SQL TRIGGERS - Complete Learning Guide

Tamil & English with 10 Single-Line Practice Tasks

---

1. TRIGGERS - Definition & Concept | தூண்டிகள் - வரையறை

What is Triggers? | தூண்டிகள் என்றால் என்ன?

Definition: A trigger is a block of SQL code that automatically executes (fires) when specific database events occur on a table.

தமிழ்: தூண்டி என்பது ஒரு SQL குறியீட்டின் தொகுதி ஆகும், இது அட்டவணையில் குறிப்பிட்ட தரவுத்தளம் நிகழ்வுகள் நிகழும்போது தானாக செயல்பாடு செய்யும்.

Why We Use Triggers? | ஏன் பயன்படுத்துகிறோம்?

Purpose	விளக்கம்	Example
Security	தரவைப் பாதுகாக்க	Prevent invalid updates
Backup	தரவுப் பிரதிக்கள் உருவாக்க	Auto-backup old values
Auditing	மாற்றங்களைக் கண்காணிக்க	Log who changed what
Prevent Invalid Operations	தவறான செயல்பாடுகளைத் தடுக்க	No salary decrease
Enforce Business Rules	விதிகளைக் கட்டாயப்படுத்த	Min salary validation
Auto-fill Values	தானாகப் பூர்த்தி செய்ய	Auto timestamp

---

2. TRIGGER SYNTAX - Structure | தூண்டி வாக்கியமைப்பு

Basic Syntax | அடிப்படை வாக்கியமைப்பு

```sql
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements here
END;
```

Components Explanation | கூறுகளின் விளக்கம்

Component	விளக்கம்	Options
CREATE TRIGGER	தூண்டி உருவாக்கு	-
trigger_name	தூண்டியின் பெயர்	any_custom_name
BEFORE | AFTER	செயல்பாடுக்கு முன்/பின்	BEFORE / AFTER
INSERT | UPDATE | DELETE	நிகழ்வு வகை	INSERT, UPDATE, DELETE
ON table_name	எந்த அட்டவணையில்	employees, departments
FOR EACH ROW	ஒவ்வொரு வரிக்கு	(Required in MySQL)
NEW	புதிய மதிப்பு	Accessible in INSERT, UPDATE
OLD	பழைய மதிப்பு	Accessible in UPDATE, DELETE

Important Keywords | முக்கிய சொற்கள்

```
NEW  - புதிய மதிப்பு - Insert/Update பிறகு மதிப்பு
OLD  - பழைய மதிப்பு - Update/Delete முன் மதிப்பு

BEFORE - செயல்பாடு நடைபெறுவதற்கு முன்பு தூண்டு
AFTER  - செயல்பாடு நடைபெறுவதற்கு பின்னர் தூண்டு
```

---

3. 6 TRIGGER TYPES - All Combinations | 6 தூண்டி வகைகள்

Type 1: BEFORE INSERT | Insert முன் தூண்டி

```sql
CREATE TRIGGER before_emp_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    -- Validates data before insertion
    -- Can modify NEW values
    SET NEW.created_date = NOW();
END;
```

Purpose: Validate and modify data BEFORE it enters database

Type 2: AFTER INSERT | Insert பிறகு தூண்டி

```sql
```

```
CREATE TRIGGER after_emp_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    -- Actions after insert complete
    -- Cannot modify NEW (already inserted)
    INSERT INTO audit_log VALUES (NEW.emp_id, 'Employee Added', NOW());
END;
```

Purpose: Perform actions AFTER data is inserted (logging, stats update)

Type 3: BEFORE UPDATE | Update முன் தூண்டி

```sql
CREATE TRIGGER before_emp_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    -- Validate changes before update
    IF NEW.salary < OLD.salary THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary cannot decrease!';
    END IF;
END;
```

Purpose: Validate or prevent invalid updates

Type 4: AFTER UPDATE | Update பிறகு தூண்டி

```sql
CREATE TRIGGER after_emp_update
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    -- Log changes after update
    INSERT INTO salary_history (emp_id, old_salary, new_salary, change_date)
    VALUES (OLD.emp_id, OLD.salary, NEW.salary, NOW());
END;
```

Purpose: Log/track changes made to data

Type 5: BEFORE DELETE | Delete முன் தூண்டி

```sql
```

```
CREATE TRIGGER before_emp_delete
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
    -- Prevent deletion or archive
    IF OLD.salary > 100000 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete senior employees!';
    END IF;
END;
```

Purpose: Prevent dangerous deletions

Type 6: AFTER DELETE | Delete பிறகு தூண்டி

```sql
CREATE TRIGGER after_emp_delete
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
    -- Archive deleted data
    INSERT INTO archived_employees SELECT * FROM employees WHERE emp_id = OLD.emp_id;
    INSERT INTO audit_log VALUES (OLD.emp_id, 'Employee Deleted', NOW());
END;
```

Purpose: Archive/log deleted data

---

4. DELIMITER - Different Types | பிரிப்பான் - வகைகள்

Default Delimiter | பொதுவான பிரிப்பான்

```sql
-- Default: ; (Semicolon)
CREATE TRIGGER trigger_name
AFTER INSERT ON table_name
FOR EACH ROW
BEGIN
    SELECT 1;
END;  -- This ; ends the trigger
```

Custom Delimiters | தனிப்பயன் பிரிப்பான்கள்

```sql
```

```
-- Change delimiter to $$
DELIMITER $$

CREATE TRIGGER trigger_name
AFTER INSERT ON table_name
FOR EACH ROW
BEGIN
    SELECT 1;
    SELECT 2;
END$$  -- Now $$ marks the end

DELIMITER ;  -- Change back to ;

-- Change delimiter to //
DELIMITER //

CREATE TRIGGER trigger_name2
AFTER UPDATE ON table_name
FOR EACH ROW
BEGIN
    SELECT 1;
END//

DELIMITER ;

-- Change delimiter to ##
DELIMITER ##

CREATE TRIGGER trigger_name3
AFTER DELETE ON table_name
FOR EACH ROW
BEGIN
    INSERT INTO log VALUES (1);
END##

DELIMITER ;
```

Why Change Delimiter?

When trigger body contains `;` (semicolons), we need different delimiter to mark trigger end.

```
sql
```

```
DELIMITER $$

CREATE TRIGGER complex_trigger
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO log1 VALUES (1);      -- Statement 1 ends with ;
    INSERT INTO log2 VALUES (2);      -- Statement 2 ends with ;
    UPDATE stats SET count = count+1; -- Statement 3 ends with ;
END$$  -- Trigger ends with $$

DELIMITER ;
```

---

5. SINGLE LINE vs MULTI-LINE TRIGGERS | ஒற்றை வரி vs பல வரி

Single Line Trigger | ஒற்றை வரி தூண்டி

```sql
-- Only ONE statement inside BEGIN-END
CREATE TRIGGER single_line_trigger
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_log VALUES (NEW.emp_id, 'Added', NOW());
END;
```

Characteristics:

- Only one SQL statement
- Simpler and faster
- Easy to maintain
- Best for simple operations

Multi-Line Trigger | பல வரி தூண்டி

```sql
```

```
-- MULTIPLE statements inside BEGIN-END
CREATE TRIGGER multi_line_trigger
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_log VALUES (NEW.emp_id, 'Added', NOW());
    UPDATE dept_stats SET emp_count = emp_count + 1 WHERE dept_id = NEW.dept_id;
    INSERT INTO notification VALUES (NEW.emp_id, 'Welcome!', NOW());
    UPDATE company_totals SET total_salary = total_salary + NEW.salary;
END;
```

Characteristics:

- Multiple SQL statements
- Complex logic possible
- Uses IF, WHILE, etc.
- Requires DELIMITER change

---

6. SINGLE LINE TRIGGERS - 10 PRACTICAL TASKS | 10 பயிற்சிகள்

Setup Tables for Practice

```
sql
```

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_name VARCHAR(50),
    salary DECIMAL(10,2),
    dept_id INT,
    hire_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE audit_log (
    log_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_id INT,
    action VARCHAR(50),
    action_date TIMESTAMP
);

CREATE TABLE salary_history (
    history_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_id INT,
    old_salary DECIMAL(10,2),
    new_salary DECIMAL(10,2),
    change_date TIMESTAMP
);

CREATE TABLE employee_count (
    total_employees INT DEFAULT 0,
    last_updated TIMESTAMP
);
```

TASK 1: AFTER INSERT - Log New Employee

```sql
DELIMITER $$

CREATE TRIGGER task1_after_insert_log
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_log (emp_id, action, action_date) VALUES (NEW.emp_id, 'Employee Inserted', NOW());
END$$

DELIMITER ;

-- Test:
INSERT INTO employees (emp_name, salary, dept_id) VALUES ('Raj Kumar', 50000, 1);
-- Check: SELECT * FROM audit_log;
```

Purpose: दर्ज करें नए कर्मचारी को लॉग करें | Log every new employee insertion

---

TASK 2: BEFORE INSERT - Auto Timestamp

```sql
DELIMITER $$

CREATE TRIGGER task2_before_insert_timestamp
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    SET NEW.hire_date = NOW();
END$$

DELIMITER ;

-- Test:
INSERT INTO employees (emp_name, salary, dept_id) VALUES ('Priya Singh', 55000, 2);
-- Check: SELECT emp_name, hire_date FROM employees;
```

Purpose: Insert செய்யும்போது தானாக தேதி நிரப்ப | Automatically set hire_date to current time

---

TASK 3: AFTER UPDATE - Log Salary Changes

```sql
DELIMITER $$

CREATE TRIGGER task3_after_update_salary_log
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO salary_history (emp_id, old_salary, new_salary, change_date)
    VALUES (OLD.emp_id, OLD.salary, NEW.salary, NOW());
END$$

DELIMITER ;

-- Test:
UPDATE employees SET salary = 60000 WHERE emp_id = 1;
-- Check: SELECT * FROM salary_history;
```

Purpose: Update பின் சம்பள மாற்றங்களைக் குறிப்பிடுக | Track all salary changes

---

TASK 4: BEFORE UPDATE - Prevent Salary Decrease

sql

```sql
DELIMITER $$

CREATE TRIGGER task4_before_update_prevent_decrease
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
   IF NEW.salary < OLD.salary THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary cannot be decreased!';
   END IF;
END$$

DELIMITER ;

-- Test (Success):
UPDATE employees SET salary = 65000 WHERE emp_id = 1;

-- Test (Error):
UPDATE employees SET salary = 40000 WHERE emp_id = 1;
-- Error: Salary cannot be decreased!
```

Purpose: Update முன் சம்பளத்தைத் தணிக்கை செய்க | Validate salary doesn't decrease

---

TASK 5: BEFORE INSERT - Validate Positive Salary

sql

```
DELIMITER $$

CREATE TRIGGER task5_before_insert_validate_salary
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
  IF NEW.salary <= 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary must be positive!';
  END IF;
END$$

DELIMITER ;

-- Test (Success):
INSERT INTO employees (emp_name, salary, dept_id) VALUES ('Amit Patel', 50000, 1);

-- Test (Error):
INSERT INTO employees (emp_name, salary, dept_id) VALUES ('Invalid', -5000, 1);
-- Error: Salary must be positive!
```

Purpose: Insert முன் சம்பளம் சரிபார்க்க | Validate salary is positive

---

TASK 6: AFTER DELETE - Archive Deleted Employee

```
sql
```

```
CREATE TABLE archived_employees (
    archive_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_id INT,
    emp_name VARCHAR(50),
    salary DECIMAL(10,2),
    dept_id INT,
    deleted_date TIMESTAMP
);

DELIMITER $$

CREATE TRIGGER task6_after_delete_archive
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO archived_employees (emp_id, emp_name, salary, dept_id, deleted_date)
    VALUES (OLD.emp_id, OLD.emp_name, OLD.salary, OLD.dept_id, NOW());
END$$

DELIMITER ;

-- Test:
DELETE FROM employees WHERE emp_id = 1;
-- Check: SELECT * FROM archived_employees;
```

Purpose: Delete பிறகு நீக்கப்பட்ட தரவை சேமிக்க | Archive all deleted employees

---

TASK 7: AFTER INSERT - Update Employee Count

```
sql
```

```sql
DELIMITER $$

CREATE TRIGGER task7_after_insert_count
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   UPDATE employee_count SET total_employees = total_employees + 1, last_updated = NOW();
END$$

DELIMITER ;

-- Test:
INSERT INTO employees (emp_name, salary, dept_id) VALUES ('New Emp', 50000, 1);
-- Check: SELECT * FROM employee_count;
```

Purpose: Insert பிறகு மொத்த எண்ணிக்கையை அப்டேட் செய்க | Auto-update total employee count

---

TASK 8: AFTER DELETE - Decrease Employee Count

```sql
DELIMITER $$

CREATE TRIGGER task8_after_delete_count
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
   UPDATE employee_count SET total_employees = total_employees - 1, last_updated = NOW();
END$$

DELIMITER ;

-- Test:
DELETE FROM employees WHERE emp_id = 2;
-- Check: SELECT * FROM employee_count;
```

Purpose: Delete பிறகு மொத்த எண்ணிக்கையைக் குறைக்க | Auto-decrease total count on deletion

---

TASK 9: BEFORE INSERT - Convert Name to Uppercase

```sql
```

```
DELIMITER $$

CREATE TRIGGER task9_before_insert_uppercase_name
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    SET NEW.emp_name = UPPER(NEW.emp_name);
END$$

DELIMITER ;

-- Test:
INSERT INTO employees (emp_name, salary, dept_id) VALUES ('john smith', 50000, 1);
-- Check: SELECT emp_name FROM employees WHERE emp_name LIKE 'JOHN%';
-- Result: JOHN SMITH (uppercase)
```

Purpose: Insert முன் பெயரை பெரிய எழுத்தில் மாற்ற | Auto-convert names to uppercase

---

TASK 10: BEFORE UPDATE - Set Modified Timestamp

```sql
ALTER TABLE employees ADD COLUMN last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP;

DELIMITER $$

CREATE TRIGGER task10_before_update_modified_timestamp
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    SET NEW.last_modified = NOW();
END$$

DELIMITER ;

-- Test:
UPDATE employees SET salary = 75000 WHERE emp_id = 1;
-- Check: SELECT emp_name, salary, last_modified FROM employees;
```

Purpose: Update பிறகு மாற்றியமைக்கப்பட்ட தேதி அப்டேட் செய்க | Auto-update modification timestamp

---

7. PRACTICAL SUMMARY - All Single Line Triggers | சுருக்கம்

Task	Type	Event	Purpose
1	AFTER	INSERT	Log new employee
2	BEFORE	INSERT	Auto timestamp
3	AFTER	UPDATE	Log salary changes
4	BEFORE	UPDATE	Prevent salary decrease
5	BEFORE	INSERT	Validate positive salary
6	AFTER	DELETE	Archive deleted employee
7	AFTER	INSERT	Update employee count
8	AFTER	DELETE	Decrease employee count
9	BEFORE	INSERT	Convert to uppercase
10	BEFORE	UPDATE	Set modified timestamp

---

8. KEY POINTS TO REMEMBER | நினைவில் கொள்ள வேண்டிய விஷயங்கள்

Important Notes | முக்கிய குறிப்புகள்

1. **NEW vs OLD**
   - `NEW` = Available in INSERT, UPDATE (after values)
   - `OLD` = Available in UPDATE, DELETE (before values)
2. **BEFORE vs AFTER**
   - `BEFORE` = Modify data before operation
   - `AFTER` = Cannot modify, only log/audit
3. **Single vs Multi-Line**
   - Single = Faster, simple operations
   - Multi = Complex logic, multiple actions
4. **Delimiter**
   - Default `;` works for single statements
   - Change delimiter for multiple statements with `;`
5. **Performance**
   - Triggers add overhead
   - Use only when necessary

- Single-line better than multi-line

6. **Debugging**
    - Test with SELECT * to verify results
    - Check audit/log tables
    - Use simple statements for clarity

---

9. INTERVIEW QUESTIONS | நேர்காணல் கேள்விகள்

Q1: What is the difference between BEFORE and AFTER trigger?

Answer: BEFORE triggers execute before operation and can modify data. AFTER triggers execute after operation completes and cannot modify data.

Q2: What is the difference between NEW and OLD?

Answer: NEW contains new values (INSERT/UPDATE). OLD contains old values (UPDATE/DELETE).

Q3: Why do we use triggers?

Answer: Security, Auditing, Backup, Prevent invalid operations, Enforce business rules.

Q4: Can we use triggers for INSERT, UPDATE, DELETE?

Answer: Yes, all three events are supported.

Q5: What is a single-line trigger?

Answer: A trigger with only one SQL statement in BEGIN-END block.

Q6: How to view triggers?

```sql
SHOW TRIGGERS;
SHOW TRIGGERS FROM database_name;
DESC table_name;
```

Q7: How to drop a trigger?

```sql
DROP TRIGGER trigger_name;
DROP TRIGGER database_name.trigger_name;
```

Q8: Can triggers call other triggers?

Answer: Yes, cascading triggers are possible (one trigger can fire another).

10. COMMON MISTAKES & SOLUTIONS | பொதுவான பிழைகள்

Mistake	Problem	Solution
Using `;` in multi-line	Syntax error	Use DELIMITER $$
Modifying OLD values	Error	Only modify NEW in BEFORE
Modifying in AFTER	Error	Use AFTER for logging only
No WHERE in UPDATE	Updates all rows	Always specify WHERE
Performance issues	Slow queries	Minimize trigger logic
Infinite loops	Recursion	Avoid self-referencing

Quick Reference - Copy & Paste Templates | வார்ப்புருக்கள்

Template 1: AFTER INSERT Log

```sql
DELIMITER $$
CREATE TRIGGER trigger_name_after_insert
AFTER INSERT ON table_name
FOR EACH ROW
BEGIN
    INSERT INTO log_table VALUES (NEW.id, 'Inserted', NOW());
END$$
DELIMITER ;
```

Template 2: BEFORE UPDATE Validation

```sql
DELIMITER $$
CREATE TRIGGER trigger_name_before_update
BEFORE UPDATE ON table_name
FOR EACH ROW
BEGIN
    IF NEW.column < OLD.column THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error message';
    END IF;
END$$
DELIMITER ;
```

Template 3: AFTER DELETE Archive

sql

```sql
DELIMITER $$
CREATE TRIGGER trigger_name_after_delete
AFTER DELETE ON table_name
FOR EACH ROW
BEGIN
    INSERT INTO archive_table SELECT * FROM OLD_TABLE WHERE id = OLD.id;
END$$
DELIMITER ;
```

Master These 10 Tasks and You'll Be Ready for SQL Trigger Interview Questions!