

Week 5

Alter, Update, Trigger and Transaction

Recap- Stored Procedure

- a set of SQL statements stored in the database that can be executed as a single unit.
- can perform operations like inserting, updating, and deleting data.

Stored Procedure

```
CREATE TABLE Students (  
    StudentID INT AUTO_INCREMENT  
    PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    Email VARCHAR(100)  
);
```

Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE AddStudent(  
    IN p_FirstName VARCHAR(50),  
    IN p_LastName VARCHAR(50),  
    IN p_Email VARCHAR(100)  
)  
BEGIN  
    INSERT INTO Students (FirstName, LastName,  
Email)  
    VALUES (p_FirstName, p_LastName, p_Email);  
END //
```

```
DELIMITER ;
```

Stored Procedure

```
CALL AddStudent('John', 'Doe',  
'john.doe@example.com');
```

```
SELECT * FROM Students;
```

Create a simple employee table in EmpDB

- emp_id – employee ID (primary key)
- first_name,
- last_name – name fields
- department – which team they work in
- salary – their pay

Create a simple employee table in EmpDB

- emp_id – employee ID (primary key)
- first_name,
- last_name – name fields
- department – which team they work in
- salary – their pay

```
CREATE DATABASE IF NOT EXISTS EmpDB;  
USE EmpDB;
```

Create a simple employee table

- emp_id – employee ID (primary key)
- first_name,
- last_name – name fields
- department – which team they work in
- salary – their pay

```
CREATE TABLE employees (  
    emp_id INT PRIMARY KEY,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    department VARCHAR(50),  
    salary DECIMAL(10, 2)  
);
```

Insert datasets

emp_id	first_name	last_name	department	salary
101	John	Smith	HR	55000.00
102	Emma	Johnson	Finance	62000.00
103	Liam	Williams	IT	75000.00
104	Olivia	Brown	Marketing	58000.00
105	Noah	Davis	IT	72000.00

Insert datasets

emp_id	first_name	last_name	department	salary
101	John	Smith	HR	55000.00
102	Emma	Johnson	Finance	62000.00
103	Liam	Williams	IT	75000.00
104	Olivia	Brown	Marketing	58000.00
105	Noah	Davis	IT	72000.00

```
INSERT INTO employees (emp_id, first_name, last_name, department,  
salary)
```

```
VALUES
```

```
(101, 'John', 'Smith', 'HR', 55000.00),  
(102, 'Emma', 'Johnson', 'Finance', 62000.00),  
(103, 'Liam', 'Williams', 'IT', 75000.00),  
(104, 'Olivia', 'Brown', 'Marketing', 58000.00),  
(105, 'Noah', 'Davis', 'IT', 72000.00);
```

Update data using UPDATE

emp_id	first_name	last_name	department	salary
101	John	Smith	HR	55000.00
102	Emma	Johnson	Finance	62000.00
103	Liam	Williams	IT	75000.00
104	Olivia	Brown	Marketing	58000.00
105	Noah	Davis	IT	72000.00

Update data using UPDATE

emp_id	first_name	last_name	department	salary
101	John	Smith	HR	55000.00
102	Emma	Johnson	Finance	62000.00
103	Liam	Williams	IT	75000.00
104	Olivia	Brown	Marketing	58000.00
105	Noah	Davis	IT	72000.00

```
UPDATE employees  
SET salary = 60000.00  
WHERE emp_id = 101;
```

Update data using UPDATE

emp_id	first_name	last_name	department	salary
101	John	Smith	HR	55000.00
102	Emma	Johnson	Finance	62000.00
103	Liam	Williams	IT	75000.00
104	Olivia	Brown	Marketing	58000.00
105	Noah	Davis	IT	72000.00

Give everyone in IT department a 10% raise:

Update data using UPDATE

emp_id	first_name	last_name	department	salary
101	John	Smith	HR	55000.00
102	Emma	Johnson	Finance	62000.00
103	Liam	Williams	IT	75000.00
104	Olivia	Brown	Marketing	58000.00
105	Noah	Davis	IT	72000.00

Give everyone in IT department a 10% raise:

```
UPDATE employees  
SET salary = salary * 1.10  
WHERE department = 'IT';
```

test with a `SELECT` first to confirm which rows will change.

Alter the table using ALTER

emp_id	first_name	last_name	department	salary	Email
101	John	Smith	HR	55000.00	
102	Emma	Johnson	Finance	62000.00	
103	Liam	Williams	IT	75000.00	
104	Olivia	Brown	Marketing	58000.00	
105	Noah	Davis	IT	72000.00	

Add a new column for employee email

test with a `SELECT` first to confirm which rows will change.

Alter the table using ALTER

emp_id	first_name	last_name	department	salary	Email
101	John	Smith	HR	55000.00	
102	Emma	Johnson	Finance	62000.00	
103	Liam	Williams	IT	75000.00	
104	Olivia	Brown	Marketing	58000.00	
105	Noah	Davis	IT	72000.00	

Add a new column for employee email

```
ALTER TABLE employees  
ADD email VARCHAR(100);
```

test with a DESCRIBE .

Update the new column email with emailids

emp_id	first_name	last_name	department	salary	Email
101	John	Smith	HR	55000.00	Fn.ln@company.com
102	Emma	Johnson	Finance	62000.00	
103	Liam	Williams	IT	75000.00	
104	Olivia	Brown	Marketing	58000.00	
105	Noah	Davis	IT	72000.00	

email column with auto-generated company emails

Update the new column email with emailids

emp_id	first_name	last_name	department	salary	Email
101	John	Smith	HR	55000.00	Fn.ln@company.com
102	Emma	Johnson	Finance	62000.00	
103	Liam	Williams	IT	75000.00	
104	Olivia	Brown	Marketing	58000.00	
105	Noah	Davis	IT	72000.00	

email column with auto-generated company emails

```
UPDATE employees
```

```
SET email = CONCAT(LOWER(first_name), ' ', LOWER(last_name), '@company.com');
```

Trigger

A **trigger** is a function that is automatically executed in response to certain events on a particular table, like insert, update, or delete.

Trigger

```
CREATE TABLE Students (  
    StudentID INT  
    AUTO_INCREMENT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    Email VARCHAR(100),  
    LastUpdated TIMESTAMP  
    DEFAULT CURRENT_TIMESTAMP ON  
    UPDATE CURRENT_TIMESTAMP  
);
```

Trigger

```
DELIMITER //
```

```
CREATE TRIGGER UpdateLastUpdated  
BEFORE UPDATE ON Students  
FOR EACH ROW  
BEGIN  
    SET NEW.LastUpdated =  
CURRENT_TIMESTAMP;  
END //
```

```
DELIMITER ;
```

Trigger

Insert Sample Data

```
INSERT INTO Students (FirstName,  
LastName, Email) VALUES  
( 'John', 'Doe', 'john.doe@example.com' ),  
( 'Jane', 'Smith',  
'jane.smith@example.com' );
```

Update a Student Record

```
UPDATE Students  
SET Email =  
'john.doe@newdomain.com'  
WHERE StudentID = 1;
```

Trigger

Verify the Update

```
SELECT * FROM Students;
```

Transaction

A **transaction** is a group of operations that are executed as a single unit.

You can either **commit** these operations to the database or **rollback** if something goes wrong.

Transaction – Example: Bank Transfer

```
CREATE TABLE Accounts (  
    AccountID INT  
    AUTO_INCREMENT PRIMARY KEY,  
    AccountHolder  
    VARCHAR(100),  
    Balance DECIMAL(10, 2)  
);
```

Transaction – Example: Bank Transfer

```
INSERT INTO Accounts  
  (AccountHolder, Balance)  
VALUES  
  ('Alice', 500.00),  
  ('Bob', 300.00);
```

Transaction – Example: Bank Transfer

Transaction Example: Money Transfer

- Deducting £100 from Alice's balance.
- Adding £100 to Bob's balance.

Transaction - Example

Start a Transaction

```
START TRANSACTION;
```

Transaction - Example

Deduct from Alice's Account

```
UPDATE Accounts  
SET Balance = Balance - 100  
WHERE AccountHolder = 'Alice';
```

Transaction - Example

Add to Bob's Account

```
UPDATE Accounts  
SET Balance = Balance + 100  
WHERE AccountHolder = 'Bob';
```

Transaction - Example

Commit the Transaction

COMMIT;

Rollback

ROLLBACK;

Transaction - Example

```
START TRANSACTION;
```

```
UPDATE Accounts  
SET Balance = Balance - 100  
WHERE AccountHolder = 'Alice';
```

```
UPDATE Accounts  
SET Balance = Balance + 100  
WHERE AccountHolder = 'Bob';
```

```
-- Check for sufficient funds  
IF (SELECT Balance FROM Accounts WHERE AccountHolder = 'Alice') < 0 THEN  
    ROLLBACK;  
ELSE  
    COMMIT;  
END IF;
```

Indexes

An **index** improves the speed of data retrieval by creating a pointer to the data in a table.