Viva Exam SQL Explanation

i have my viva and practical exam tommorow of sppu se it and this is the code which i will be perfoming =

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name CHAR(50),
  salary INT
);

CREATE TABLE audit_log (
  message CHAR(100)
);

DELIMITER //

CREATE TRIGGER row_trigger
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
  INSERT INTO audit_log
  VALUES (CONCAT('Row Trigger - Inserted: ', NEW.name));
END;
//

SET @done = 0;

CREATE TRIGGER insert_simulated_statement_trigger
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
  IF @done = 0 THEN
    INSERT INTO audit_log
    VALUES ('Statement Trigger - Insert started on employees table');
    SET @done = 1;
  END IF;
END;
//

DELIMITER ;




SET @done = 0;
```

INSERT INTO employees VALUES
(1, 'Bhavesh', 30000),
(2, 'Raj', 35000);

SELECT * FROM audit_log;

explain each and every line of code with all possible question and answers asked for it during viva and all changes which can be asked to be done on codde ..

Here's a full explanation of your SQL trigger code with **line-by-line breakdown**, **possible viva questions**, and **variations/changes you might be asked to make** during your exam.

🔵 CODE EXPLANATION (LINE BY LINE)

```sql
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name CHAR(50),
  salary INT
);
```

- **Creates** a table named `employees`.
- It has 3 columns: `id` (integer, primary key), `name` (character field, fixed size 50), and `salary` (integer).
- `PRIMARY KEY` ensures each employee has a unique ID.

```sql
CREATE TABLE audit_log (
  message CHAR(100)
);
```

- Creates a simple `audit_log` table with a single column `message`.
- This will store audit messages from triggers.

```sql
DELIMITER //
```

- Changes the SQL command delimiter to `//` instead of `;`.
- Necessary to define the **body of a trigger** (or procedure) which contains multiple statements.

```sql
CREATE TRIGGER row_trigger
AFTER INSERT ON employees
FOR EACH ROW
```

```
BEGIN
  INSERT INTO audit_log
  VALUES (CONCAT('Row Trigger - Inserted: ', NEW.name));
END;
//
```

- **Trigger Name**: `row_trigger`
- **Trigger Timing**: `AFTER INSERT`
- **Trigger Level**: `FOR EACH ROW` → Executes for each row inserted.
- **Logic**: After inserting an employee, the trigger logs a message with the name inserted.

```
sql

SET @done = 0;
```

- Declares and initializes a **session variable** `@done` to 0.
- Used to control execution in the next trigger (statement-level simulation).

```
sql

CREATE TRIGGER insert_simulated_statement_trigger
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
  IF @done = 0 THEN
    INSERT INTO audit_log
    VALUES ('Statement Trigger - Insert started on employees table');
    SET @done = 1;
  END IF;
END;
//
```

- **Trigger Name**: `insert_simulated_statement_trigger`
- **Timing**: `BEFORE INSERT`
- **Level**: `FOR EACH ROW`
- Simulates a **statement-level trigger** using `@done` flag.
- Logs once **at the start** of the multi-row insert.

```
sql

DELIMITER ;
```

- Resets the delimiter back to `;`.

```
sql

SET @done = 0;
```

- Resets the flag so the "statement trigger" simulation works again.

```sql

INSERT INTO employees VALUES
(1, 'Bhavesh', 30000),
(2, 'Raj', 35000);
```

- Inserts two rows into the `employees` table.

```sql

SELECT * FROM audit_log;
```

- Displays the logs inserted by both triggers.

✅ OUTPUT:

message
Statement Trigger - Insert started on employees table
Row Trigger - Inserted: Bhavesh
Row Trigger - Inserted: Raj

🎓 IMPORTANT VIVA QUESTIONS & ANSWERS

Question	Answer
What is a trigger?	A trigger is a set of SQL statements that automatically executes when a specific database event occurs (INSERT, UPDATE, DELETE).
Difference between row-level and statement-level triggers?	Row-level executes for each row; statement-level executes once per SQL statement.
Can MySQL support statement-level triggers?	No, MySQL only supports row-level triggers. Statement-level is simulated using flags or counters.
What is the purpose of `DELIMITER //`?	It allows writing multi-line SQL blocks by temporarily changing the default `;` delimiter.
Why did we use `@done` variable?	To simulate a statement-level trigger by controlling the insert message to appear only once.
When does `AFTER INSERT` trigger execute?	It runs after the insert operation is committed.
Can we use `BEFORE INSERT` and `AFTER INSERT` together?	Yes, but they must perform different tasks and not conflict.
What does `NEW.name` represent?	It's a reference to the value being inserted into the `name` column during the INSERT operation.

🛠️ CHANGES THAT MAY BE ASKED

Task/Change	How to Do It
Change `AFTER` trigger to log salary as well	`VALUES (CONCAT('Row Trigger - Inserted: ', NEW.name, ', Salary: ', NEW.salary));`
Change the trigger to work on DELETE instead of INSERT	Replace `AFTER INSERT` with `AFTER DELETE` and use `OLD.name`
Modify audit_log to include timestamp	Add column `timestamp DATETIME DEFAULT CURRENT_TIMESTAMP`
Add UPDATE trigger to log salary changes	Use `BEFORE UPDATE` or `AFTER UPDATE` and compare `OLD.salary` with `NEW.salary`
Restrict insert if salary < 10000	Use `BEFORE INSERT` trigger and `SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary too low';`
Change CHAR to VARCHAR for name and message	Modify column types to `VARCHAR(50)` and `VARCHAR(100)` respectively for better storage efficiency

---

Would you like me to provide a **diagram of how these triggers work** visually (flowchart-style)?

explain this
estrict insert if salary < 10000

To **restrict INSERT if salary < 10000**, you can use a **BEFORE INSERT trigger** that checks the value of `NEW.salary`. If it's less than 10000, you stop the insert using the `SIGNAL` statement, which throws an error.

✅ Updated Code to Restrict Low Salaries

```sql

DELIMITER //

CREATE TRIGGER check_salary_before_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
  IF NEW.salary < 10000 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Salary too low. Must be at least 10000.';
  END IF;
END;
//

DELIMITER ;
```

🔍 Explanation

Line	What It Does
`BEFORE INSERT`	Runs the trigger **before** inserting data into `employees` table.
`IF NEW.salary < 10000`	Checks the incoming salary value.
`SIGNAL SQLSTATE '45000'`	This raises a custom error. `'45000'` is a generic state for user-defined exceptions.

Line	What It Does
`` `SET MESSAGE_TEXT` ``	Sets the error message shown to the user when salary is too low.

💥 Example

```sql
sql

INSERT INTO employees VALUES (3, 'Sneha', 8000);
```

🔴 Error Output:

```scss
scss

ERROR 1644 (45000): Salary too low. Must be at least 10000.
```

Would you like me to show a version of the full code with this restriction added?