



CSE 311L(Database Management System)

LAB-Week 08 (Lecture 1)

## Controlling User Access

### Topics:

- ▶ Creating Users
- ▶ Granting System Privileges
- ▶ What is a role?
- ▶ Creating and Granting Privileges to a Role
- ▶ Changing Password
- ▶ Granting Object Privileges
- ▶ Using WITH GRANT OPTION and PUBLIC key

### Creating Users

```
CREATE USER 'faisal'@'localhost'  
IDENTIFIED BY '1234'
```

### Granting System Privileges

- **ALL PRIVILEGES** – Grants all privileges to a user account.
- **CREATE** – The user account is allowed to create databases and tables.
- **DROP** - The user account is allowed to drop databases and tables.
- **DELETE** - The user account is allowed to delete rows from a specific table.
- **INSERT** - The user account is allowed to insert rows into a specific table.
- **SELECT** – The user account is allowed to read a database.
- **UPDATE** - The user account is allowed to update table rows.

### Granting ALL Privileges on a Database for a USER

```
GRANT ALL PRIVILEGES  
ON employee_information.*  
TO 'faisal'@'localhost'
```

### Revoke ALL Privileges on a Database for a USER

```
REVOKE ALL PRIVILEGES  
ON `employee_information`.*  
FROM 'faisal'@'localhost';
```

### Granting CERTAIN Privileges on a Database for a USER

```
GRANT SELECT, INSERT, DELETE ON  
employee_information.*  
TO faisal@'localhost'
```

### Revoke CERTAIN Privileges on a Database for a USER

```
REVOKE SELECT, DELETE  
ON `employee_information`.*  
FROM 'faisal'@'localhost';
```

### Display MySQL User Account Privileges

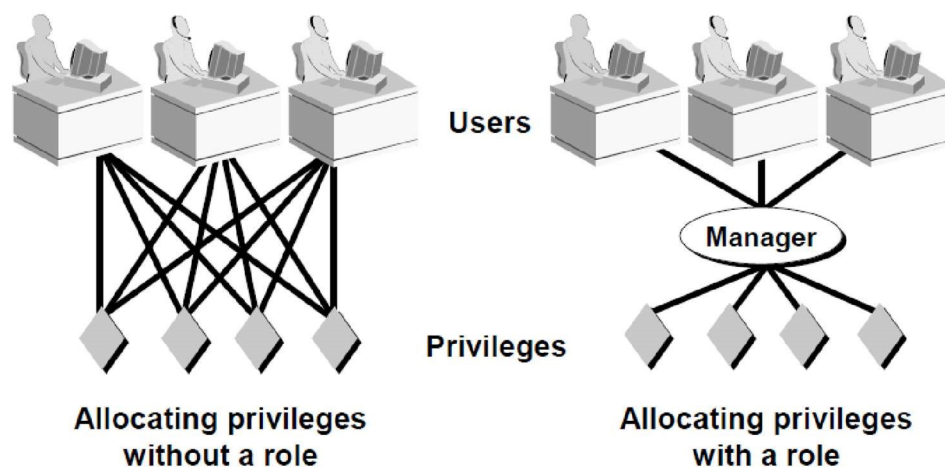
```
SHOW GRANTS FOR 'faisal'@'localhost';
```

### Remove a USER

```
DROP USER 'faisal'@'localhost'
```

What is a role?

Note: MySQL 8.0 has a working role implementation.



### Creating and Granting Privileges to a Role

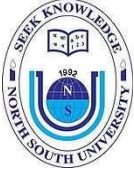
```
CREATE ROLE manager;
```

```
GRANT SELECT  
ON employee_information.*  
TO manager;
```

```
GRANT manager  
TO 'FAISAL'@'localhost'
```

## Changing Password

```
ALTER USER 'FAISAL'@'localhost'  
IDENTIFIED BY 4567;
```



CSE 311L(Database Management System)

LAB-Week 08 (Lecture 2)

## Triggers/Stored Procedure Examples & Demo

**First, create a new table named employees\_audit to keep the changes to the employees table:**

```
CREATE TABLE employees_audit (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    employee_ID INT NOT NULL,  
    lastname VARCHAR(50) NOT NULL,  
    changedat DATETIME DEFAULT NULL,  
    action VARCHAR(50) DEFAULT NULL  
);
```

### Trigger Example

Basic syntax of the CREATE TRIGGER statement:

**DELIMITER \$\$**

```
CREATE TRIGGER trigger_name  
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }  
ON table_name FOR EACH ROW  
BEGIN  
    trigger_body  
END$$;
```

The trigger body can access the values of the column being affected by the DML statement.

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

### Create a BEFORE UPDATE Trigger on employees

DELIMITER \$\$

```
CREATE TRIGGER before_employee_update
  BEFORE UPDATE ON employees
  FOR EACH ROW
BEGIN
  DECLARE action_re VARCHAR(20);
  SET action_re = 'BEFORE UPDATE';
  INSERT INTO employees_audit (`action`, `changedat`, `employee_ID`, `lastname`)
  VALUES
  (action_re, NOW(), OLD.employee_ID, OLD.last_name);
END$$
```

### Create a BEFORE UPDATE Trigger on employees: IF-ELSE Condition

```
DELIMITER $$
CREATE TRIGGER before_employee_update_detail_2
  BEFORE UPDATE ON employees
  FOR EACH ROW
BEGIN
  DECLARE action_re VARCHAR(50);

  IF(NEW.employee_ID <> OLD.employee_ID) THEN
    SET action_re = 'EMP ID UPDATED';
  ELSEIF(NEW.last_name <> OLD.last_name) THEN
    SET action_re = 'EMP Last Name Updated';
  ELSE SET action_re = 'Updated';
  END IF;

  INSERT INTO employees_audit (`action`, `changedat`, `employee_ID`, `lastname`)
  VALUES
  (action_re, NOW(), OLD.employee_ID, OLD.last_name);
END$$
DELIMITER $$
```

### DROP A Trigger

```
DROP TRIGGER IF EXISTS `before_employee_update`
```

## Stored Procedure Example

**Create a Procedure that returns all employee details:**

```
DELIMITER $$
```

```
CREATE PROCEDURE GETEMP()  
BEGIN  
SELECT * FROM employees;  
END$$
```

```
DELIMITER ;
```

## Stored Procedure Parameters

In MySQL, a parameter has one of three modes: IN,OUT, or INOUT.

- IN is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure.
- The value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program.
- An INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.

```
DELIMITER $$
```

```
CREATE PROCEDURE GETEMP_2(IN EMP_ID INT, OUT TOTAL_NUM INT)
```

```
BEGIN
```

```
DECLARE NUM INT;
```

```
SELECT COUNT(*)
```

```
INTO NUM
```

```
FROM employees;
```

```
SET TOTAL_NUM = NUM;
```

```
END$$
```

```
DELIMITER ;
```

### **Call the Procedures**

```
CALL GETEMP_2(100, @n);  
SELECT @n;
```

### **Stored Procedure [IF-ELSE]**

```
DELIMITER $$  
CREATE PROCEDURE CHECK_SALARY(IN EMP_ID1 INT, IN EMP_ID2 INT, OUT  
status_s varchar(80))  
BEGIN  
    DECLARE SALARY_1 DOUBLE;  
    DECLARE SALARY_2 DOUBLE;  
  
    SELECT SALARY  
    INTO SALARY_1  
    FROM employees  
    WHERE employees.Employee_Id = EMP_ID1;  
  
    SELECT SALARY  
    INTO SALARY_2  
    FROM employees  
    WHERE employees.Employee_Id = EMP_ID2;  
  
    IF (SALARY_1 = SALARY_2) THEN SET status_s = 'EQUAL';  
    ELSEIF (SALARY_1 > SALARY_2) THEN SET status_s = "EMP1 Has Higher Salary";  
    ELSE SET status_s = "EMP2 Has Higher Salary";  
    END IF;  
  
    END $$  
  
DELIMITER ;
```

### **Call The Procedures**

```
CALL CHECK_SALARY(101, 104, @out);  
SELECT @out;
```