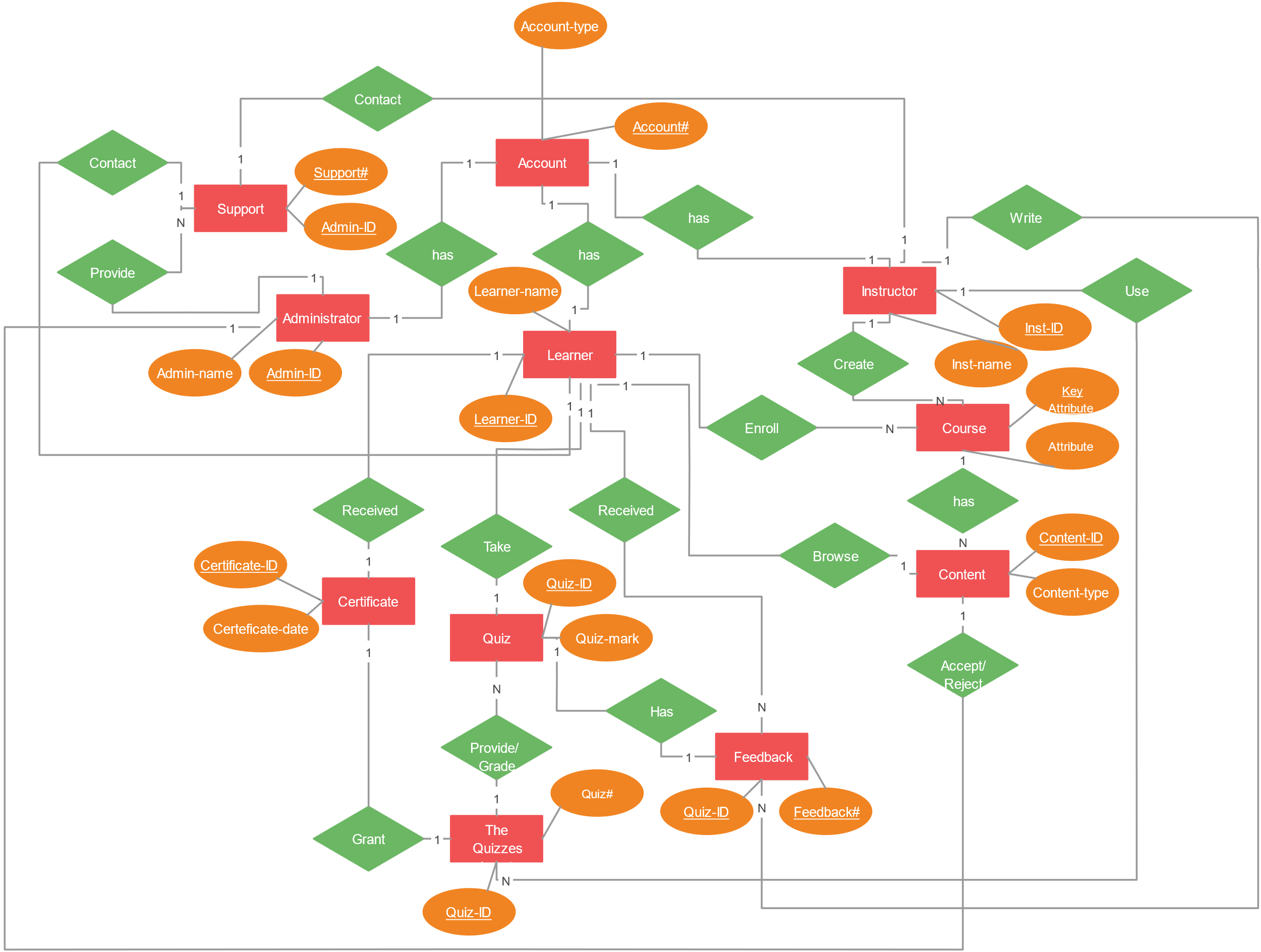
### Assumptions and Dependencies

Only authenticated users who have registered can access this website, and if they have not already, they can do so right away. Instructors will be able to create and sell their courses. Students will be able to enroll in the courses using any credit and debit card.

### Future Enhancements

Currently, the application is not horizontally scalable, eventually, when the application gains more traffic, the architecture should be able to scale horizontally. CDN can be integrated to render pages faster regardless of the location of the user. Multi Factor authentication can be integrated to increase login security.

**ENTITY RELATIONSHIP DIAGRAM**



**CODE :**

CREATE DATABASE LEARNER\_GUIDE;

USE LEARNER\_GUIDE;

-- Create the administrator table

CREATE TABLE ADMINISTRATOR (

    admin\_id INT(5) PRIMARY KEY,

    admin\_name VARCHAR(100) NOT NULL,

    admin\_email VARCHAR(100) NOT NULL

);

CREATE TABLE INSTRUCTOR(

    inst\_id INT(5) PRIMARY KEY,

    inst\_name VARCHAR(100) NOT NULL,

    inst\_email VARCHAR(100) NOT NULL,

    inst\_phno VARCHAR(15) NOT NULL

);

-- Create the learner table

CREATE TABLE LEARNER (

    learner\_id INT(6) PRIMARY KEY,

    learner\_name VARCHAR(100) NOT NULL,

    learner\_phno INT(10) NOT NULL,

    learner\_email VARCHAR(100) NOT NULL

);

CREATE TABLE COURSE (

    course\_id INT(3) PRIMARY KEY,

    course\_name VARCHAR(100) NOT NULL,

    FOREIGN KEY (inst\_id) REFERENCES INSTRUCTOR(inst\_id)

);

-- Create the quiz table

CREATE TABLE QUIZ(

    quiz\_id INT(2) PRIMARY KEY,

    quiz\_marks DECIMAL(5,2) NOT NULL

);

-- Create the certificate table

CREATE TABLE IF NOT EXISTS certificate (

    certificate\_id INT(1) PRIMARY KEY,

    certificate\_date DATE NOT NULL,

);

-- Inserting values into the administrator table

INSERT INTO ADMINISTRATOR (admin\_id, admin\_name, admin\_email) VALUES

(10001, 'Admin1', 'admin1@gmail.com'),

(10012, 'Admin2','admin2@gmail.com'),

(10023, 'Admin3','admin3@gmail.com'),

(13004, 'Admin4','admin4@gmail.com'),

(10405, 'Admin5','admin5@gmail.com');

INSERT INTO INSTRUCTOR (inst\_id, inst\_name, inst\_email,inst\_phno) VALUES

(10201, 'Inst1', 'inst1@gmail.com', 9881188897),

(10202, 'Inst2','inst2@gmail.com', 9888338891),

(10203, 'Inst3','inst3@gmail.com', 9844450011),

(10204, 'Inst4','inst4@gmail.com', 9866778897),

(10205, 'Inst5','inst5@gmail.com', 9111456797);

-- Inserting values into the learner table

INSERT INTO learner (learner\_id, learner\_name, learner\_email, learner\_phno) VALUES

(573091, 'John Doe', 'jodo1@gmail.com', 9881188897),

(573092, 'Jane Smith', 'jasm2@gmail.com', 9881188897),

(573093, 'Alice Johnson', 'aljh3@gmail.com', 9881188897),

(573094, 'Bob Williams', 'bowi4@gmail.com', 9881188897),

(573095, 'Emma Brown', 'embr5@gmail.com', 9881188897);

-- Inserting values into the course table

INSERT INTO course (course\_id, course\_name) VALUES

(211, 'Computer Science'),

(212, 'Mathematics'),

(213, 'Physics'),

(214, 'English'),

(215, 'History');

-- Inserting values into the quiz table

INSERT INTO quiz (quiz\_id, quiz\_marks) VALUES

(12, 90),

(13, 78),

(14, 92.5),

(15, 85);

-- Inserting values into the certificate table

INSERT INTO certificate (certificate\_id, certificate\_date) VALUES

(2, '2023-02-20'),

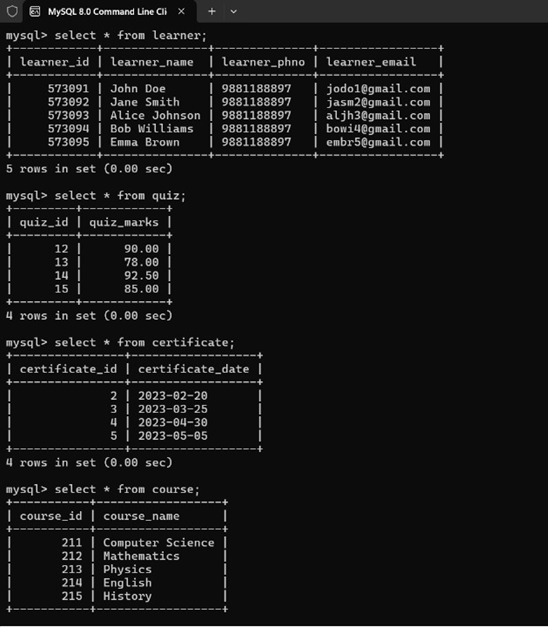
(3, '2023-03-25'),

(4, '2023-04-30'),

(5, '2023-05-05');

**OUTPUT –**





**CONSTRAINTS**

**Constraints for Learners Table**

ALTER TABLE Learners

ADD CONSTRAINT PK\_Learners PRIMARY KEY (learner\_id),

ADD CONSTRAINT FK\_Course\_Learner FOREIGN KEY (course\_id) REFERENCES Courses(course\_id);

**Constraints for Instructors Table**

ALTER TABLE Instructors

ADD CONSTRAINT PK\_Instructors PRIMARY KEY (instructor\_id);

**Constraints for Administrator Table**

ALTER TABLE Administrator

ADD CONSTRAINT PK\_Administrator PRIMARY KEY (admin\_id);

**Constraints for Courses Table**

ALTER TABLE Courses

ADD CONSTRAINT PK\_Courses PRIMARY KEY (course\_id),

ADD CONSTRAINT FK\_Instructor\_Courses FOREIGN KEY (instructor\_id) REFERENCES Instructors(instructor\_id);

**Constraints for Quiz Table**

ALTER TABLE Quiz

ADD CONSTRAINT PK\_Quiz PRIMARY KEY (quiz\_id),

ADD CONSTRAINT FK\_Course\_Quiz FOREIGN KEY (course\_id) REFERENCES Courses(course\_id);

**Constraints for Feedback Table**

ALTER TABLE Feedback

ADD CONSTRAINT PK\_Feedback PRIMARY KEY (feedback\_id),

ADD CONSTRAINT FK\_Learner\_Feedback FOREIGN KEY (learner\_id) REFERENCES Learners(learner\_id),

ADD CONSTRAINT FK\_Course\_Feedback FOREIGN KEY (course\_id) REFERENCES Courses(course\_id);

**Constraints** **for Certificate Table**

ALTER TABLE Certificate

ADD CONSTRAINT PK\_Certificate PRIMARY KEY (certificate\_id),

ADD CONSTRAINT FK\_Learner\_Certificate FOREIGN KEY (learner\_id) REFERENCES Learners(learner\_id),

ADD CONSTRAINT FK\_Course\_Certificate FOREIGN KEY (course\_id) REFERENCES Courses(course\_id);

**VIEW**

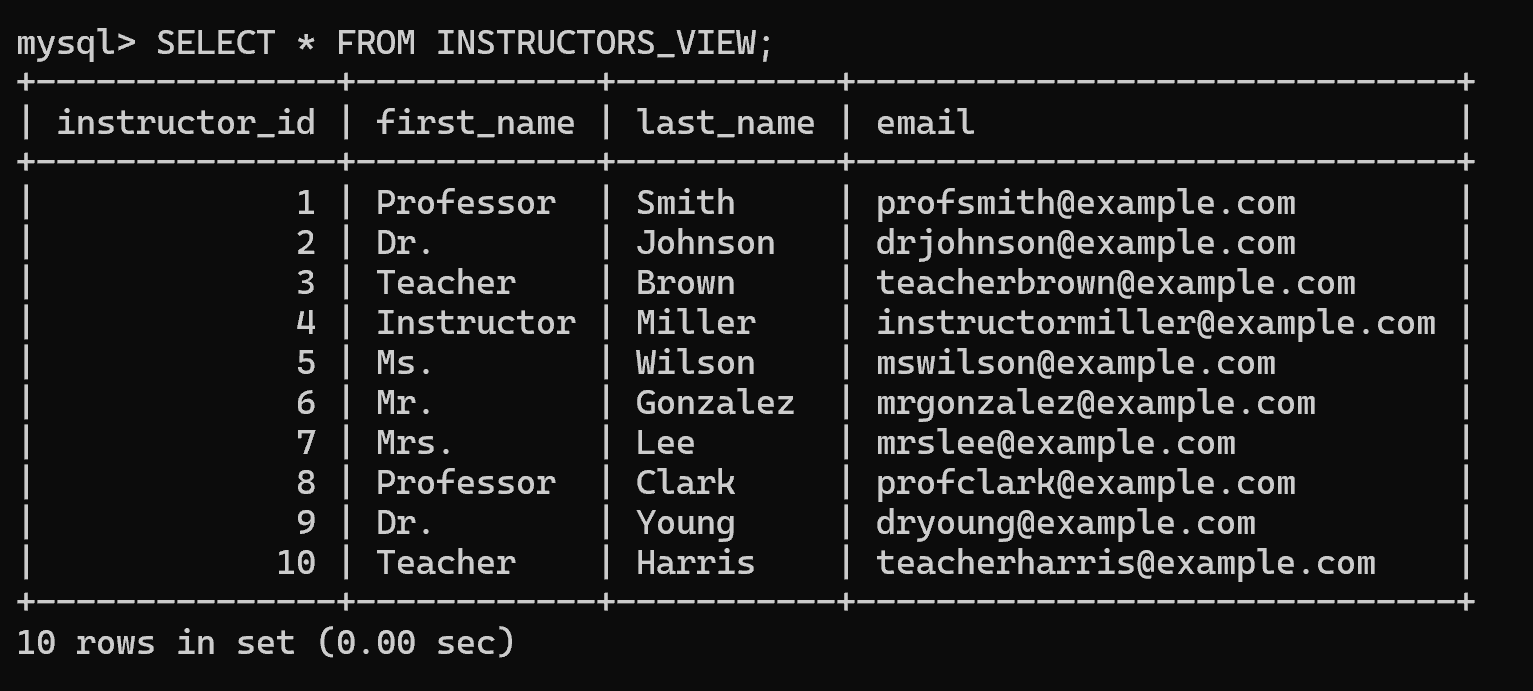
**1. Create a view for learners, instructors, administrator, courses, quiz, feedback and certificate.**

**View for Instructors table**

CREATE VIEW Instructors\_View AS

SELECT instructor\_id, first\_name, last\_name, email

FROM Instructors;



**View for Learners table**

CREATE VIEW Learners\_View AS

SELECT learner\_id, first\_name, last\_name, email, date\_of\_birth, enrollment\_date

FROM Learners;

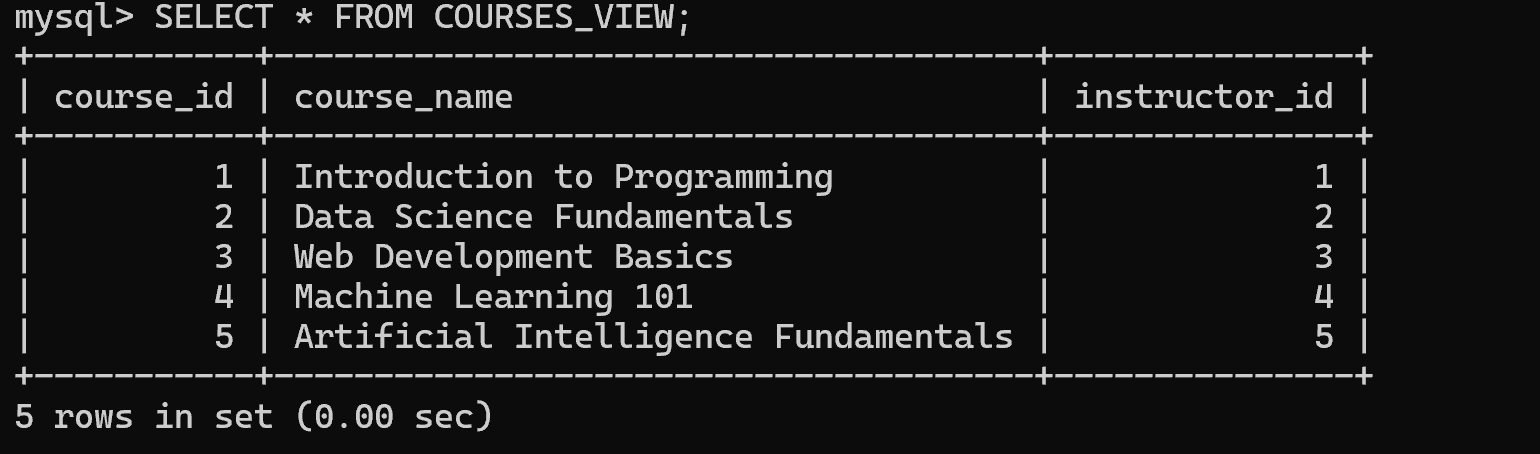


**View for Courses table**

CREATE VIEW Courses\_View AS

SELECT course\_id, course\_name, instructor\_id

FROM Courses;

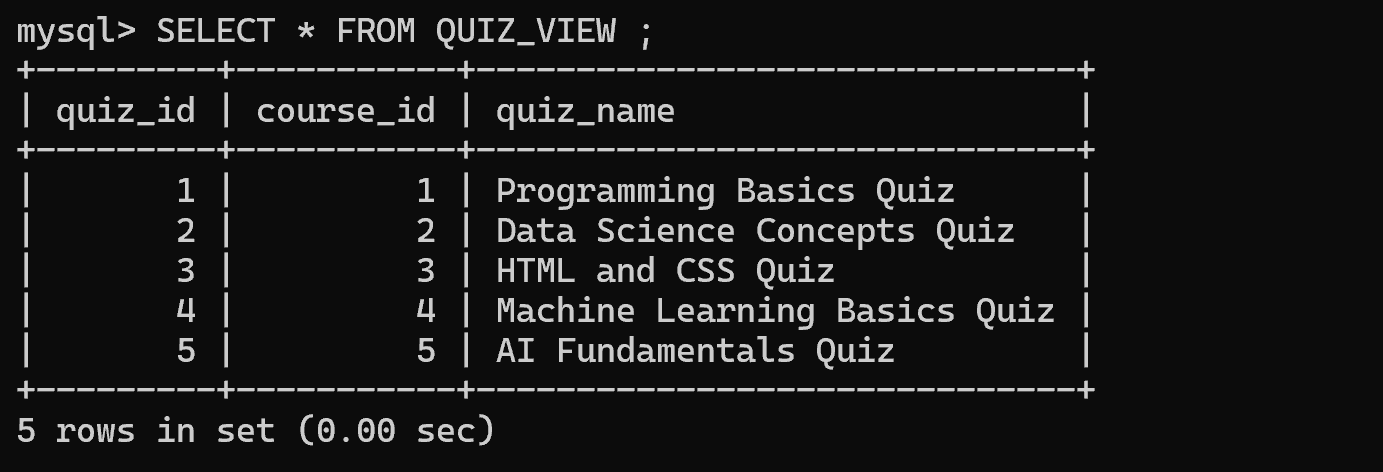


**View for Quiz table**

CREATE VIEW Quiz\_View AS

SELECT quiz\_id, course\_id, quiz\_name

FROM Quiz;

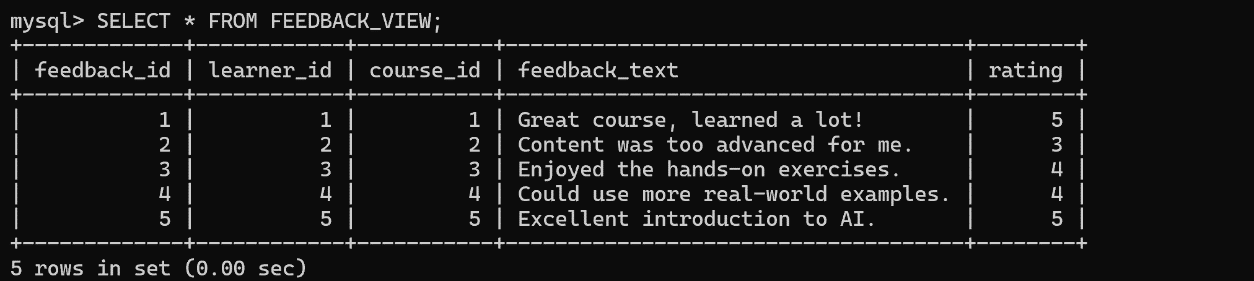


**View for Feedback table**

CREATE VIEW Feedback\_View AS

SELECT feedback\_id, learner\_id, course\_id, feedback\_text, rating

FROM Feedback;

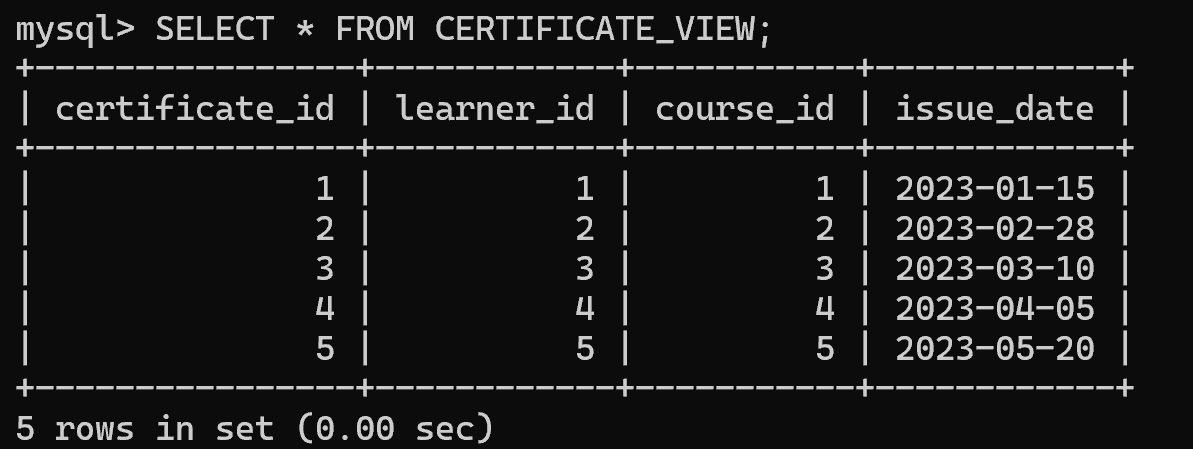


**View for Certificate table**

CREATE VIEW Certificate\_View AS

SELECT certificate\_id, learner\_id, course\_id, issue\_date

FROM Certificate;



**View for Administrator table**

CREATE VIEW Administrator\_View AS

SELECT admin\_id, first\_name, last\_name, email

FROM Administrator;

**SETS AND JOINS**

-- **Query to find learners who provided feedback, along with their courses and instructors**

**-- Selecting learners who provided feedback along with their feedback detail**s

SELECT

L.first\_name AS learner\_first\_name,

L.last\_name AS learner\_last\_name,

F.feedback\_text,

F.rating,

NULL AS course\_name, -- Placeholder for course name

NULL AS instructor\_first\_name, -- Placeholder for instructor first name

NULL AS instructor\_last\_name -- Placeholder for instructor last name

FROM

Learners L

LEFT JOIN

Feedback F ON L.learner\_id = F.learner\_id

UNION

course\_id | course\_name | instructor\_id | first\_name | last\_name | email

----------|--------------------------|---------------|------------|--------------------|-------------------------

1 | Introduction to Programming | 1 | Michael | Johnson | michael.johnson@example.com

2 | Web Development Fundamentals | 2 | Emily | Brown | emily.brown@example.com

-- **Selecting learners along with their enrolled courses and instructors**

SELECT

L.first\_name AS learner\_first\_name,

L.last\_name AS learner\_last\_name,

NULL AS feedback\_text, -- Placeholder for feedback text

NULL AS rating, -- Placeholder for rating

C.course\_name,

I.first\_name AS instructor\_first\_name,

I.last\_name AS instructor\_last\_name

FROM

Learners L

LEFT JOIN

Courses C ON L.course\_id = C.course\_id

LEFT JOIN

Instructors I ON C.instructor\_id = I.instructor\_id;

**TRIGGERS**

**Trigger for Learners Table:**

Let's say you want to automatically insert a record into the Certificate table whenever a new learner is inserted into the Learners table.

sql CREATE TRIGGER insert\_certificate AFTER INSERT ON Learners

FOR EACH ROW

BEGIN

INSERT INTO Certificate (learner\_id, course\_id, issue\_date)

VALUES (NEW.learner\_id, 1, NOW()); -- Assuming course\_id 1 is a default course for certification

END;

**Trigger for Courses Table:**

Let's say you want to automatically insert a record into the Quiz table whenever a new course is inserted into the Courses table.

sql

CREATE TRIGGER insert\_quiz AFTER INSERT ON Courses

FOR EACH ROW

BEGIN

INSERT INTO Quiz (course\_id, quiz\_name)

VALUES (NEW.course\_id, CONCAT('Quiz for ', NEW.course\_name));

END;

**Trigger for Feedback Table:**

Let's say you want to log feedback actions in a separate table whenever a new feedback is inserted into the Feedback table.

sql

CREATE TRIGGER log\_feedback\_action AFTER INSERT ON Feedback

FOR EACH ROW

BEGIN

INSERT INTO Feedback\_Log (action\_type, learner\_id, course\_id, feedback\_text, rating, action\_timestamp)

VALUES ('INSERT', NEW.learner\_id, NEW.course\_id, NEW.feedback\_text, NEW.rating, NOW());

END;

CREATE TRIGGER prevent\_delete\_certificate BEFORE DELETE ON Certificate

FOR EACH ROW

BEGIN

-- Check if the certificate is associated with a learner

IF OLD.learner\_id IS NOT NULL THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Cannot delete certificates associated with learners.';

END IF;

END;

CREATE TRIGGER log\_administrator\_updates AFTER UPDATE ON Administrator

FOR EACH ROW

BEGIN

INSERT INTO Administrator\_Log (admin\_id, updated\_column, old\_value, new\_value, update\_timestamp)

VALUES (NEW.admin\_id, 'ALL', CONCAT('Old values: ', OLD.first\_name, ', ', OLD.last\_name, ', ', OLD.email), CONCAT('New values: ', NEW.first\_name, ', ', NEW.last\_name, ', ', NEW.email), NOW());

END;

**CURSOR**

**Cursor for Learners table**

DROP PROCEDURE IF EXISTS fetch\_learners;

DELIMITER //

CREATE PROCEDURE fetch\_learners()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE learner\_id INT;

DECLARE first\_name VARCHAR(50);

DECLARE last\_name VARCHAR(50);

DECLARE email VARCHAR(100);

DECLARE date\_of\_birth DATE;

DECLARE enrollment\_date TIMESTAMP;

DECLARE cur CURSOR FOR SELECT \* FROM Learners;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO learner\_id, first\_name, last\_name, email, date\_of\_birth, enrollment\_date;

IF done THEN

LEAVE read\_loop;

END IF;

-- Process fetched data as needed

-- For example, you can print or manipulate the fetched data

-- Here, we'll just print the fetched data

SELECT learner\_id, first\_name, last\_name, email, date\_of\_birth, enrollment\_date;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

**Cursor for Instructors table**

DROP PROCEDURE IF EXISTS fetch\_instructors;

DELIMITER //

CREATE PROCEDURE fetch\_instructors()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE instructor\_id INT;

DECLARE first\_name VARCHAR(50);

DECLARE last\_name VARCHAR(50);

DECLARE email VARCHAR(100);

DECLARE cur CURSOR FOR SELECT \* FROM Instructors;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO instructor\_id, first\_name, last\_name, email;

IF done THEN

LEAVE read\_loop;

END IF;

-- Process fetched data as needed

-- For example, you can print or manipulate the fetched data

-- Here, we'll just print the fetched data

SELECT instructor\_id, first\_name, last\_name, email;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

**Cursor for Administrator table**

DROP PROCEDURE IF EXISTS fetch\_administrators;

DELIMITER //

CREATE PROCEDURE fetch\_administrators()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE admin\_id INT;

DECLARE first\_name VARCHAR(50);

DECLARE last\_name VARCHAR(50);

DECLARE email VARCHAR(100);

DECLARE cur CURSOR FOR SELECT \* FROM Administrator;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO admin\_id, first\_name, last\_name, email;

IF done THEN

LEAVE read\_loop;

END IF;

-- Process fetched data as needed

-- For example, you can print or manipulate the fetched data

-- Here, we'll just print the fetched data

SELECT admin\_id, first\_name, last\_name, email;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

**Cursor for Courses table**

DROP PROCEDURE IF EXISTS fetch\_courses;

DELIMITER //

CREATE PROCEDURE fetch\_courses()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE course\_id INT;

DECLARE course\_name VARCHAR(100);

DECLARE instructor\_id INT;

DECLARE cur CURSOR FOR SELECT \* FROM Courses;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO course\_id, course\_name, instructor\_id;

IF done THEN

LEAVE read\_loop;

END IF;

-- Process fetched data as needed

-- For example, you can print or manipulate the fetched data

-- Here, we'll just print the fetched data

SELECT course\_id, course\_name, instructor\_id;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

**Cursor for Quiz table**

DROP PROCEDURE IF EXISTS fetch\_quiz;

DELIMITER //

CREATE PROCEDURE fetch\_quiz()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE quiz\_id INT;

DECLARE course\_id INT;

DECLARE quiz\_name VARCHAR(100);

DECLARE cur CURSOR FOR SELECT \* FROM Quiz;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO quiz\_id, course\_id, quiz\_name;

IF done THEN

LEAVE read\_loop;

END IF;

-- Process fetched data as needed

-- For example, you can print or manipulate the fetched data

-- Here, we'll just print the fetched data

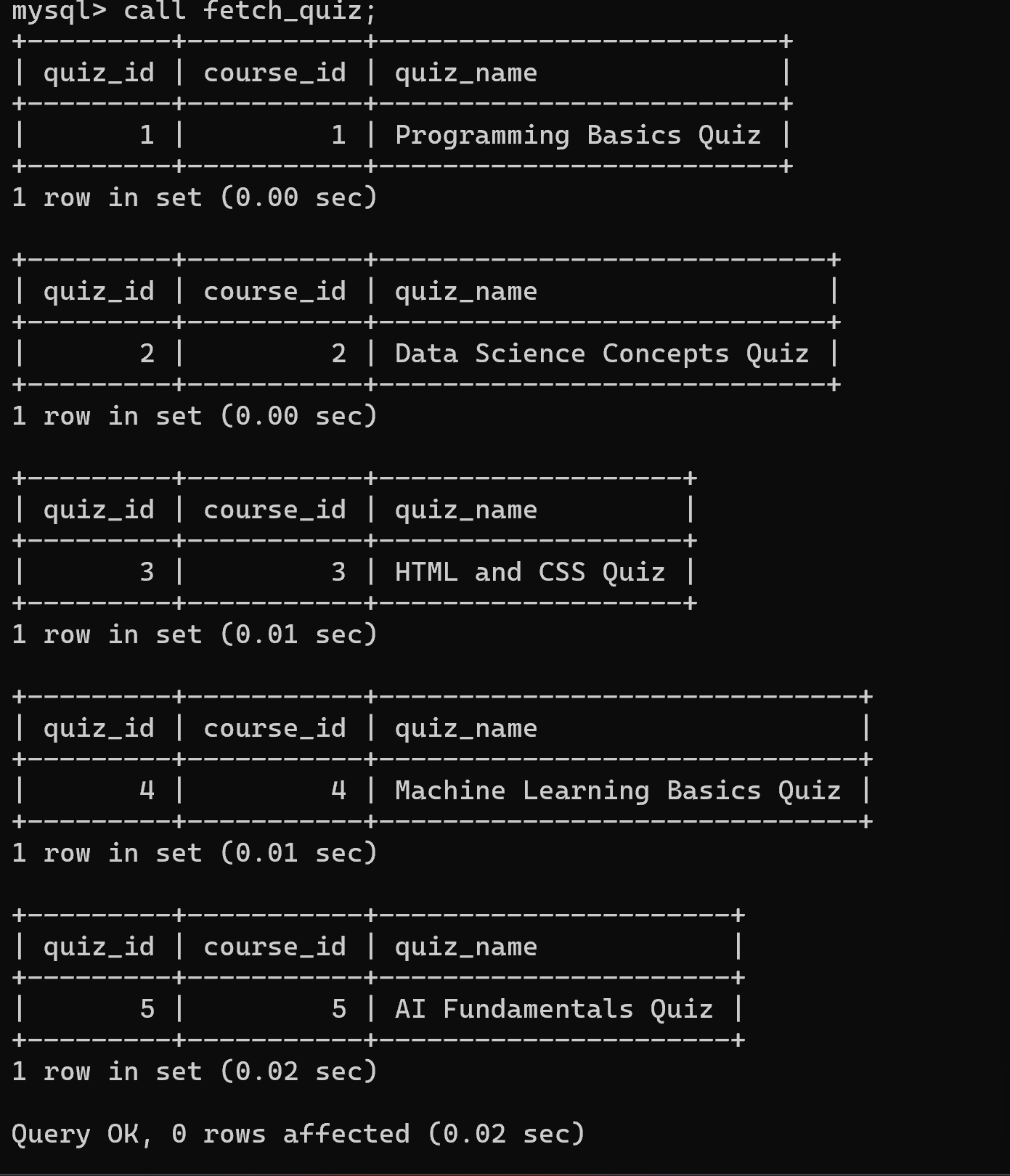
SELECT quiz\_id, course\_id, quiz\_name;

END LOOP;

CLOSE cur;

END //

DELIMITER ;



**Cursor for Feedback table**

DROP PROCEDURE IF EXISTS fetch\_feedback;

DELIMITER //

CREATE PROCEDURE fetch\_feedback()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE feedback\_id INT;

DECLARE learner\_id INT;

DECLARE course\_id INT;

DECLARE feedback\_text TEXT;

DECLARE rating INT;

DECLARE cur CURSOR FOR SELECT \* FROM Feedback;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO feedback\_id, learner\_id, course\_id, feedback\_text, rating;

IF done THEN

LEAVE read\_loop;

END IF;

-- Process fetched data as needed

-- For example, you can print or manipulate the fetched data

-- Here, we'll just print the fetched data

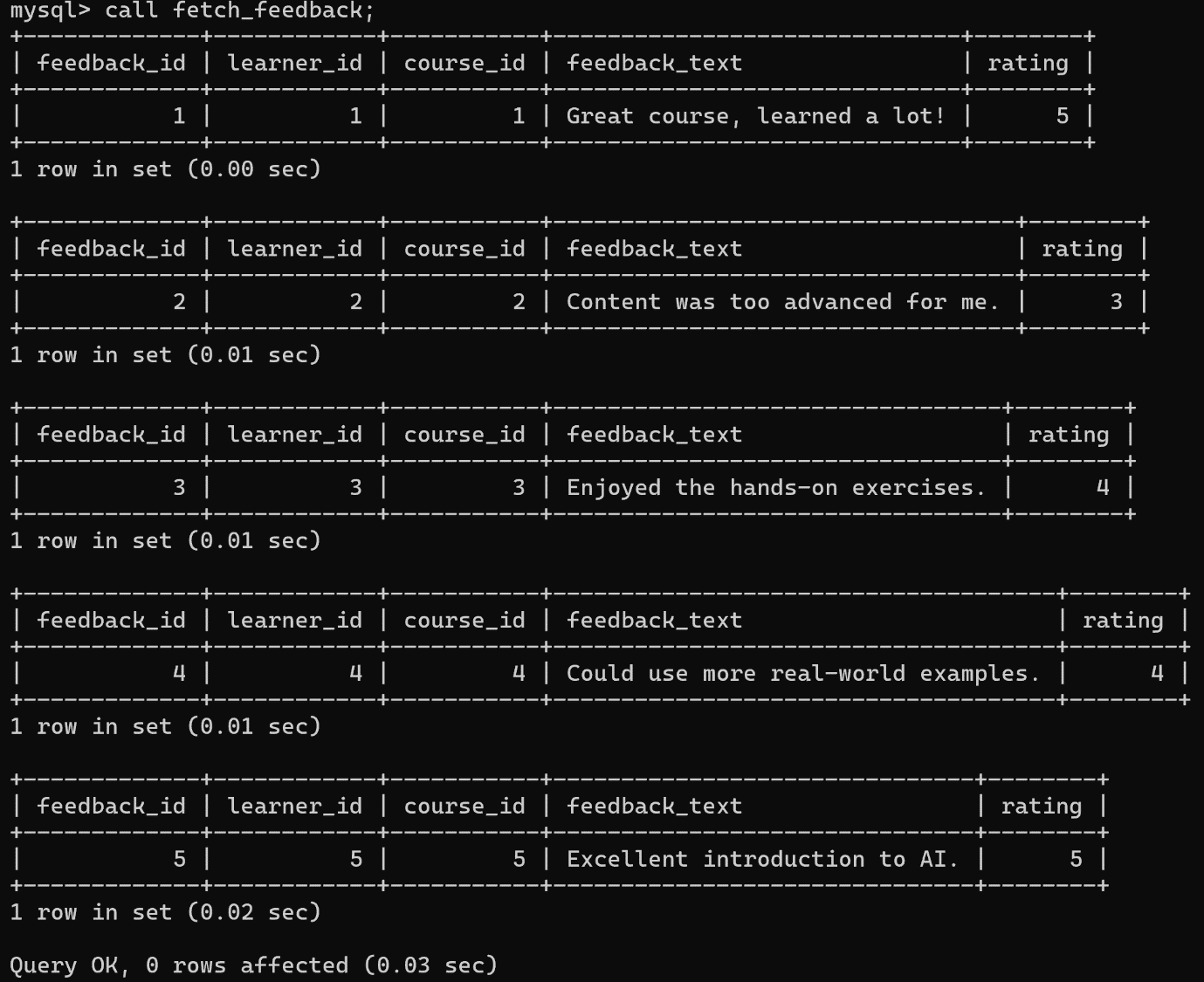
SELECT feedback\_id, learner\_id, course\_id, feedback\_text, rating;

END LOOP;

CLOSE cur;

END //

DELIMITER ;



**Cursor for Certificate table**

DROP PROCEDURE IF EXISTS fetch\_certificate;

DELIMITER //

CREATE PROCEDURE fetch\_certificate()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE certificate\_id INT;

DECLARE learner\_id INT;

DECLARE course\_id INT;

DECLARE issue\_date DATE;

DECLARE cur CURSOR FOR SELECT \* FROM Certificate;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO certificate\_id, learner\_id, course\_id, issue\_date;

IF done THEN

LEAVE read\_loop;

END IF;

-- Process fetched data as needed

-- For example, you can print or manipulate the fetched data

-- Here, we'll just print the fetched data

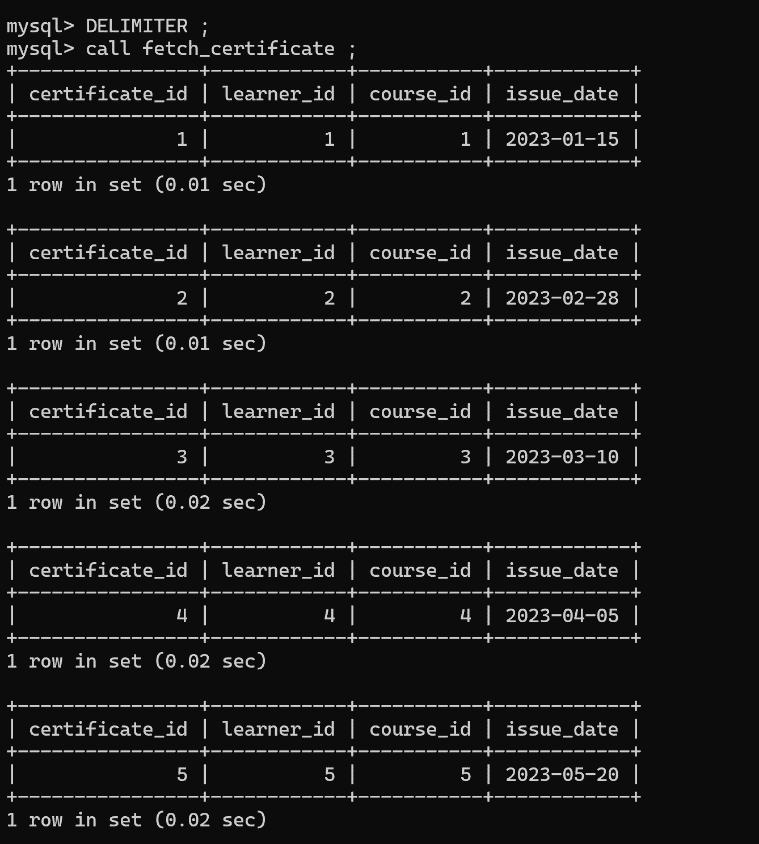
SELECT certificate\_id, learner\_id, course\_id, issue\_date;

END LOOP;

CLOSE cur;

END //

DELIMITER ;



DELIMITER //

CREATE TRIGGER UpdateEnrollmentDate

BEFORE INSERT ON Learners

FOR EACH ROW

BEGIN

SET NEW.enrollment\_date = CURRENT\_TIMESTAMP;

END;

//

DELIMITER ;

INSERT INTO Learners (first\_name, last\_name, email, password, date\_of\_birth)

VALUES ('Sarah', 'Johnson', 'sarah.johnson@example.com', 'sarah123', '1998-07-10');

SELECT \* FROM Learners;

UPDATE Learners

SET password = 'newpassword'

WHERE learner\_id = 1;

DELETE FROM Learners

WHERE learner\_id = 2;

**NORMALIZATION**

Normalization is a database design technique used to organize data in a relational database efficiently. It aims to minimize redundancy and dependency by organizing the fields and tables of a database. The process involves dividing large tables into smaller ones and defining relationships between them.

**LEARNER’S TABLE**

**CODE:**

CREATE TABLE learners\_before\_1nf (

learner\_id INT PRIMARY KEY,

learner\_name VARCHAR(100),

email VARCHAR(100),

date\_of\_birth DATE,

address VARCHAR(255),

gender ENUM('Male', 'Female', 'Other'),

course\_id INT,

course\_name VARCHAR(100),

instructor\_id INT,

instructor\_name VARCHAR(100),

department\_id INT,

department\_name VARCHAR(100),

department\_head VARCHAR(100)

);

INSERT INTO learners\_before\_1nf (learner\_id, learner\_name, email, date\_of\_birth, address, gender, course\_id, course\_name, instructor\_id, instructor\_name, department\_id, department\_name, department\_head)

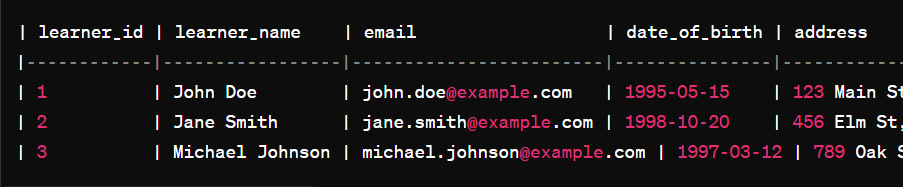
VALUES

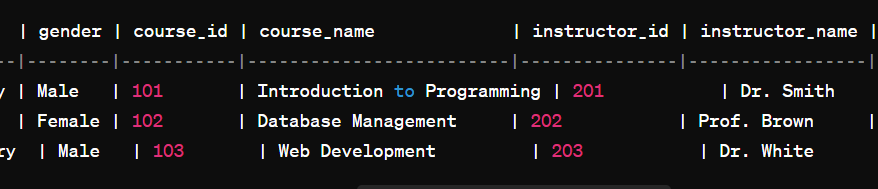
(1, 'John Doe', 'john.doe@example.com', '1995-05-15', '123 Main St, City, Country', 'Male', 101, 'Introduction to Programming', 201, 'Dr. Smith', 301, 'Computer Science', 'Prof. Johnson'),

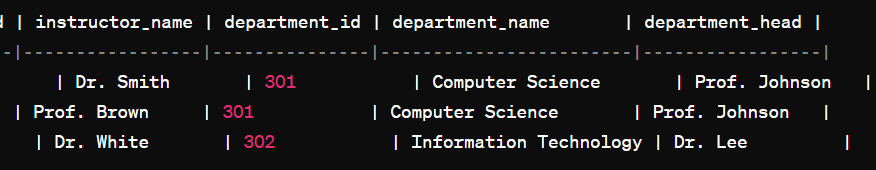
(2, 'Jane Smith', 'jane.smith@example.com', '1998-10-20', '456 Elm St, City, Country', 'Female', 102, 'Database Management', 202, 'Prof. Brown', 301, 'Computer Science', 'Prof. Johnson'),

(3, 'Michael Johnson', 'michael.johnson@example.com', '1997-03-12', '789 Oak St, City, Country', 'Male', 103, 'Web Development', 203, 'Dr. White', 302, 'Information Technology', 'Dr. Lee');

**TABLE:**



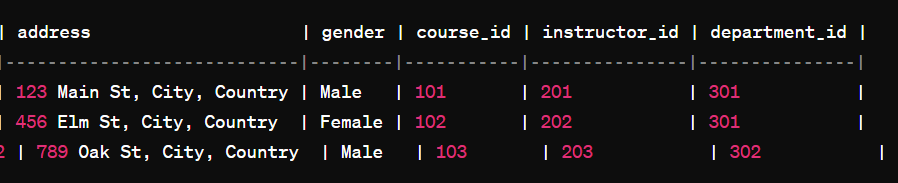




**1ST NORMAL FORM:**

First Normal Form (1NF) is a property of a relation in a relational database. It is the simplest form of normalization and sets the foundation for database design by ensuring that each attribute contains atomic values, meaning that each attribute holds only a single value, and there are no repeating groups or arrays within a tuple.

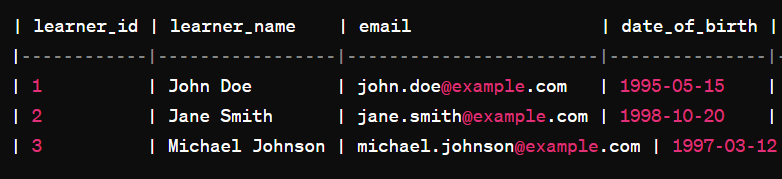


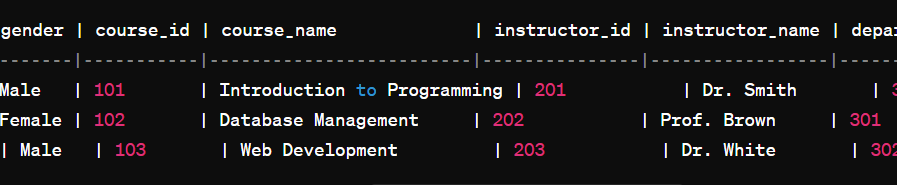


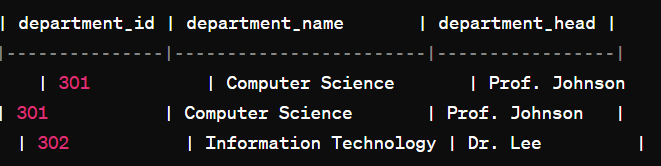
**2ND NORMAL FORM:**

Second Normal Form (2NF) is a property of a relation in a relational database that builds upon the First Normal Form (1NF). It further refines the database design by eliminating partial dependencies, ensuring that all non-key attributes are fully functionally dependent on the entire primary key.

BEFORE:





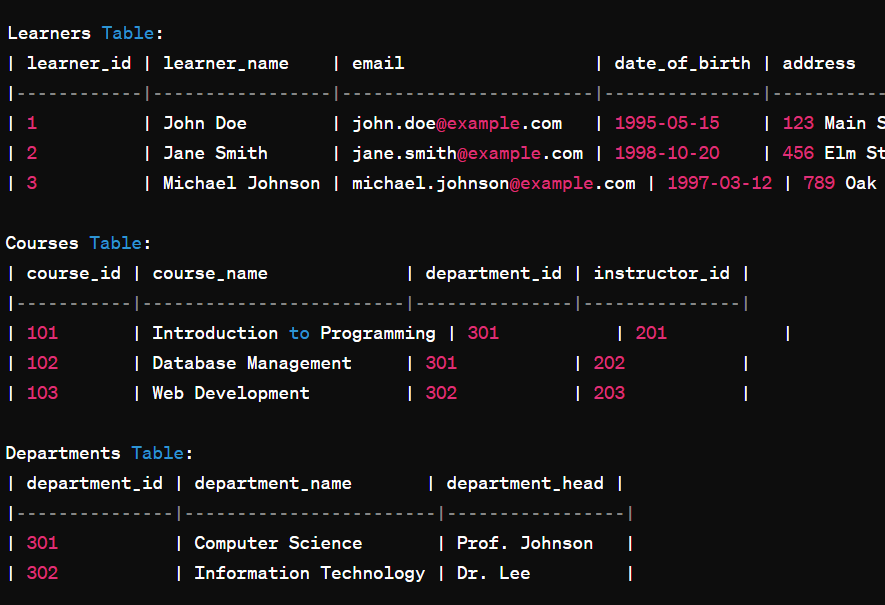


**AFTER 2NF:**



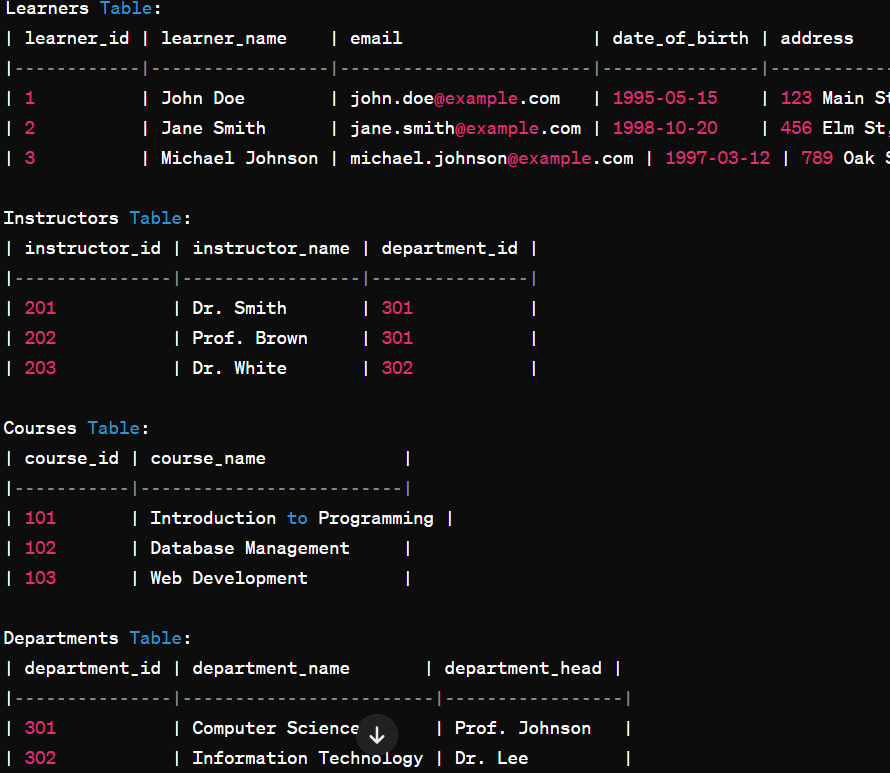
**3RD NORMAL FORM:**

Third Normal Form (3NF) is a property of a relation in a relational database that builds upon the Second Normal Form (2NF). It further refines the database design by eliminating transitive dependencies, ensuring that all non-key attributes are fully functionally dependent only on the primary key and not on other non-key attributes.

APPLYING 3NF:

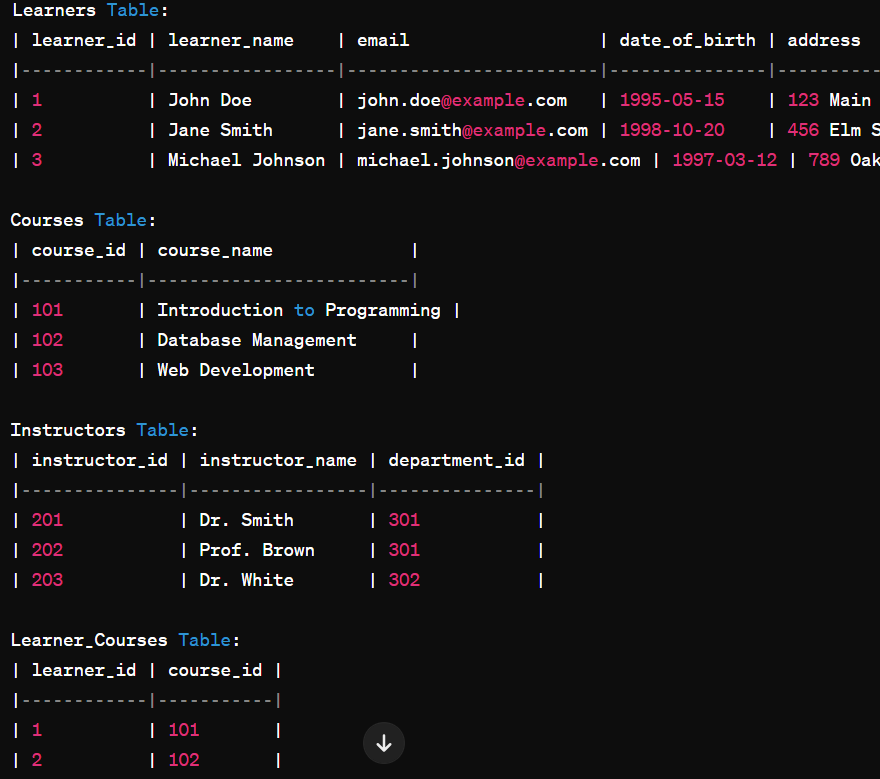
**4TH NORMAL FORM:**

Fourth Normal Form (4NF) in database normalization eliminates multi-valued dependencies, ensuring that each attribute is fully functionally dependent on the primary key. This reduces redundancy and improves data integrity.



**5TH NORMAL FORM:**

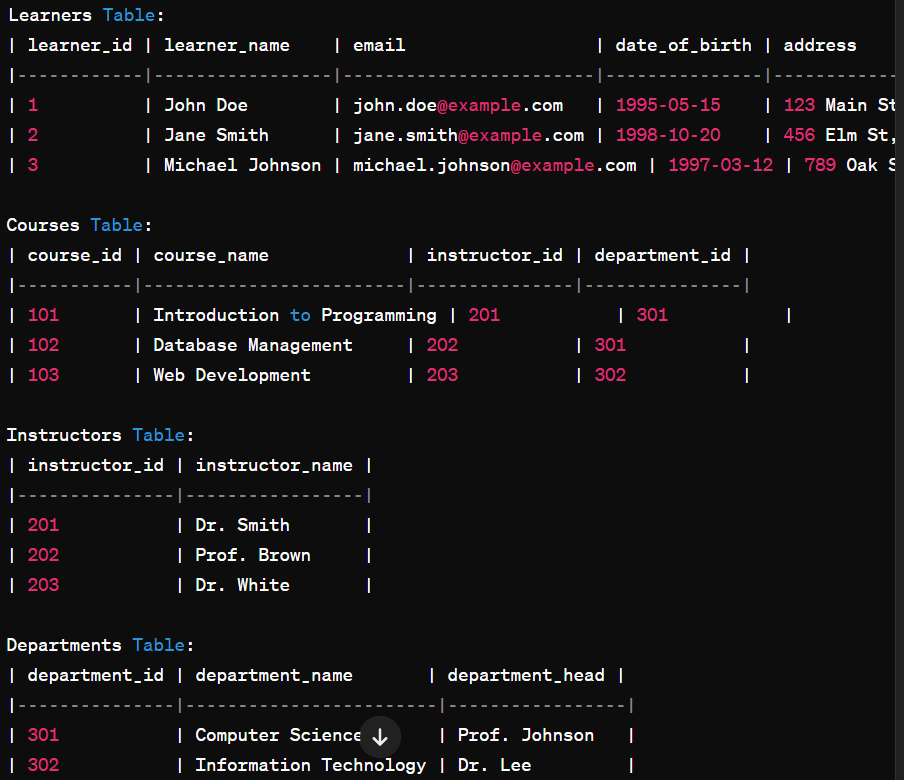
Fifth Normal Form (5NF) further refines database organization by removing join dependencies, which enhances data integrity and minimizes redundancy, improving database efficiency and maintenance.



**BCNF:**

Boyce-Codd Normal Form (BCNF) is a level of database normalization that refines the Third Normal Form (3NF) by addressing certain types of functional dependencies. A table is said to be in BCNF if it meets the following conditions:

* Functional Dependency: Every determinant (i.e., attribute or set of attributes) that is not a candidate key must be a superkey. In other words, every non-trivial functional dependency in the table should be determined by a candidate key.
* Elimination of Partial Dependencies: BCNF eliminates partial dependencies, ensuring that each attribute in a table is fully functionally dependent on the primary key, and there are no attributes that depend on only part of the primary key.



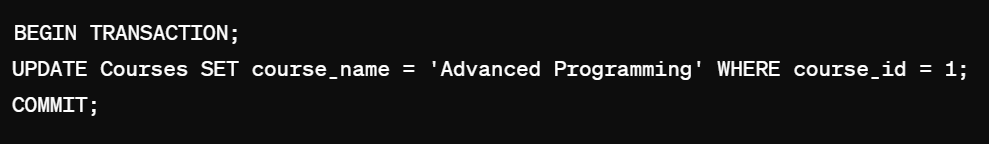
**Implementation of concurrency control and recovery mechanisms**

Sample Output for Concurrency Control:

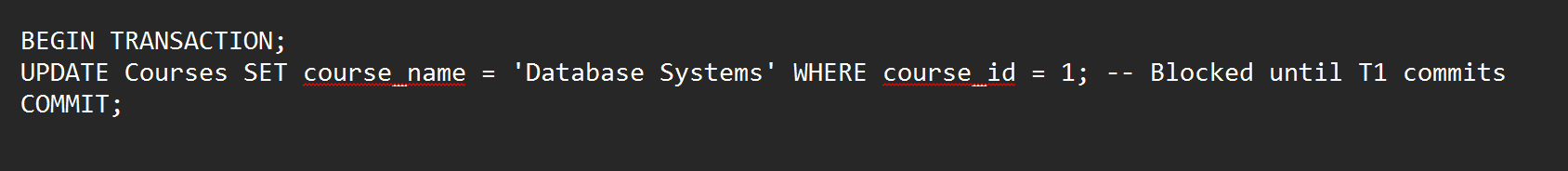
Locking Mechanism:

Assume two transactions are concurrently updating the same row in the Courses table. The lock manager ensures that only one transaction can modify the row at a time to prevent conflicts.

**Transaction 1 (T1):**

****

**Transaction 2 (T2**):

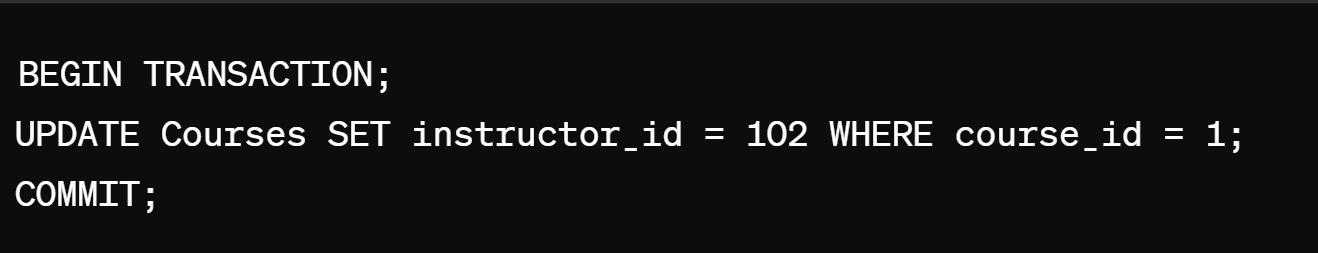
****

In this scenario, Transaction 2 will be blocked until Transaction 1 commits or rolls back.

Multi-Version Concurrency Control (MVCC):

Assume a transaction reads data from the Courses table while another transaction is updating the same row.

**Transaction 1 (T1):**

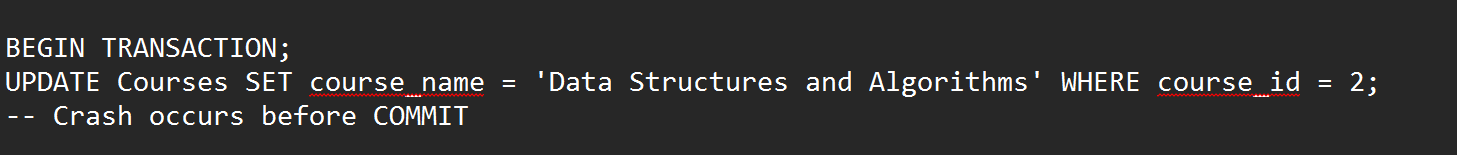
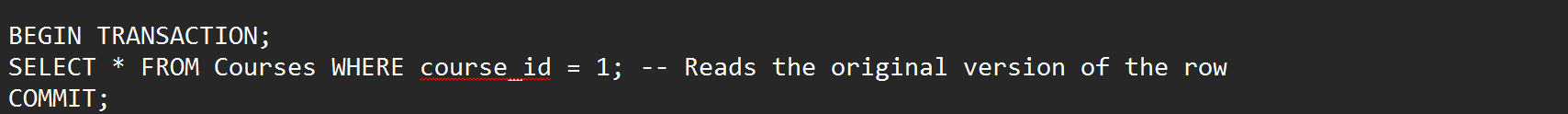
****

Transaction 2 reads the original version of the row before Transaction 1's update.

**Sample Output for Recovery Mechanisms:**

**Write-Ahead Logging (WAL):**

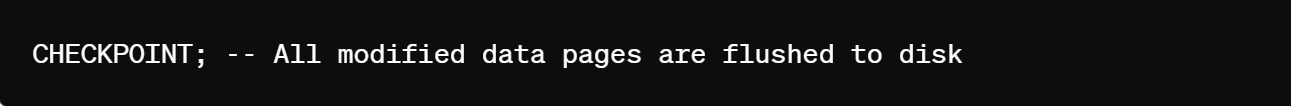
Assume a transaction updates data in the Courses table, and the system crashes before the changes are written to disk.

****

Upon recovery, the system replays the log to redo the update operation and restore the database to a consistent state.

**Checkpoints:**

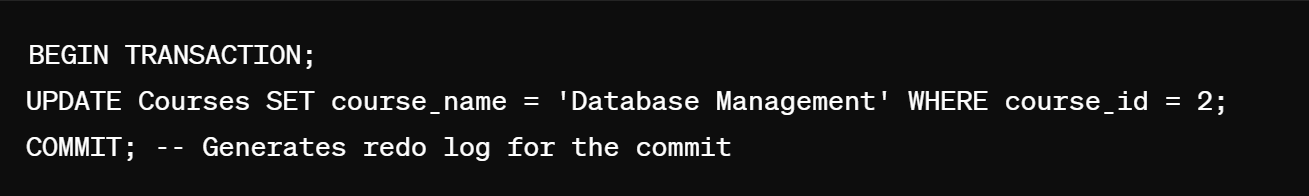
Assume a checkpoint is triggered, and all modified data pages are flushed to disk.

****

This ensures that the database has a consistent state on disk, making recovery faster and more efficient.

**Undo/Redo Logging:**

Assume a transaction commits its changes, and the system logs both before and after images of the modified data.

****

If a crash occurs, the system uses the redo log to reapply the committed changes during recovery.

**Replication and Backup:**

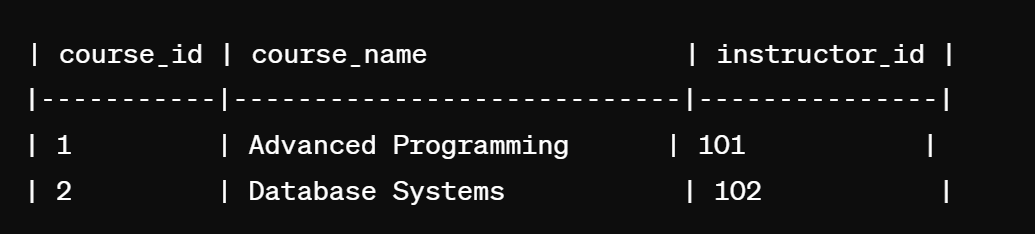
Assume database replication and regular backups are configured to ensure data availability and disaster recovery.

**Database Replication:** Maintains copies of data on multiple servers for fault tolerance. If one server fails, the other servers can continue serving requests.

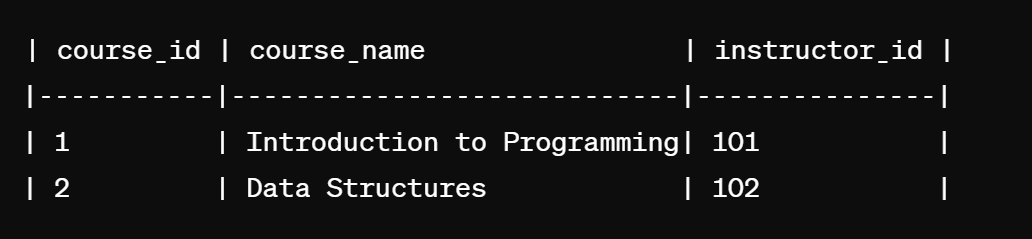
Regular Backups: Periodically back up the database to ensure data integrity. In case of data loss or corruption, the database can be restored from the backups.

**Sample Updated Tables for Concurrency Control:**

**Courses Table (after locking mechanism):**

****

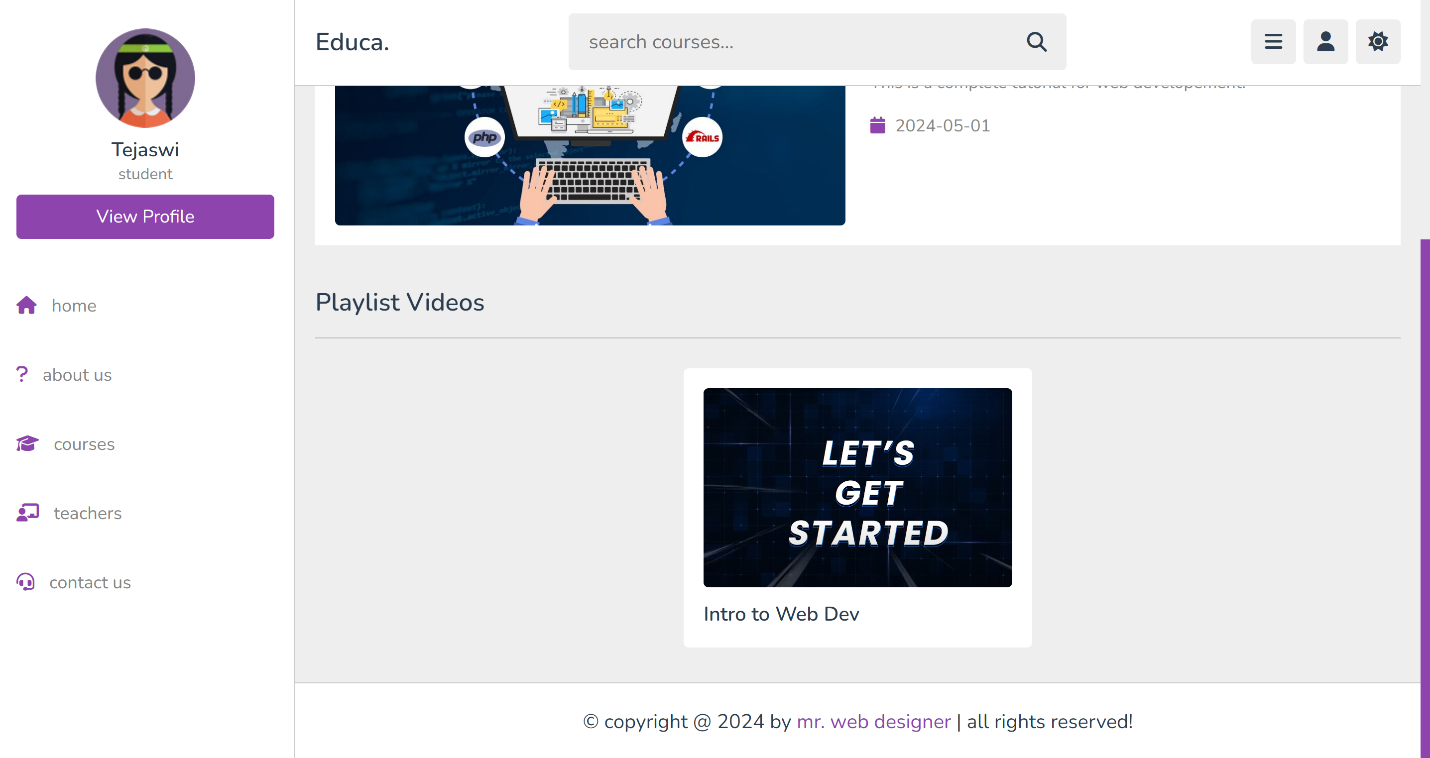
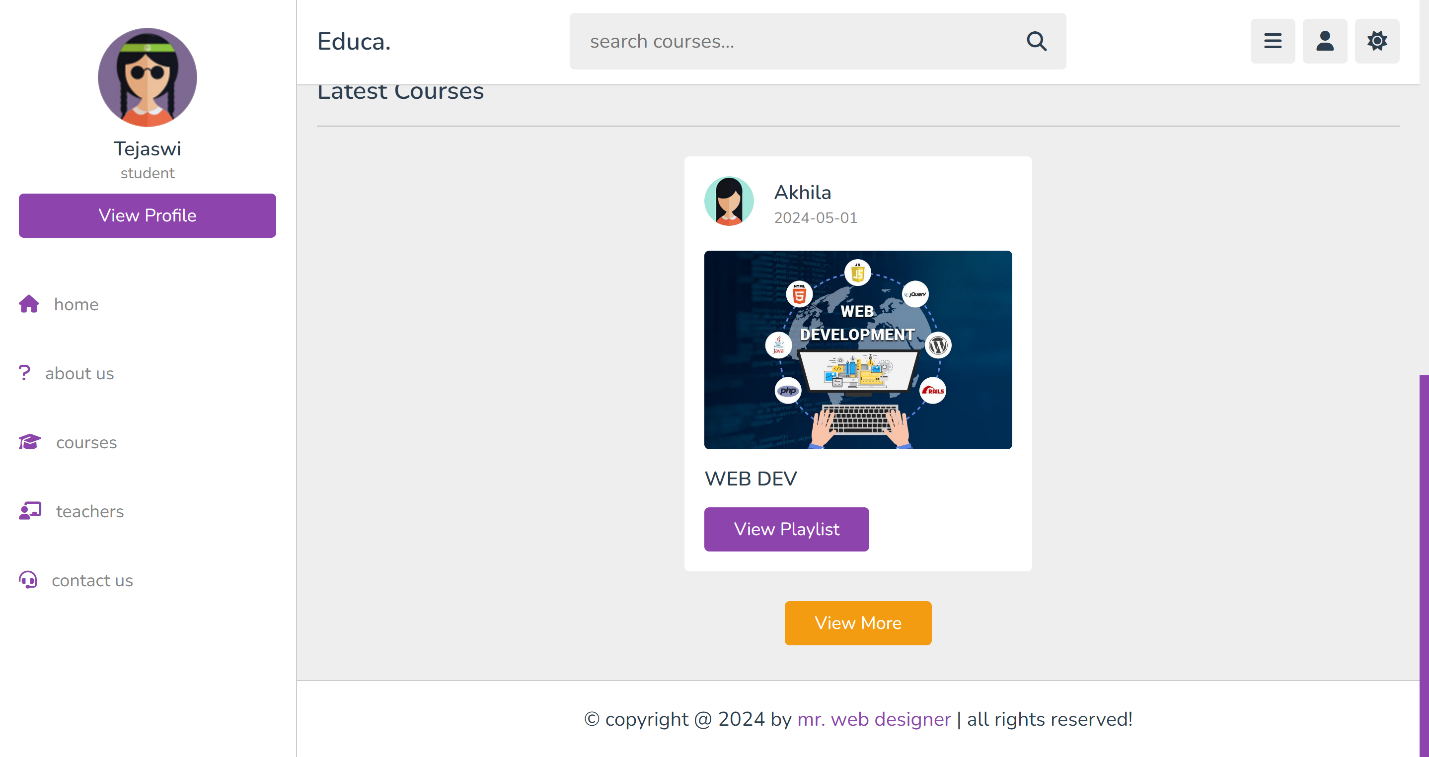
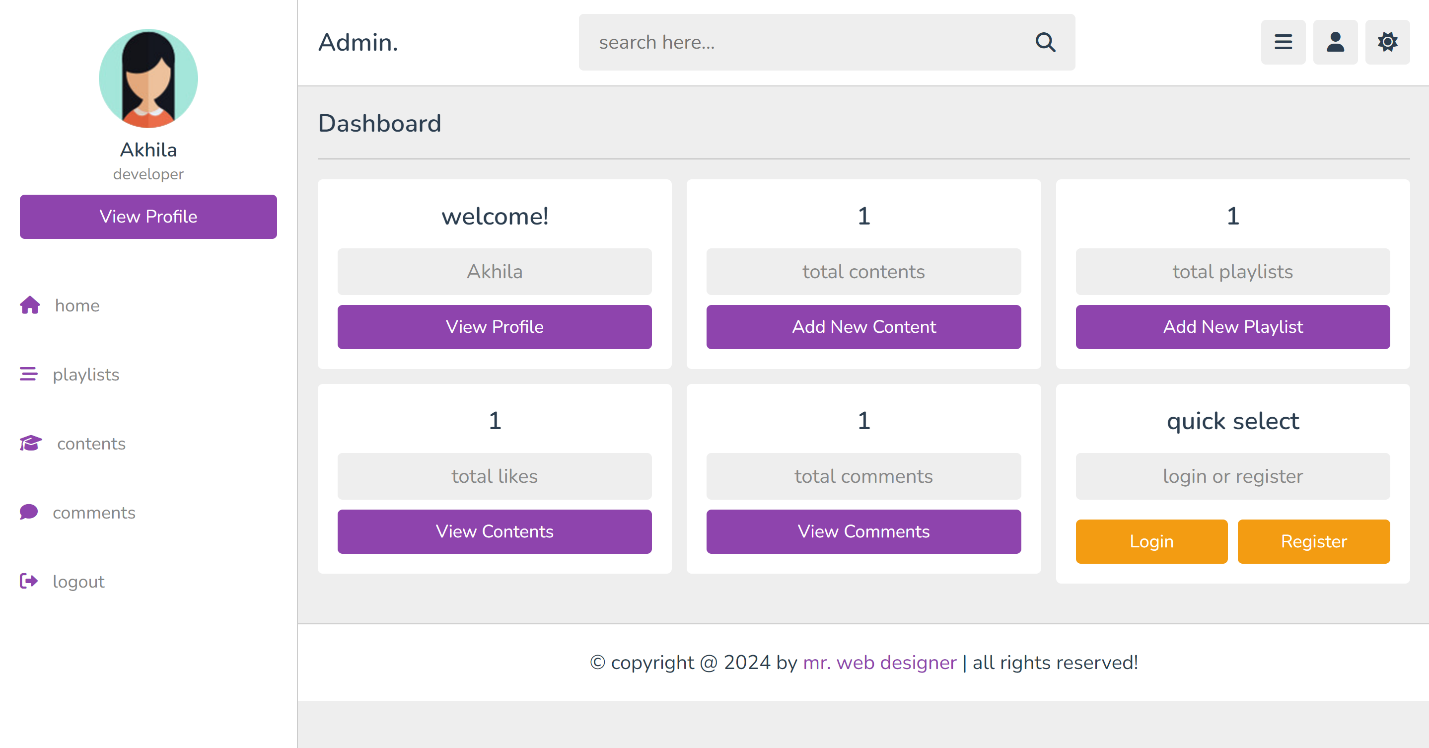
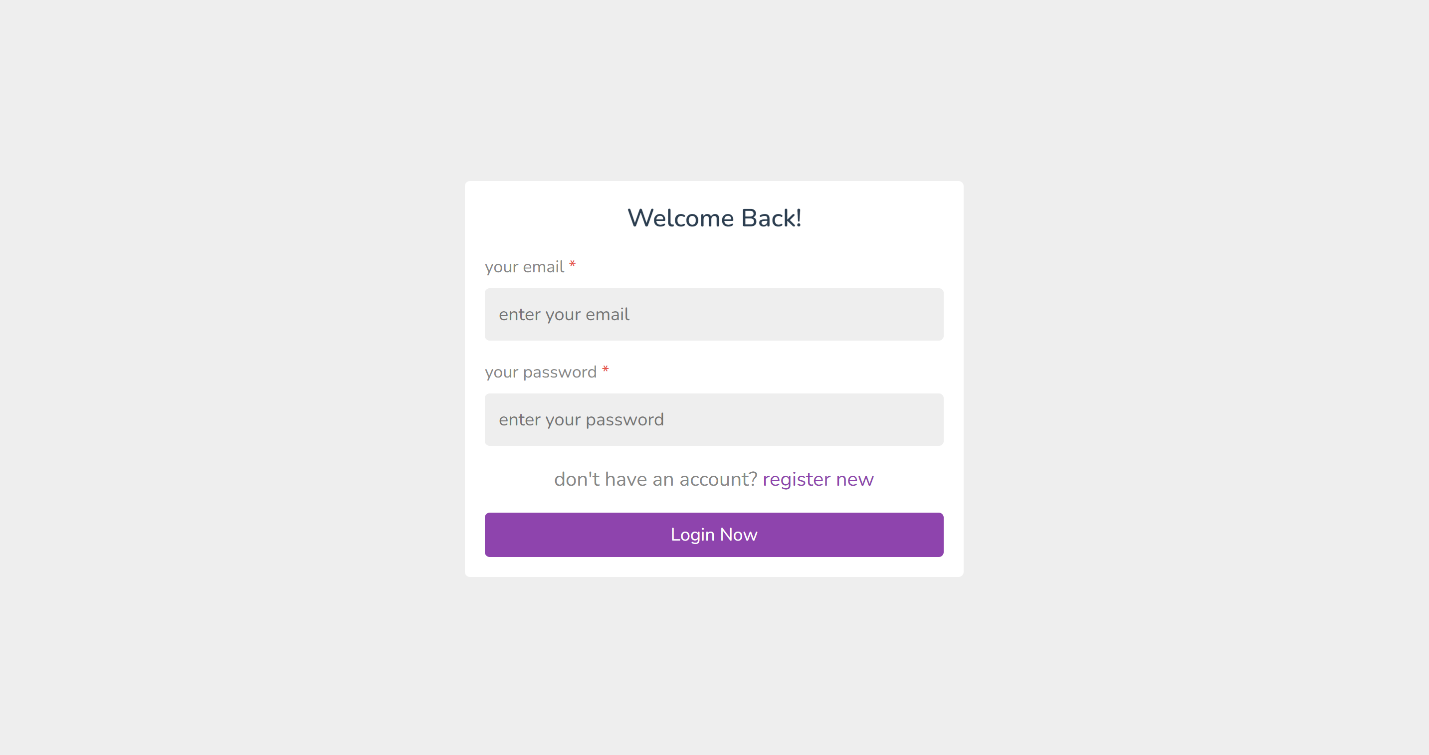
#### Courses Table (After MVCC):

****

Assuming Transaction 2 reads the original version of the row before Transaction 1's update)

**Code for the Project:**

**Result and Screenshots of the Project**

****