01

ADVANCED DATABASE SYSTEMS

MySQL Stored Procedures:

Control Flow and Transactions

Control Statements, Error Handling, Transactions, and Modifications

Presented By: ***Solis, Rein (Leader)***
Taporco, Trixie Mae
Reyes, Charles
Ahron
Roquiza, Nathaniel
Ylagan, Janelle

Presentation Date: 05/06/25

GROUP #10

Table of Contents

What are Stored Procedures?

Stored Procedures are precompiled SQL routines stored in the database that can be called repeatedly to perform specific tasks such as queries, updates, or complex business logic. They allow parameter inputs, control flow (IF, CASE, loops), error handling, and transaction management.

Benifits:

- Improves performance by reducing redundant SQL parsing
- Enhances security and access control
- Encapsulates logic for reuse and consistency

Tables used: employees account_transactions

Employee Table

QUERY >>>

```
CREATE TABLE
employees (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  salary DECIMAL(10,2) NOT NULL,
  department VARCHAR(50),
  hire_date DATE
);

INSERT INTO
  employees (name, salary, department, hire_date)
VALUES
  ('John Smith', 75000.00, 'IT', '2020-01-15'),
  ('Sarah Johnson', 55000.00, 'HR', '2019-05-22'),
  ('Michael Brown', 48000.00, 'Sales', '2021-03-10'),
  ('Emily Davis', 62000.00, 'IT', '2018-11-05'),
  ('David Wilson', 52000.00, 'Marketing', '2022-02-18');
```

Table >>>

Table: employees

id	name	salary	department	hire_date
1	John Smith	75000.00	IT	2020-01-15
2	Sarah Johnson	55000.00	HR	2019-05-22
3	Michael Brown	48000.00	Sales	2021-03-10
4	Emily Davis	62000.00	IT	2018-11-05
5	David Wilson	52000.00	Marketing	2022-02-18

PRESENTATION

Account_Transactions

QUERY >>>

```
CREATE TABLE
account_transactions (
  transaction_id INT PRIMARY KEY AUTO_INCREMENT,
  account_id INT NOT NULL,
  amount DECIMAL(10,2) NOT NULL,
  transaction_type ENUM('deposit', 'withdrawal') NOT NULL,
  transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  status VARCHAR(20) DEFAULT 'pending'
);

INSERT INTO
  account_transactions (account_id, amount, transaction_type, status)
VALUES
  (101, 500.00, 'deposit', 'completed'),
  (102, 300.00, 'withdrawal', 'completed'),
  (101, 200.00, 'withdrawal', 'completed'),
  (103, 1000.00, 'deposit', 'completed'),
  (102, 150.00, 'deposit', 'completed');
```

Table >>>

account_transactions Table (Sample Output)

transaction_id	account_id	amount	transaction_type	transaction_date	status
1	101	500.00	deposit	2025-05-04 10:35:00[1]	completed
2	102	300.00	withdrawal	2025-05-04 10:35:00[1]	completed
3	101	200.00	withdrawal	2025-05-04 10:35:00[1]	completed
4	103	1000.00	deposit	2025-05-04 10:35:00[1]	completed
5	102	150.00	deposit	2025-05-04 10:35:00[1]	completed

PRESENTATION

Topic #1

Control Flow Statements

Conditional Statements:

- IF, ELSE, ELSEIF
- CASE

Loops:

- WHILE
- LOOP
- REPEAT

Control Statements (IF CASE)

Creating a Stored Procedure with an IF Statement

Stored Procedure for checking if the employee is in the IT department.

```
-- The delimiter is to change the ; into //, for creating statements in our stored procedure
DELIMITER //
-- IN means that this takes a parameter to be used in the procedure
CREATE PROCEDURE CheckIfEmployeeIT(IN emp_name VARCHAR(64))
BEGIN
  DECLARE emp_salary DECIMAL(10, 2);
  DECLARE emp_department VARCHAR(50);

  SELECT salary, department INTO emp_salary, emp_department FROM employees WHERE name = emp_name;

  IF emp_department = 'IT' THEN
    SELECT CONCAT ('Name: ', emp_name, ' - Salary: ', emp_salary, ' - Department: ', emp_department) as result;
  ELSE
    SELECT CONCAT (emp_name, ' not found in IT department.') as error;
  END IF;
END //
DELIMITER ;
```

Calling the Stored Procedure:

```
CALL CheckIfEmployeeIT('John Smith');
```

Checking Stored Procedure parameters:

```
SELECT * FROM information_schema.parameters WHERE SPECIFIC_NAME = 'CheckIfEmployeeIT';
```

Stored Procedure for checking the salary of the employee.

```
DELIMITER //
CREATE PROCEDURE CheckSalary(IN emp_id INT)
BEGIN
  DECLARE emp_salary DECIMAL(10,2);
  DECLARE emp_name VARCHAR(100);

  SELECT salary, name INTO emp_salary, emp_name FROM employees WHERE id = emp_id;

  IF emp_salary > 60000 THEN
    SELECT CONCAT(emp_name, ' has a high salary: $', emp_salary) AS message;
  ELSEIF emp_salary > 50000 THEN
    SELECT CONCAT(emp_name, ' has a medium salary: $', emp_salary) AS message;
  ELSE
    SELECT CONCAT(emp_name, ' has a low salary: $', emp_salary) AS message;
  END IF;

END //
DELIMITER ;
```

Showing the Stored Procedure:

```
SHOW PROCEDURE STATUS;
```

Dropping the Stored Procedure:

```
DROP PROCEDURE CheckSalary;
```

Using the Stored Procedure:

```
CALL CheckSalary(1);
```

Creating a Stored Procedure with CASE Statement

Stored Procedure for checking the department of the Employee

```
DELIMITER //
CREATE PROCEDURE GetEmployeeDepartment(IN emp_id INT)
  BEGIN
  DECLARE emp_dept VARCHAR(64);
  DECLARE result VARCHAR(64);

  SELECT department INTO emp_dept FROM employees WHERE id = emp_id;

  SET result = CASE
    WHEN emp_dept = 'IT' THEN 'Employee belongs to the IT department.'
    WHEN emp_dept = 'HR' THEN 'Employee belongs to the HR department.'
    WHEN emp_dept = 'Sales' THEN 'Employee belongs to the Sales department.'
    WHEN emp_dept = 'Marketing' THEN 'Employee belongs to the Marketing department.'
    ELSE 'New Hire'
  END;

  SELECT result;

  END //
DELIMITER ;
```

Stored Procedure for checking the department of the Employee

```
DELIMITER //
CREATE PROCEDURE GetEmployeeDepartment(IN emp_id INT)
  BEGIN
  DECLARE emp_dept VARCHAR(64);
  DECLARE result VARCHAR(64);

  SELECT department INTO emp_dept FROM employees WHERE id = emp_id;

  SET result = CASE
    WHEN emp_dept = 'IT' THEN 'Employee belongs to the IT department.'
    WHEN emp_dept = 'HR' THEN 'Employee belongs to the HR department.'
    WHEN emp_dept = 'Sales' THEN 'Employee belongs to the Sales department.'
    WHEN emp_dept = 'Marketing' THEN 'Employee belongs to the Marketing department.'
    ELSE 'New Hire'
  END;

  SELECT result;

  END //
DELIMITER ;
```

Stored Procedure for checking the level of the Employee

```
DELIMITER //
CREATE PROCEDURE GetEmployeeLevel(IN emp_id INT)
  BEGIN
  DECLARE emp_years INT;
  DECLARE emp_level VARCHAR(20);

  SELECT TIMESTAMPDIFF(YEAR, hire_date, CURDATE()) INTO emp_years
  FROM employees WHERE id = emp_id;

  SET emp_level = CASE
    WHEN emp_years >= 5 THEN 'Senior'
    WHEN emp_years >= 3 THEN 'Mid-level'
    WHEN emp_years >= 1 THEN 'Junior'
    ELSE 'New Hire'
  END;

  SELECT CONCAT('Employee #', emp_id, ' is a ', emp_level, ' employee') AS result;

  END //
DELIMITER ;
```

Using the Stored Procedure: CALL GetEmployeeLevel(4);

Creating a Stored Procedure with a WHILE LOOP

Get first 3 records in the employees table

```
DELIMITER //
CREATE PROCEDURE GetFirstThreeEmployees()
  BEGIN
  DECLARE count INT(11) DEFAULT 0;

  WHILE count < 3 DO
    SELECT * FROM employees LIMIT 1 OFFSET count;
    SET count = count + 1;
  END WHILE;

  END //
DELIMITER ;
```

Using the Stored Procedure with the WHILE LOOP

```
CALL GetFirstThreeEmployees();
```

Raise all IT employees a 5% raise until average salary reaches 65,000

```
DELIMITER //
CREATE PROCEDURE RaiseITSalaries()
  BEGIN
  DECLARE avg_salary DECIMAL(10,2);

  -- Calculate initial average
  SELECT AVG(salary) INTO avg_salary FROM employees WHERE department = 'IT';

  WHILE avg_salary < 65000 DO
    -- Give 5% raise
    UPDATE employees
    SET salary = salary * 1.05
    WHERE department = 'IT';

    -- Recalculate average
    SELECT AVG(salary) INTO avg_salary FROM employees WHERE department = 'IT';

    SELECT CONCAT('New average IT salary: $', ROUND(avg_salary, 2)) AS message;
  END WHILE;

  SELECT CONCAT ('Target average salary reached! New Average: ', avg_salary) AS final_message;
END //
DELIMITER ;
```

Using the Stored Procedure with a WHILE LOOP:

```
CALL RaiseITSalaries();
```

Creating a Stored Procedure with a WHILE LOOP

Get first 3 records in the employees table

```
DELIMITER //
CREATE PROCEDURE GetFirstThreeEmployees()
  BEGIN
  DECLARE count INT(11) DEFAULT 0;

  WHILE count < 3 DO
    SELECT * FROM employees LIMIT 1 OFFSET count;
    SET count = count + 1;
  END WHILE;

  END //
DELIMITER ;
```

Using the Stored Procedure with the WHILE LOOP

```
CALL GetFirstThreeEmployees();
```

Creating a Stored Procedure with a LOOP

Getting the employee names and salaries until it is less than 50000

```
DELIMITER //
CREATE PROCEDURE PrintEmployeesUntilLowSalary()
BEGIN
  DECLARE counter INT DEFAULT 1;
  DECLARE emp_name VARCHAR(100);
  DECLARE emp_salary DECIMAL(10,2);

  my_loop: LOOP
      SELECT name, salary INTO emp_name, emp_salary
      FROM employees WHERE id = counter;

      IF emp_salary < 50000 THEN
          SELECT CONCAT('Stopping at ', emp_name, ' with low salary: $', emp_salary) AS message;
          LEAVE my_loop;
      END IF;

      SELECT CONCAT('Processing: ', emp_name, ' ($', emp_salary, ')') AS message;

      SET counter = counter + 1;

      IF counter > (SELECT COUNT(*) FROM employees) THEN
          LEAVE my_loop;
      END IF;
  END LOOP;
END //
DELIMITER ;
```

Using the Stored Procedure with LOOP:

```
CALL RaiseITSalaries();
```

Creating a Stored Procedure with REPEAT

```
DELIMITER //
CREATE PROCEDURE GiveBonusesUntilCondition()
BEGIN
  DECLARE high_earners INT;

  REPEAT
    UPDATE employees
    SET salary = salary + 1000
    WHERE salary = (SELECT MIN(salary) FROM employees);

    SELECT COUNT(*) INTO high_earners FROM employees WHERE salary > 60000;

    SELECT CONCAT('High earners count: ', high_earners) AS message;

    UNTIL high_earners >= 3
  END REPEAT;

  SELECT 'At least 3 high earners now!' as final_message;
END //
DELIMITER ;
```

Error Handling using DECLARE CONTINUE HANDLER

```
DELIMITER //
CREATE PROCEDURE GetEmployeeSalary(IN emp_id INT)
BEGIN
  DECLARE emp_salary DECIMAL(10,2);
  DECLARE emp_name VARCHAR(100);

  DECLARE CONTINUE HANDLER FOR NOT FOUND
  BEGIN
    SELECT CONCAT('Employee ID ', emp_id, ' not found') AS message;
  END;

  SELECT name, salary INTO emp_name, emp_salary
  FROM employees WHERE id = emp_id;

  IF emp_name IS NOT NULL THEN
    SELECT CONCAT(emp_name, ' earns $', emp_salary) AS salary_info;
  END IF;
END //
DELIMITER ;
```

Using the GetEmployeeSalary procedure:

CALL GetEmployeeSalary(2); -- Exists (Sarah Johnson)
CALL GetEmployeeSalary(99); -- Doesn't exist

Error handling specific SQL Errors

```
DELIMITER //
CREATE PROCEDURE AddEmployee(
  IN new_id INT,
  IN new_name VARCHAR(100),
  IN new_salary DECIMAL(10,2),
  IN new_dept VARCHAR(50)
)
BEGIN
  -- Handlers for specific error codes

  -- Duplicate key error
  DECLARE EXIT HANDLER FOR 1062
  BEGIN
    SELECT CONCAT('Error: Employee ID ', new_id, ' already exists') AS message;
  END;

  -- Attempt the insert
  INSERT INTO employees (id, name, salary, department, hire_date)
  VALUES (new_id, new_name, new_salary, new_dept, CURDATE());

  SELECT 'Employee added successfully!' AS message;
END //
DELIMITER ;
```

Using the AddEmployee procedure

```
CALL AddEmployee(6, 'Lisa Chen', 58000, 'IT'); -- Success
CALL AddEmployee(1, 'Duplicate', 50000, 'HR'); -- Fails (ID 1 exists)
```

Thank you For watching and Listening