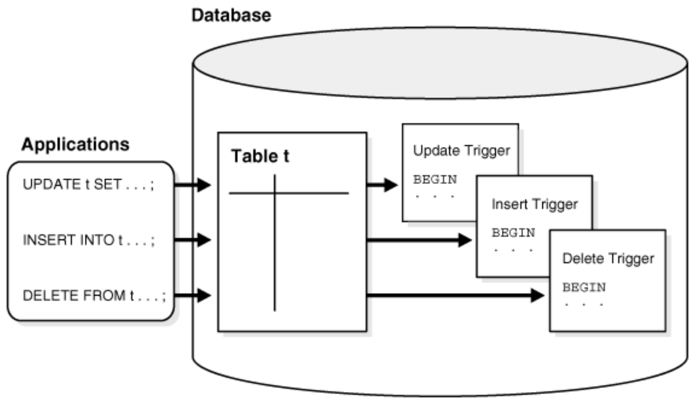
CMSC 508 Database Theory

Advanced SQL (II)

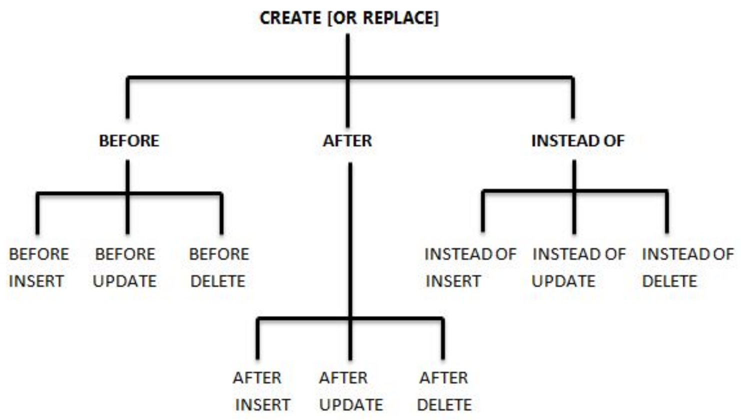
Dr. Alberto Cano
Assistant Professor
Department of Computer Science

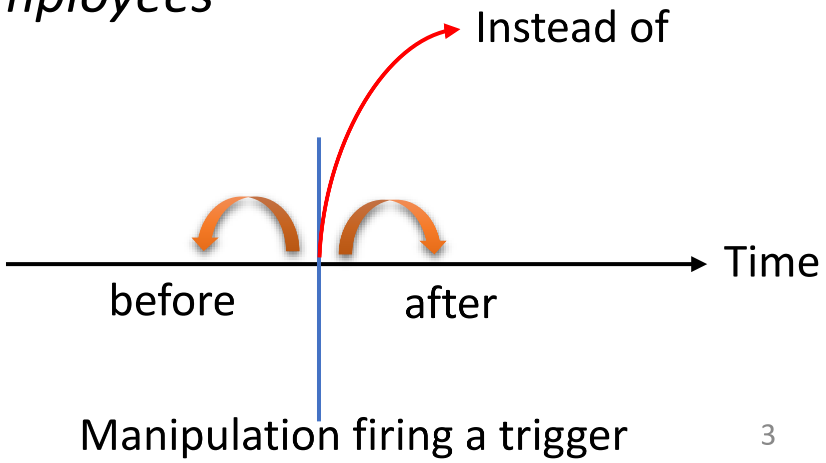
Chapter 4 from Database System Concepts, 6th Ed. by Silberschatz, Korth, Sudarshan, 2011
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003

- Triggers
  - A **trigger** is a statement that is executed **automatically** by the DBMS as a result of a modification to the database
  - To design a trigger mechanism, we must:
    - Specify the **conditions** under which the trigger is to be executed
    - Specify the **actions** to be taken when the trigger executes
  - Triggers vs procedures: a procedure is explicitly run by an user or trigger. Triggers are **implicitly fired** by the DBMS when a triggering condition occurs.



- Triggers
  - Triggering event can be **insert**, **delete** or **update**
  - Triggers may execute **before**, **after** or **instead of** a manipulation
  - Triggers on **update** can be restricted to specific attributes
    - Examples:
      - **before insert on** *departments*
      - **after update of** *salary* **on** *employees*
      - **instead of delete on** *employees*





- Triggers
  - Values of attributes before and after an update can be referenced

Statement	**:old.attribute**	**:new.attribute**
INSERT	NULL	Post-insert value
UPDATE	Pre-update value	Post-update value
DELETE	Pre-delete value	NULL

CREATE OR REPLACE TRIGGER *log_salary*
AFTER UPDATE OF *salary* **ON** *employees*
FOR EACH ROW
BEGIN
INSERT INTO *sal_log* (log_date, employee_id, new_salary, old_salary)
VALUES (SYSDATE, :new.employee_id, :new.salary, :old.salary);
END;

- Trigger (AFTER):
  - Commonly employed for log information after modification
  - DB example: maintaining the job history of the employees

CREATE OR REPLACE TRIGGER *update_job_history*
AFTER UPDATE OF *job_id*, *department_id* **ON** *employees*
FOR EACH ROW
BEGIN
add_job_history(:old.employee_id, :old.hire_date, sysdate,
                :old.job_id, :old.department_id);
END;

where add_job_history is a **procedure** performing:

INSERT INTO *job_history* **VALUES**
(p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id);

- Trigger (BEFORE):
  - Commonly employed for checking conditions prior modification
  - Example: check salary conditions

CREATE OR REPLACE TRIGGER *salary_check*
BEFORE INSERT OR UPDATE OF *job_id, salary* **ON** *employees*
FOR EACH ROW
BEGIN
check_sal(:new.job_id, :new.salary, :new.last_name);
END;

where check_sal is a procedure performing a validation of the salary, e.g: AVG (salary) – 2*STDDEV(salary) < :new.salary < AVG(salary) + 2*STDDEV(salary)) given the salary of the job_id group.

- Trigger (INSTEAD OF):
  - Provide a transparent way of modifying **views** that cannot be modified directly through DML (INSERT, UPDATE, DELETE)

```
CREATE OR REPLACE TRIGGER insert_emp_dept  INSTEAD OF INSERT ON emp_dept_join
DECLARE  v_department_id departments.department_id%TYPE;
BEGIN
  BEGIN
    SELECT department_id INTO v_department_id
    FROM   departments
    WHERE  department_name = :new.department_name;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      INSERT INTO departments (department_id, department_name)
             VALUES (departments_seq.nextval, :new.department_name)
             RETURNING department_id INTO v_department_id;
  END;

  INSERT INTO employees (employee_id, first_name, last_name, department_id)
  VALUES(employees_seq.nextval, :new.first_name, :new.last_name, v_department_id);
END;
```

- Trigger examples:
  - Create a trigger to maintain a new column in the departments table that stores the total salary of all members in a department
  - Prerequisites:

    alter table *departments* **add** *total_salary* **numeric**;
  - Logic: trigger should be executed when:
    - New employee is inserted
    - Employee is removed
    - Employee's salary is updated
    - Employee's department is updated

- Trigger examples:

```
CREATE OR REPLACE TRIGGER total_salary
AFTER DELETE OR INSERT OR UPDATE OF department_id, salary ON employees
    FOR EACH ROW BEGIN
        IF DELETING OR (UPDATING AND :old.department_id != :new.department_id)
            THEN UPDATE departments
            SET total_salary = total_salary - :old.salary
            WHERE department_id = :old.department_id;
        END IF;
        IF INSERTING OR (UPDATING AND :old.department_id != :new.department_id)
            THEN UPDATE departments
            SET total_salary = total_salary + :new.salary
            WHERE department_id = :new.department_id;
        END IF;
        IF (UPDATING AND :old.department_id = :new.department_id AND :old.salary != :new.salary)
            THEN UPDATE departments
            SET total_salary = total_salary - :old.salary + :new.salary
            WHERE department_id = :new.department_id;
        END IF;
    END;
```

- Trigger examples:
  - Create a trigger to **maintain** a derived column that stores the total salary of all members in a department
  - Issues:
    - How to compute the current total salary?

    1) update *employees* **set** *salary* = *salary*; ?

    2) update *departments* **set** *total_salary* = 0; then 1) ?

    total_salary is null … total_salary + :new.salary will be null

    salary = *salary* … will execute the trigger, but any condition is satisfied

- Trigger examples:
  - Create a trigger to **maintain** a derived column that stores the total salary of all members in a department
  - Issues:
    - How to compute the current total salary?

update *departments* d
set *d.total_salary* =
    (**select** sum(*e.salary*) **from** employees e
    where *d.department_id* = *e.department_id*);

    - What if inserting/updating wrong department ID?

Referential constraints will halt the query violating integrity

- Trigger exercise:
  - Create a trigger to increase the salary (+5% of current salary) of the employees belonging to a department every time an employee joins that department.

  - Identify conditions to execute the trigger
  - Identify actions using new and old references
  - Merge conditions with common actions

Nope, not this kind of trigger

- Trigger exercise:
  - Create a trigger to increase the salary (+5% of current salary) of the employees belonging to a department every time an employee joins that department.

```
CREATE OR REPLACE TRIGGER update_salary
AFTER INSERT OR UPDATE OF department_id ON employees
FOR EACH ROW
BEGIN
IF INSERTING OR (UPDATING AND :old.department_id != :new.department_id)
THEN UPDATE employees
SET salary = salary*1.05
WHERE department_id = :new.department_id;
END IF;
END;
```

Trigger compiles and everything looks good. Let's run something to execute it

- Trigger exercise:
  - Create a trigger to increase the salary (+5% of current salary) of the employees belonging to a department every time an employee joins that department.

```
CREATE OR REPLACE TRIGGER update_salary
AFTER INSERT OR UPDATE OF department_id ON employees
FOR EACH ROW
BEGIN
IF INSERTING OR (UPDATING AND :old.department_id != :new.department_id)
THEN UPDATE employees
SET salary = salary*1.05
WHERE department_id = :new.department_id;
END IF;
END;
```

ERROR: **EMPLOYEES is mutating, trigger/function may not see it**
Within a stored function or trigger, it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

- Trigger exercise:

- A **mutating table** is a table that is currently being modified by an update, delete, or insert statement. When a trigger tries to reference a table that is in state of flux (being changed), it is considered "mutating", and raises an error since Oracle should never return inconsistent data

```
CREATE OR REPLACE TRIGGER update_salary
BEFORE INSERT ON employees
FOR EACH ROW BEGIN
UPDATE employees
SET salary = salary*1.05
WHERE department_id = :new.department_id;
END;
```

This does not produce any error

How about updating the department_id for a current employ?
Cannot update salary after updating department_id -> mutating table

CMSC 508
Database Theory

Advanced SQL (II)

Dr. Alberto Cano
Assistant Professor
Department of Computer Science

Chapter 4 from Database System Concepts, 6th Ed. by Silberschatz, Korth, Sudarshan, 2011
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003