**Assignment- Triggers - keys**

1. Write a SQL query that creates a trigger on a table.

Sol:

SQL query to create a trigger on a table:

CREATE TRIGGER trigger\_name

AFTER INSERT ON table\_name

FOR EACH ROW

BEGIN

-- your trigger logic here

END;

**Explanation:** In this example, the trigger is named trigger\_name and is set to fire after an INSERT operation is performed on the table\_name table. The FOR EACH ROW clause indicates that the trigger logic will be executed for each row that is affected by the INSERT operation.

You can replace the comment -- your trigger logic here with the actual SQL statements that you want the trigger to execute. These statements can reference the NEW and OLD pseudo rows to access the values of the columns affected by the trigger.

For example, Use a trigger to update a timestamp column whenever a new row is inserted:

CREATE TRIGGER update\_timestamp

AFTER INSERT ON my\_table

FOR EACH ROW

BEGIN

UPDATE my\_table SET created\_at = CURRENT\_TIMESTAMP WHERE id = NEW.id;

END;

**Explanation:** This trigger updates the created\_at column with the current timestamp whenever a new row is inserted into my\_table. The WHERE clause ensures that only the row that was just inserted is updated.

1. Write a SQL query that uses a trigger to update data in a table.

Sol:

CREATE TRIGGER update\_sales\_total

AFTER INSERT ON sales

FOR EACH ROW

BEGIN

UPDATE sales\_summary SET total\_sales = total\_sales + NEW.amount WHERE month = MONTH(NEW.date);

END;

**Explanation:** In this example, the trigger is named update\_sales\_total and is set to fire after an INSERT operation is performed on the sales table. The FOR EACH ROW clause indicates that the trigger logic will be executed for each row that is affected by the INSERT operation.

The trigger logic updates the total\_sales column in a separate sales\_summary table based on the amount and date columns in the sales table. Specifically, it adds the value of NEW.amount (the amount of the sale that was just inserted) to the total\_sales value for the corresponding month in the sales\_summary table. The MONTH() function is used to extract the month from the NEW.date column.

Note that this is just a simple example, and your specific use case may require a different trigger logic or table structure. The important thing is to make sure that the trigger logic is designed to achieve the desired behavior in response to the trigger event.

1. Give an example of a scenario where a trigger would be useful in SQL and explain how it can be implemented in a query.

Sol:

CREATE TRIGGER update\_order\_total

AFTER INSERT ON order\_items

FOR EACH ROW

BEGIN

UPDATE orders SET total\_price = total\_price + NEW.price WHERE id = NEW.order\_id;

END;

**Explanation:** In this example, the trigger is named update\_order\_total and is set to fire after an INSERT operation is performed on the order\_items table. The FOR EACH ROW clause indicates that the trigger logic will be executed for each row that is affected by the INSERT operation.

The trigger logic updates the total\_price column in the orders table based on the price and order\_id columns in the order\_items table. Specifically, it adds the value of NEW.price (the price of the new item that was just inserted) to the total\_price value for the corresponding order in the orders table, identified by the NEW.order\_id column.

This trigger ensures that the total\_price column in the orders table is always up-to-date and reflects the total cost of all items included in each order. It also helps to maintain referential integrity between the two tables by ensuring that any changes made to the order\_items table are reflected in the corresponding orders table.

1. Explain the difference between the AFTER and INSTEAD OF trigger operators in SQL.

Sol:

In SQL, there are two different types of trigger operators: AFTER and INSTEAD OF. The main difference between the two is the timing of when the trigger is executed relative to the triggering event.

An AFTER trigger is executed after the triggering event has already occurred. For example, an AFTER trigger that is set to fire after an INSERT operation on a table would be executed after the new record has already been added to the table. AFTER triggers are commonly used to perform additional actions on the data that was affected by the triggering event.

On the other hand, an INSTEAD OF trigger is executed instead of the triggering event. For example, an INSTEAD OF trigger that is set to fire instead of an INSERT operation on a view would be executed instead of adding a new record to the underlying table(s) that the view is based on. INSTEAD OF triggers are commonly used to modify or redirect the behavior of a triggering event.

To summarize, the main difference between the AFTER and INSTEAD OF trigger operators in SQL is the timing of when the trigger is executed relative to the triggering event. An AFTER trigger is executed after the event has occurred, while an INSTEAD OF trigger is executed instead of the event.

1. What is the purpose of the INSTEAD OF DELETE trigger operator in SQL?

Sol:

The purpose of the INSTEAD OF DELETE trigger operator in SQL is to allow you to define custom behavior for the DELETE operation on a view or table.

By default, when you issue a DELETE statement on a view or table, the corresponding row or rows are deleted from the underlying table(s). However, if you define an INSTEAD OF DELETE trigger on the view or table, you can override this default behavior and specify custom logic to be executed instead of the DELETE operation.

For example, you might use an INSTEAD OF DELETE trigger to perform additional checks or validation before allowing a row to be deleted, or to update related records in other tables to maintain referential integrity.

Here's an example of an INSTEAD OF DELETE trigger that prevents rows from being deleted from a table under certain conditions:

CREATE TRIGGER prevent\_delete

INSTEAD OF DELETE

ON my\_table

BEGIN

IF (SELECT COUNT(\*) FROM my\_table WHERE id = OLD.id AND status = 'locked') > 0 THEN

RAISE (ABORT, 'Cannot delete locked record');

ELSE

DELETE FROM my\_table WHERE id = OLD.id;

END IF;

END;

**Explanation:** In this example, the trigger is named prevent\_delete and is set to fire instead of a DELETE operation on the my\_table table. The OLD keyword is used to refer to the row or rows that would have been deleted if the trigger had not been defined.

The trigger logic checks whether the row that is being deleted has a status of 'locked'. If it does, the trigger raises an error using the RAISE statement and the ABORT option, which prevents the DELETE operation from proceeding. Otherwise, the trigger deletes the row using a standard DELETE statement.

This trigger ensures that rows with a status of 'locked' cannot be deleted from the table and provides custom error handling for this specific scenario.