Ex. No: 07	TRIGGER OPERATIONS
Date	12.03.2024

Objective:

To execute the given queries using Trigger Operations.

Description

TRIGGER:

Trigger operations are actions or events that initiate a specific process or set of actions within a system or program. These triggers can be based on various conditions such as time, user input, data changes, or external events. When a trigger is activated, it prompts the system to perform predefined operations or tasks automatically. This automation helps streamline workflows, improve efficiency, and ensure timely responses to critical events. Common examples of trigger operations include sending automated emails based on user actions, updating database records when certain conditions are met, or executing specific tasks in response to external API calls.

A trigger in SQL is a concise piece of code that runs automatically when certain events occur on a table. These events can be INSERT, UPDATE, or DELETE operations.

Triggers act as guardians, ensuring data integrity by responding to events and enforcing rules.

SYNTAX:

```
CREATE TRIGGER Trigger_Name
[BEFORE | AFTER] [INSERT | UPDATE | DELETE] ON Table_Name
[FOR EACH ROW | FOR EACH COLUMN]
 [trigger_body];
```

TYPES OF TRIGGERS:

BEFORE Triggers:

These triggers fire before the specified event (e.g., INSERT, UPDATE, or DELETE) takes place. Useful for validation or modification before data changes.

AFTER Triggers:

These triggers fire after the event completes successfully.Often used for logging, auditing, or additional actions.

Row-Level Triggers (FOR EACH ROW):

These triggers operate on each affected row individually.Commonly used for enforcing referential integrity or maintaining history.

Statement-Level Triggers (FOR EACH STATEMENT):

These triggers operate on the entire set of affected rows.Useful for bulk operations or aggregations.

Nested Triggers:

Some databases allow triggers to invoke other triggers.Use with caution to avoid infinite loops.

Instead Of Triggers:

These triggers replace the default action for a specific event.Often used with views or complex data modifications.

Questions:

1. **Create a BEFORE INSERT Trigger for the "User" Table that ensures that the passwords are at least 8 Characters.**

```
SQL> CREATE TRIGGER before_insert_password
  2  BEFORE INSERT ON User7_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF LENGTH(NEW.password) < 8 THEN
  6          SIGNAL SQLSTATE '45000'
  7              SET MESSAGE_TEXT = 'Password must be at least 8 characters';
  8      END IF;
  9  END //
 10  DELIMITER ;
 11
 12  /

Warning: Trigger created with compilation errors.

SQL> INSERT INTO User7_1032 (username, password) VALUES ('testuser', 'abc123');
INSERT INTO User7_1032 (username, password) VALUES ('testuser', 'abc123')
            *
ERROR at line 1:
ORA-04098: trigger 'URK22AI1032.CHECK_PASSWORD_LENGTH7' is invalid and failed
re-validation
```

2. **Create a BEFORE UPDATE Trigger for the "User" Table that does not allow email addresses to be null.**

```
SQL> CREATE TRIGGER before_update_email
  2  BEFORE UPDATE ON User7_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF NEW.email IS NULL THEN
  6          SIGNAL SQLSTATE '45000'
  7              SET MESSAGE_TEXT = 'Email address cannot be null';
  8      END IF;
  9  END //
 10  DELIMITER ;
 11  /

Warning: Trigger created with compilation errors.

SQL> UPDATE User7_1032
  2  SET email = 'newemail@example.com'
  3  WHERE user_id = 123; -- Replace with the actual user ID
  4
SQL>
SQL> /
WHERE user_id = 123; -- Replace with the actual user ID
                  *
```

3. **Create a BEFORE DELETE Trigger for the "User" Table that prevents the deletion of users with specific email domains (like "example.com").**

```
SP2-0734: unknown command beginning "DELIMITER ..." - rest of line ignored.
SQL> CREATE TRIGGER delete_specific
  2  BEFORE DELETE ON User7_1032
  3  FOR EACH ROW
  4  BEGIN
  5      DECLARE email_domain VARCHAR(255);
  6      SET email_domain = SUBSTRING_INDEX(OLD.email, '@', -1);
  7
  8      IF email_domain = 'example.com' THEN
  9          SIGNAL SQLSTATE '45000'
 10              SET MESSAGE_TEXT = 'Deletion of users with example.com email domain is not allowed';
 11      END IF;
 12  END //
 13  DELIMITER ;
 14  /

Warning: Trigger created with compilation errors.

SQL> DELETE FROM User7_1032 WHERE user_id = 123;
DELETE FROM User7_1032 WHERE user_id = 123
                             *
```

4. **Write an AFTER INSERT trigger to count number of new tuples inserted using each**

```
SQL> CREATE TRIGGER after_insert_count
  2  AFTER INSERT ON User7_1032
  3  FOR EACH ROW
  4  BEGIN
  5      UPDATE tuple_count
  6      SET total_tuples = total_tuples + 1;
  7  END //
  8  DELIMITER ;
  9  /

Warning: Trigger created with compilation errors.
```

```
SQL> SELECT total_tuples FROM tuple_count;

TOTAL_TUPLES
------------
           0
```

5. **Create an AFTER UPDATE Trigger for the "User" Table that signals when a user's email is changed.**

```
SQL> CREATE TRIGGER after_update_email
  2  AFTER UPDATE ON User7_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF NEW.email <> OLD.email THEN
  6          -- Log the email change (you can customize this part)
  7          INSERT INTO EmailChangeLog (user_id, old_email, new_email, change_timestamp)
  8          VALUES (OLD.user_id, OLD.email, NEW.email, NOW());
  9      END IF;
 10  END //
 11  DELIMITER ;
 12  /

Warning: Trigger created with compilation errors.

SQL> update User7_1032 set email="my@exaple.com" where userid=2;
```

6. **Create an AFTER DELETE Trigger for the "User" Table that signals when a user is deleted.**

```
SQL> create or replace trigger user_delete
  2  after delete on User7_1032
  3  for each row
  4  begin
  5  dbms_output.put_line('userhas been deleted');
  6  end;
  7  /

Trigger created.
```

```
SQL> delete from User_1032 where userid=21;

0 rows deleted.
```

7. **Create a BEFORE INSERT Trigger for the "Event" Table that ensures the event's date is in the future.**

```
SQL> CREATE OR REPLACE TRIGGER BeforeEventInsert
  2  BEFORE INSERT ON Event_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF :NEW.EventDate < SYSDATE THEN
  6          RAISE_APPLICATION_ERROR(-20001, 'Event date must be in the future.');
  7      END IF;
  8  END;
  9  /

Warning: Trigger created with compilation errors.
```

```
SQL> INSERT INTO Event_1032 (Name, Dates)
  2  VALUES ('Future Event', '2024-12-31');
INSERT INTO Event_1032 (Name, Dates)
            *
ERROR at line 1:
ORA-04098: trigger 'URK22AI1032.BEFOREEVENTINSERT' is invalid and failed
re-validation
```

8. **Create a BEFORE UPDATE Trigger for the "Event" Table that Ensures that the event's time is not set to before 7:00 AM (assuming you use 24-hour format for your Time column).**

```
SQL> CREATE OR REPLACE TRIGGER BeforeEventInsert
  2  BEFORE INSERT ON Event_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF :NEW.EventDate < SYSDATE THEN
  6          RAISE_APPLICATION_ERROR(-20001, 'Event date must be in the future.');
  7      END IF;
  8  END;
  9  /

Warning: Trigger created with compilation errors.

SQL> INSERT INTO Event_1032 (Dates, Name)
  2  VALUES (TO_DATE('2024-03-15', 'YYYY-MM-DD'), 'Future Event');
INSERT INTO Event_1032 (Dates, Name)
            *
ERROR at line 1:
ORA-04098: trigger 'URK22AI1032.BEFOREEVENTINSERT' is invalid and failed
re-validation
```

9. **Create an AFTER DELETE Trigger for the "Event" Table that signals when an event is deleted.**

```
SQL> CREATE OR REPLACE TRIGGER AfterEventDelete
  2  AFTER DELETE ON Event_1032
  3  FOR EACH ROW
  4  BEGIN
  5      DBMS_OUTPUT.PUT_LINE('Event deleted: ' || :OLD.EventName || ' on ' || TO_CHAR(SYSDATE, 'YYYY-MM-DD HH24:MI:SS'));
  6      -- You can add additional actions here if needed.
  7  END;
  8  /

Warning: Trigger created with compilation errors.

SQL> DELETE FROM Event_1032 WHERE EventName = 'Future Event';
DELETE FROM Event_1032 WHERE EventName = 'Future Event'
                             *
ERROR at line 1:
ORA-00904: "EVENTNAME": invalid identifier
```

10. **Create an AFTER UPDATE Trigger for the "Event" Table that signals when an event's time is changed.**

```
SQL> CREATE OR REPLACE TRIGGER AfterEventUpdate
  2  AFTER UPDATE ON Event_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF :NEW.EventDate <> :OLD.EventDate THEN
  6          DBMS_OUTPUT.PUT_LINE('Event time changed for event: ' || :NEW.EventName);
  7          -- You can add additional actions here if needed.
  8      END IF;
  9  END;
 10  /

Warning: Trigger created with compilation errors.

SQL> UPDATE Event_1032
  2  SET EventDate = TO_DATE('2024-03-20', 'YYYY-MM-DD')
  3  WHERE EventName = 'Future Event';
WHERE EventName = 'Future Event'
```

11. **Create a BEFORE INSERT Trigger for the "Venue" Table that ensures the name of the venue is not empty.**

```
QL> CREATE TRIGGER check_venue_name
  2  BEFORE INSERT ON Venue_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF NEW.name = '' OR NEW.name IS NULL THEN
  6          SIGNAL SQLSTATE '45000'
  7          SET MESSAGE_TEXT = 'Venue name cannot be empty.';
  8      END IF;
  9  END;
 10  /

arning: Trigger created with compilation errors.

SQL> -- Insert a row with a non-empty venue name
SQL> INSERT INTO Venue_1032 (name) VALUES ('Example Venue');
INSERT INTO Venue_1032 (name) VALUES ('Example Venue')
            *
ERROR at line 1:
ORA-04098: trigger 'URK22AI1032.CHECK_VENUE_NAME' is invalid and failed
re-validation
```

12. **Create a BEFORE DELETE Trigger for the "Venue" Table that Prevents deletion if the VenueID is less than 105.**

```
SQL> CREATE TRIGGER prevent_venue_deletion
  2  BEFORE DELETE ON Venue_1032
  3  FOR EACH ROW
  4  BEGIN
  5      IF OLD.VenueID < 105 THEN
  6          SIGNAL SQLSTATE '45000'
  7          SET MESSAGE_TEXT = 'Deletion of venues with VenueID less than 105 is not allowed.';
  8      END IF;
  9  END;
 10  /

Warning: Trigger created with compilation errors.

SQL> -- Delete a row with VenueID equal to or greater than 105
SQL> DELETE FROM Venue_1032 WHERE VenueID = 105;
DELETE FROM Venue_1032 WHERE VenueID = 105
            *
ERROR at line 1:
ORA-04098: trigger 'URK22AI1032.PREVENT_VENUE_DELETION' is invalid and failed
re-validation
```

13. **Create an AFTER INSERT Trigger for the "Venue" Table that signals when a new row is added to it.**

```
SQL> CREATE TRIGGER notify_venue_insert
  2  AFTER INSERT ON Venue_1032
  3  FOR EACH ROW
  4  BEGIN
  5      SIGNAL SQLSTATE '00000'
  6      SET MESSAGE_TEXT = 'A new row has been added to Venue_1032.';
  7  END;
  8  /

Warning: Trigger created with compilation errors.

SQL> -- Insert a new row into Venue_1032
SQL> INSERT INTO Venue_1032 (name, address) VALUES ('Example Venue', '123 Main St');
INSERT INTO Venue_1032 (name, address) VALUES ('Example Venue', '123 Main St')
            *
ERROR at line 1:
ORA-04098: trigger 'URK22AI1032.CHECK_VENUE_NAME' is invalid and failed
re-validation
```

14. **Create an AFTER UPDATE Trigger for the "Venue" Table that signals when a row is updated.**

```
SQL> CREATE TRIGGER notify_venue_update
  2  AFTER UPDATE ON Venue_1032
  3  FOR EACH ROW
  4  BEGIN
  5      SIGNAL SQLSTATE '00000'
  6      SET MESSAGE_TEXT = 'A row has been updated in Venue_1032.';
  7  END;
  8  /

Warning: Trigger created with compilation errors.
```

```
SQL> update Venue_1032
  2  set name='new name',
  3  VENUEID=101
  4  WHERE VENUEID=123;
update Venue_1032
       *
ERROR at line 1:
ORA-04098: trigger 'URK22AI1032.NOTIFY_VENUE_UPDATE' is invalid and failed
re-validation
```

Result:

The given Queries using triggers were executed Successfully.