UNIT 5
Database Triggers

(5 Hours) — Database Programming (CACS453)

1. Introduction to Database Triggers

A database trigger is a stored PL/SQL block that Oracle executes automatically — "fires" — in response to a specified DML event (INSERT, UPDATE, or DELETE) or certain DDL or database events, occurring on a table or view. Unlike a procedure, a trigger is never called explicitly by the programmer; it runs implicitly whenever its triggering event occurs, making it useful for enforcing business rules, auditing changes, maintaining derived data, and replicating changes across tables.

bcalink.com

1.1 Why Use Triggers?

- Enforce complex business rules that cannot be expressed with simple constraints (e.g. CHECK or FOREIGN KEY).
- Audit data changes automatically, recording who changed what and when.
- Maintain derived or redundant data, such as keeping a summary or audit table in sync with a base table.
- Enable updates through views that would not otherwise be directly updatable (via INSTEAD OF triggers).
- Prevent invalid transactions, such as DML operations outside business hours.

1.2 General Trigger Syntax

Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
  {BEFORE | AFTER | INSTEAD OF}
  {INSERT | UPDATE [OF column_list] | DELETE} [OR ...]
  ON table_name
  [REFERENCING OLD AS old NEW AS new]
  [FOR EACH ROW]
  [WHEN (condition)]
BEGIN
  -- trigger body: PL/SQL statements
END;
/
```

A trigger definition is built from three independent choices, which together determine exactly when and how the trigger fires:

Dimension	Options	Determines
Timing	BEFORE, AFTER, INSTEAD OF	Whether the trigger fires before, after, or in place of the triggering statement
Event	INSERT, UPDATE, DELETE (or combinations with OR)	Which DML operation causes the trigger to fire
Level	FOR EACH ROW (row-level) or omitted (statement-level)	Whether the trigger fires once per affected row, or once per statement

bcalink.com

2. BEFORE and AFTER Triggers

The timing clause — BEFORE or AFTER — determines whether the trigger body executes before or after the triggering DML statement actually modifies the table.

2.1 BEFORE Triggers

A BEFORE trigger fires before the triggering DML operation is applied to the table. It is typically used to validate or modify data before it is written, since the underlying row has not yet been changed when the trigger runs — for row-level triggers, this also means the new column values can still be changed via the :NEW qualifier before they are actually stored.

BEFORE INSERT Trigger — Example

```
CREATE OR REPLACE TRIGGER trg_emp_before_insert
  BEFORE INSERT ON employees
  FOR EACH ROW
BEGIN
  IF :NEW.salary IS NULL THEN
    :NEW.salary := 0;  -- supply a default before the row is inserted
  END IF;
  :NEW.hire_date := NVL(:NEW.hire_date, SYSDATE);
END;
/
```

2.2 AFTER Triggers

An AFTER trigger fires after the triggering DML operation has already been applied to the table. It is typically used for auditing, logging, or cascading changes to other tables, since by the time it runs the change is already committed to the current transaction and column values can be read but not modified.

bealink.com

AFTER UPDATE Trigger — Example

```
CREATE OR REPLACE TRIGGER trg_emp_salary_audit
  AFTER UPDATE OF salary ON employees
  FOR EACH ROW
BEGIN
  INSERT INTO salary_audit (emp_id, old_salary, new_salary, changed_on)
  VALUES (:OLD.emp_id, :OLD.salary, :NEW.salary, SYSDATE);
END;
/
```

Aspect	BEFORE Trigger	AFTER Trigger
Fires	Before the row/statement change is applied	After the row/statement change has been applied
:NEW column values	Can be read and modified	Can be read only, not modified
Typical Use	Validation, defaulting, data cleansing	Auditing, logging, cascading updates to other tables

Aspect	BEFORE Trigger	AFTER Trigger
Performance	Slightly cheaper — avoids an unnecessary write if validation fails	Extra write already applied even if later logic fails

Note: Attempting to assign a value to :NEW inside an AFTER trigger raises a compile-time error (ORA-04084) — column values can only be modified in a BEFORE row-level trigger.

---

3. Row-Level and Statement-Level Triggers

Independent of timing, a trigger's level determines how many times it fires relative to the rows affected by the triggering statement.

3.1 Row-Level Triggers

A row-level trigger, declared with FOR EACH ROW, fires once for every row affected by the triggering statement. It has access to the :OLD and :NEW pseudo-records, representing the row's values before and after the change, which makes it suitable for logic that depends on individual column values.

Qualifier	Available In	Meaning
:OLD.column	UPDATE, DELETE	The row's value before the change (NULL for INSERT)
:NEW.column	INSERT, UPDATE	The row's value after the change (NULL for DELETE)

Row-Level Trigger — Example

```
CREATE OR REPLACE TRIGGER trg_emp_salary_check
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
BEGIN
  IF :NEW.salary < :OLD.salary THEN
    RAISE_APPLICATION_ERROR(-20010, 'Salary cannot be decreased.');
  END IF;
END;
/
```

3.2 Statement-Level Triggers

bcalink.com

A statement-level trigger (the default, when FOR EACH ROW is omitted) fires exactly once for the triggering statement, regardless of how many rows it affects — even if zero rows are affected. Because it does not run per row, it has no access to :OLD or :NEW, and is typically used for checks or actions that apply to the statement as a whole.

Statement-Level Trigger — Example

```
CREATE OR REPLACE TRIGGER trg_emp_no_weekend_dml
  BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
  IF TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
    RAISE_APPLICATION_ERROR(-20011,
      'DML on employees is not permitted on weekends.');
  END IF;
END;
/
```

Aspect	Row-Level Trigger	Statement-Level Trigger
Keyword	FOR EACH ROW	(omitted)
Fires	Once per affected row	Once per triggering statement
:OLD / :NEW available?	Yes	No
Typical Use	Per-row validation, auditing individual changes	Statement-wide checks (e.g. time-of-day restrictions, single summary log entry)

bcalink.com

4. INSTEAD OF Triggers

An INSTEAD OF trigger is created on a view, not a table, and its body runs in place of the DML statement rather than before or after it. This is essential for views that Oracle cannot otherwise modify directly — most commonly views built from a join of multiple tables, or views containing aggregate functions, DISTINCT, or GROUP BY — since Oracle does not know which underlying base table a plain INSERT/UPDATE/DELETE against such a view should actually affect.

When an INSTEAD OF trigger is defined, any DML issued against the view triggers the PL/SQL block instead of an attempted (and otherwise likely unsupported) direct modification — the trigger body then decides explicitly which base table(s) to insert into, update, or delete from.

Example Setup — A Non-Updatable Join View

```
CREATE VIEW emp_dept_view AS
SELECT e.emp_id, e.emp_name, e.salary,
    d.department_id, d.department_name
FROM employees e JOIN departments d
 ON e.department_id = d.department_id;
```

INSTEAD OF INSERT Trigger — Example

```
CREATE OR REPLACE TRIGGER trg_emp_dept_insert
 INSTEAD OF INSERT ON emp_dept_view
 FOR EACH ROW
BEGIN
 INSERT INTO employees (emp_id, emp_name, salary, department_id)
 VALUES (:NEW.emp_id, :NEW.emp_name, :NEW.salary, :NEW.department_id);
 -- Optionally also maintain the departments table here,
 -- e.g. inserting a new department if it doesn't already exist.
END;
/
```

Once this trigger exists, an application can issue a normal INSERT against the view:

Resulting DML — Example

```
INSERT INTO emp_dept_view (emp_id, emp_name, salary, department_id, department_name)
VALUES (105, 'Bikash Thapa', 60000, 20, 'Sales');
```

Note: From the calling application's perspective, the view now behaves as though it were directly insertable — the fact that an INSTEAD OF trigger is silently redirecting the operation to the base tables is completely transparent.

INSTEAD OF triggers are always defined FOR EACH ROW — Oracle does not support INSTEAD OF triggers at the statement level.

---

5. Trigger Predicates & Management

5.1 Combining Multiple Events with Trigger Predicates

A single trigger can respond to more than one DML event by combining them with OR in its triggering clause. Inside the trigger body, the boolean predicates INSERTING, UPDATING,

bcalink.com

and DELETING identify which specific event caused this particular firing.

Multi-Event Trigger with Predicates — Example

```
CREATE OR REPLACE TRIGGER trg_emp_audit_all
  AFTER INSERT OR UPDATE OR DELETE ON employees
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO emp_audit VALUES (:NEW.emp_id, 'INSERT', SYSDATE);
  ELSIF UPDATING THEN
    INSERT INTO emp_audit VALUES (:NEW.emp_id, 'UPDATE', SYSDATE);
  ELSIF DELETING THEN
    INSERT INTO emp_audit VALUES (:OLD.emp_id, 'DELETE', SYSDATE);
  END IF;
END;
/
```

5.2 The WHEN Clause

A row-level trigger can include an optional WHEN clause, which restricts firing to rows that satisfy a given condition, avoiding the need for an IF check as the first statement inside the trigger body.

WHEN Clause — Example

```
CREATE OR REPLACE TRIGGER trg_emp_raise_only
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
  WHEN (NEW.salary > OLD.salary)
BEGIN
  DBMS_OUTPUT.PUT_LINE('Salary increased for employee ' || :NEW.emp_id);
END;
/
```

Note: Inside the WHEN clause itself, :OLD and :NEW are written without the colon prefix — simply OLD and NEW — since the WHEN condition is not part of the PL/SQL executable section.

bcalink.com

5.3 Enabling, Disabling, and Dropping Triggers

Statement	Effect
ALTER TRIGGER trigger_name DISABLE;	Prevents the trigger from firing, without removing its definition
ALTER TRIGGER trigger_name ENABLE;	Re-enables a previously disabled trigger
ALTER TABLE table_name DISABLE ALL TRIGGERS;	Disables every trigger defined on the table
DROP TRIGGER trigger_name;	Permanently removes the trigger definition

Note: Triggers are also useful to disable temporarily during bulk data loads (e.g. initial migration), so that per-row trigger logic does not slow down or interfere with a large batch operation.

Key Takeaways

✓ A trigger is defined by three independent choices: timing (BEFORE/AFTER/INSTEAD OF), event (INSERT/UPDATE/DELETE), and level (row or statement).

✓ BEFORE triggers run before the change is applied and may modify :NEW values; AFTER triggers run afterward and can only read :OLD/:NEW.

✓ Row-level triggers (FOR EACH ROW) fire once per affected row and expose :OLD/:NEW; statement-level triggers fire once per statement regardless of row count.

✓ INSTEAD OF triggers apply only to views and replace the DML entirely, making otherwise non-updatable views (joins, aggregates) modifiable.

✓ INSERTING / UPDATING / DELETING predicates let one trigger distinguish which event fired it; the WHEN clause restricts row-level firing by condition.

✓ Triggers can be enabled, disabled, or dropped with ALTER TRIGGER / ALTER TABLE / DROP TRIGGER, useful for maintenance and bulk-load scenarios.

bealink.com

— End of Unit 5 —