**STORED PROCEDURE  
EX.**

**DELIMITER $$  
  
CREATE PROCEDURE AddUser ( IN p\_name varchar(100), IN p\_username varchar(100), IN p\_password varchar(100))s**

**BEGIN**

**INSERT INTO users (name,username,password) VALUES (p\_name,p\_username,p\_password);**

**END $$**

**DELIMITER ;**

**CALL AddUser('BSIT3', 'ITM04', 'YourPassword’);**

**TO DROP procedure  
  
DROP procedure procedure\_name;  
  
  
To Create Stored Procedure**

**EX. # 2**

**DELIMITER $$  
  
CREATE PROCEDURE GetAllUsers()**

**BEGIN**

**SELECT \* FROM users;**

**END $$**

**DELIMITER ;**

**CALL GetAllUsers()**

**To Show all procedure call:**

**Show procedure status where db="my\_db";**

**To View Procedure Codes**

**Show create procedure your\_procedure\_name;**

**Advance Stored Procedure  
EX. # 3  
DELIMITER &&**

**CREATE PROCEDURE UpdateSalary(**

**IN emp\_id INT,**

**IN new\_salary DECIMAL(10,2)**

**)**

**BEGIN**

**DECLARE current\_salary DECIMAL(10,2);**

**SELECT salary INTO current\_salary FROM employees WHERE id = emp\_id;**

**IF new\_salary > current\_salary THEN**

**UPDATE employees SET salary = new\_salary WHERE id = emp\_id;**

**ELSE**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'New salary must be higher than the current salary';**

**END IF;**

**END &&**

**DELIMITER ;  
  
OUT function  
DELIMITER //**

**CREATE PROCEDURE GetEmployeeSalary(**

**IN emp\_id INT,**

**OUT emp\_salary DECIMAL(10,2)**

**)**

**BEGIN**

**-- Get the employee's salary**

**SELECT salary INTO emp\_salary FROM employees WHERE id = emp\_id;**

**END //**

**DELIMITER ;  
  
  
  
CALL GetEmployeeSalary(1,@salary);**

**Select @salary as emp\_salary;**

**To Create Trigger**

**DELIMITER $$**

**CREATE TRIGGER after\_user\_insert**

**AFTER INSERT ON users**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO logs (DateInserted)**

**VALUES (NOW());**

**END $$**

**DELIMITER ;**

**TO DROP trigger**

**DROP TRIGGER trigger\_name;**

**To View All Trigger:  
Show triggers from your\_db;**

**To View Triiger Code:**

**SHOW CREATE TRIGGER trigger\_name;**

**BEFORE INSERT**

**DELIMITER //**

**CREATE TRIGGER before\_insert\_employee**

**BEFORE INSERT ON employees**

**FOR EACH ROW**

**BEGIN**

**IF NEW.salary < 10000 THEN**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'Salary must be at least 10,000';**

**END IF;**

**END //**

**DELIMITER ;**

**AFTER INSERT**

**DELIMITER //**

**CREATE TRIGGER after\_insert\_employee**

**AFTER INSERT ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO employee\_logs (employee\_id, action, action\_date)**

**VALUES (NEW.id, 'INSERT', NOW());**

**END //**

**DELIMITER ;**

**BEFORE UPDATE Trigger**

**DELIMITER //**

**CREATE TRIGGER before\_update\_salary**

**BEFORE UPDATE ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO salary\_history (employee\_id, old\_salary, updated\_at)**

**VALUES (OLD.id, OLD.salary, NOW());**

**END //**

**DELIMITER ;**

**AFTER UPDATE Trigger**

**DELIMITER //**

**CREATE TRIGGER after\_update\_salary**

**AFTER UPDATE ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO audit\_logs (employee\_id, old\_salary, new\_salary, changed\_at)**

**VALUES (OLD.id, OLD.salary, NEW.salary, NOW());**

**END //**

**DELIMITER ;  
  
BEFORE DELETE Trigger  
DELIMITER //**

**CREATE TRIGGER before\_delete\_employee**

**BEFORE DELETE ON employees**

**FOR EACH ROW**

**BEGIN**

**IF OLD.role = 'Manager' THEN**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'Managers cannot be deleted';**

**END IF;**

**END //**

**DELIMITER ;**

**AFTER DELETE Trigger**

**DELIMITER //**

**CREATE TRIGGER before\_delete\_employee**

**BEFORE DELETE ON employees**

**FOR EACH ROW**

**BEGIN**

**IF OLD.role = 'Manager' THEN**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'Managers cannot be deleted';**

**END IF;**

**END //**

**DELIMITER ;**

**AFTER DELETE Trigger**

**DELIMITER //**

**CREATE TRIGGER after\_delete\_employee**

**AFTER DELETE ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO deleted\_employees (id, name, deleted\_at)**

**VALUES (OLD.id, OLD.name, NOW());**

**END //**

**DELIMITER ;**

**How To Create a New Credentials**

**CREATE USER 'newuser'@'localhost' IDENTIFIED BY ‘yourpassword’;**

* **newuser is the username.**
* **localhost means user can connect to localmachine only.Alternate % for anyhost.**
* **yourpassword is the password for user.**

**How To Show and Delete Users**

**To Show**

**SELECT User, Host FROM mysql.user;  
  
To Delete or Drop**

**DROP USER 'username’@’localhost';**

**How Grant All Privileges on User With Specific Database**

**GRANT ALL PRIVILEGES ON my\_db.\* TO 'newuser'@'localhost’;**

**How to Grant Privileges on User with specific database and table.**

**GRANT SELECT,INSERT ON my\_db.users TO 'newuser'@'localhost’;**

**To show granted privileges of user**

**SHOW GRANTS FOR ‘newuser'@'localhost';**

**How Remove All Privileges on User With Specific Database**

**REVOKE ALL PRIVILEGES ON my\_db.\* FROM ‘newuser'@'localhost';**

**How to Remove Granted Privileges on User with specific database and table.**

**Revoke SELECT,INSERT ON my\_db.users FROM 'newuser'@'localhost’;**

**To ensure that changes are applied:**

**Privileges List**

**FLUSH PRIVILEGES;**

**ALL PRIVILEGES= Grants all available privileges.**

**SELECT = Allows reading data.**

**INSERT= Allows inserting rows into tables.**

**UPDATE= Allows modifying existing rows.**

**DELETE= Allows deleting rows.**

**CREATE= Allows creating new databases and tables.**

**DROP = Allows deleting databases and tables.**

**INDEX = Allows creating and removing indexes.**

**ALTER = Allows modifying table structure.**

**EXECUTE = Allows executing stored procedures/functions.**

**CREATE ROUTINE = Allows creating stored procedures and functions.**

**ALTER ROUTINE = Allows altering or dropping stored routines.**

**EVENT = Allows creating, altering, and dropping events.**

**TRIGGER = Allows creating and dropping triggers.**

**GRANT OPTION = Allows the user to grant privileges they have to others.**

**CREATE USER = Allows creating and managing users.**

**SHOW DATABASES = Allows seeing all databases (not just those the user has access to).**

**To export database**

**mysqldump -u [username] -p [database\_name] > [output\_file].sql**

**To Export Specific Database Table**

**mysqldump -u [username] -p [database\_name] [table\_name] > [output\_file].sql**

**How to Import Database**

**mysql -u [username] -p -e "CREATE DATABASE [database\_name];“**

**mysql -u [username] -p [database\_name] < [input\_file].sql**

**How to export table as csv file**

**SELECT \* FROM [table]**

**INTO OUTFILE ‘[path]/[filename].csv'**

**FIELDS TERMINATED BY ','**

**ENCLOSED BY '"'**

**LINES TERMINATED BY '\n';**

**How To IMPORT as CSV file:**

**LOAD DATA INFILE ‘[path]\[filename].csv'**

**INTO TABLE [tablename]**

**FIELDS TERMINATED BY ','**

**ENCLOSED BY '"'**

**LINES TERMINATED BY '\n'**

**IGNORE 1 LINES;**

**CREATING TABLE**

**-- Create a users table for our examples**

**CREATE TABLE users (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100) NOT NULL,**

**username VARCHAR(100) UNIQUE NOT NULL,**

**password VARCHAR(100) NOT NULL,**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**

**);**

**-- Create an employees table for trigger examples**

**CREATE TABLE employees (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100) NOT NULL,**

**salary DECIMAL(10,2) NOT NULL,**

**role VARCHAR(50),**

**department VARCHAR(50)**

**);**

**-- Create supporting tables for triggers**

**CREATE TABLE logs (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**DateInserted DATETIME**

**);**

**CREATE TABLE employee\_logs (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**employee\_id INT,**

**action VARCHAR(20),**

**action\_date DATETIME**

**);**

**CREATE TABLE salary\_history (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**employee\_id INT,**

**old\_salary DECIMAL(10,2),**

**updated\_at DATETIME**

**);**

**CREATE TABLE audit\_logs (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**employee\_id INT,**

**old\_salary DECIMAL(10,2),**

**new\_salary DECIMAL(10,2),**

**changed\_at DATETIME**

**);**

**CREATE TABLE deleted\_employees (**

**id INT,**

**name VARCHAR(100),**

**deleted\_at DATETIME**

**);**

**STORED PROCEDURE**

**-- Example 1: AddUser procedure**

**DELIMITER $$**

**CREATE PROCEDURE AddUser(**

**IN p\_name VARCHAR(100),**

**IN p\_username VARCHAR(100),**

**IN p\_password VARCHAR(100)**

**BEGIN**

**INSERT INTO users (name, username, password)**

**VALUES (p\_name, p\_username, p\_password);**

**END $$**

**DELIMITER ;**

**-- Example 2: GetAllUsers procedure**

**DELIMITER $$**

**CREATE PROCEDURE GetAllUsers()**

**BEGIN**

**SELECT \* FROM users;**

**END $$**

**DELIMITER ;**

**-- Example 3: UpdateSalary procedure with validation**

**DELIMITER $$**

**CREATE PROCEDURE UpdateSalary(**

**IN emp\_id INT,**

**IN new\_salary DECIMAL(10,2)**

**)**

**BEGIN**

**DECLARE current\_salary DECIMAL(10,2);**

**SELECT salary INTO current\_salary FROM employees WHERE id = emp\_id;**

**IF new\_salary > current\_salary THEN**

**UPDATE employees SET salary = new\_salary WHERE id = emp\_id;**

**ELSE**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'New salary must be higher than the current salary';**

**END IF;**

**END $$**

**DELIMITER ;**

**-- Example 4: GetEmployeeSalary with OUT parameter**

**DELIMITER $$**

**CREATE PROCEDURE GetEmployeeSalary(**

**IN emp\_id INT,**

**OUT emp\_salary DECIMAL(10,2)**

**)**

**BEGIN**

**SELECT salary INTO emp\_salary FROM employees WHERE id = emp\_id;**

**END $$**

**DELIMITER ;**

**TRIGGERS**

**-- AFTER INSERT trigger on users table**

**DELIMITER $$**

**CREATE TRIGGER after\_user\_insert**

**AFTER INSERT ON users**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO logs (DateInserted) VALUES (NOW());**

**END $$**

**DELIMITER ;**

**-- BEFORE INSERT trigger with validation**

**DELIMITER $$**

**CREATE TRIGGER before\_insert\_employee**

**BEFORE INSERT ON employees**

**FOR EACH ROW**

**BEGIN**

**IF NEW.salary < 10000 THEN**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'Salary must be at least 10,000';**

**END IF;**

**END $$**

**DELIMITER ;**

**-- AFTER INSERT trigger for logging**

**DELIMITER $$**

**CREATE TRIGGER after\_insert\_employee**

**AFTER INSERT ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO employee\_logs (employee\_id, action, action\_date)**

**VALUES (NEW.id, 'INSERT', NOW());**

**END $$**

**DELIMITER ;**

**-- BEFORE UPDATE trigger to save old salary**

**DELIMITER $$**

**CREATE TRIGGER before\_update\_salary**

**BEFORE UPDATE ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO salary\_history (employee\_id, old\_salary, updated\_at)**

**VALUES (OLD.id, OLD.salary, NOW());**

**END $$**

**DELIMITER ;**

**-- AFTER UPDATE trigger for audit**

**DELIMITER $$**

**CREATE TRIGGER after\_update\_salary**

**AFTER UPDATE ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO audit\_logs (employee\_id, old\_salary, new\_salary, changed\_at)**

**VALUES (OLD.id, OLD.salary, NEW.salary, NOW());**

**END $$**

**DELIMITER ;**

**-- BEFORE DELETE trigger with validation**

**DELIMITER $$**

**CREATE TRIGGER before\_delete\_employee**

**BEFORE DELETE ON employees**

**FOR EACH ROW**

**BEGIN**

**IF OLD.role = 'Manager' THEN**

**SIGNAL SQLSTATE '45000'**

**SET MESSAGE\_TEXT = 'Managers cannot be deleted';**

**END IF;**

**END $$**

**DELIMITER ;**

**-- AFTER DELETE trigger for archiving**

**DELIMITER $$**

**CREATE TRIGGER after\_delete\_employee**

**AFTER DELETE ON employees**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO deleted\_employees (id, name, deleted\_at)**

**VALUES (OLD.id, OLD.name, NOW());**

**END $$**

**DELIMITER ;**

**User Creation and Privileges**

**-- Create a new user with password**

**CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'yourpassword';**

**-- Grant all privileges on a specific database**

**GRANT ALL PRIVILEGES ON my\_db.\* TO 'newuser'@'localhost';**

**-- Grant specific privileges on a specific table**

**GRANT SELECT, INSERT ON my\_db.users TO 'newuser'@'localhost';**

**-- Show user privileges**

**SHOW GRANTS FOR 'newuser'@'localhost';**

**-- Revoke privileges**

**REVOKE ALL PRIVILEGES ON my\_db.\* FROM 'newuser'@'localhost';**

**-- Revoke specific privileges**

**REVOKE SELECT, INSERT ON my\_db.users FROM 'newuser'@'localhost';**

**-- Delete a user**

**DROP USER 'username'@'localhost';**

**-- Apply privilege changes**

**FLUSH PRIVILEGES;**