Lesson 7.3: Triggers & Views

CSC430/530 – DATABASE MANAGEMENT SYSTEMS

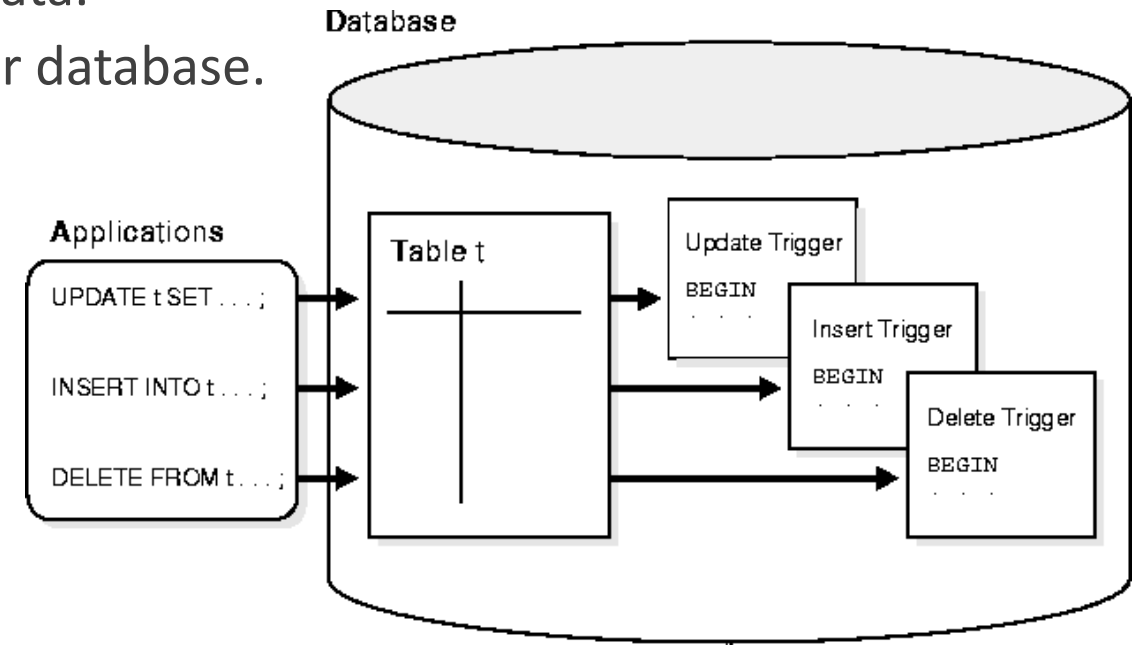
DR. ANDREY TIMOFEYEV

OUTLINE

- Triggers.
  - Introduction.
  - Syntax.
- Views.
  - Introduction.
  - Syntax.
  - Implementation & updates.

TRIGGERS: INTRO

- **Triggers** are used to specify **automatic actions** that the database system will perform when certain **events** and **conditions** occur.
- **Why using triggers**?
  - **Actions** are performed **automatically** within the database.
    - No need to write any **extra** application **code**.
  - **Complex** application **logic** is executed "*close*" to the data.
  - Triggers increase the **complexity** and **flexibility** of your database.
- Possible triggers **use cases**:
  - Enforce complex **data integrity rules**.
  - Prevent **invalid DML** transactions from occurring.
  - Enhance complex **database security** rules.



Trigger actions on database

TRIGGERS: SYNTAX (1)

- **General syntax** of a trigger.
  - ON *<event>*
    IF *<condition>*
    THEN *<action>*
    - **<event>** - request to execute **database operation**.
    - **<condition>** - predicate evaluated on **database state**.
    - **<action>** - execution of procedure that might involve **database updates**.
- Event(s) – **INSERT**, **DELETE**, **UPDATE** statements.
  - **BEFORE** modifier – trigger is executed **before** the operation specified in the event is **executed**.
  - **AFTER** modifier – trigger is executed **after** the operation specified in the event is **completed**.
- Condition(s) – anything that is allowed in a **WHERE** clause.
  - Determines whether the **trigger action** is **executed**.
- Action(s) – individual (or sequence of) **SQL statement(s)** or stored **procedures**.

TRIGGERS: SYNTAX (2)

- **General syntax** of a trigger.
  - ON *<event>*
    IF *<condition>*
    THEN *<action>*
    - **<event>** - request to execute **database operation**.
    - **<condition>** - predicate evaluated on **database state**.
    - **<action>** - execution of procedure that might involve **database updates**.
- **Example:**
  ON <updating employee salary>
  IF <salary of employee is higher than salary of supervisor>
  THEN <restrict update>

- Triggers have access to the **data before** (*NEW*) or **after** (*OLD*) the **execution** of the triggering event.
  - **SET** is used to **modify fields** of table that activated the trigger.

INSERT	DELETE	UPDATE
NEW		NEW
	OLD	OLD

TRIGGERS: HANDS ON (1)

- Write a trigger to update the salary of an employee with an average salary of the department where he/she works BEFORE INSERT(ing) the record in the employee table IF the salary is empty or NULL.

TRIGGERS: HANDS ON (1)

- Write a trigger to update the salary of an employee with an average salary of the department where he/she works BEFORE INSERT(ing) the record in the employee table IF the salary is empty or NULL.

```
DELIMITER $$
CREATE TRIGGER emp_addr_trig
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
        IF (NEW.salary = '' OR NEW.salary IS NULL)
        THEN
        SET NEW.salary =  (SELECT AVG(salary)
                                FROM employee e
                                WHERE e.dno = NEW.dno);
        END IF;
END$$
DELIMITER ;
```

TRIGGERS: HANDS ON (1)

- Write a trigger to update the salary of an employee with an average salary of the department where he/she works BEFORE INSERT(ing) the record in the employee table IF the salary is empty or NULL.

```
DELIMITER $$
CREATE TRIGGER emp_addr_trig
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
        IF (NEW.salary = '' OR NEW.salary IS NULL)
        THEN
        SET NEW.salary =  (SELECT AVG(salary)
                                   FROM employee e
                                   WHERE e.dno = NEW.dno);
        END IF;
END$$
DELIMITER ;
```

Event

Condition

Action

TRIGGERS: HANDS ON (2)

- Write a trigger to enforce following constraint:
  - Dependent relationship must be either spouse, son, or daughter. If anything else, then display message – “Please, provide valid relationship (Spouse, Son or Daughter).”

TRIGGERS: HANDS ON (2)

- Write a trigger to enforce following constraint:
  - Dependent relationship must be either spouse, son, or daughter, if anything else, then display message – “Please, provide valid relationship (Spouse, Son or Daughter).”

```
DELIMITER $$
CREATE TRIGGER dependent_relationship
BEFORE INSERT ON dependent
FOR EACH ROW
BEGIN
        DECLARE msg VARCHAR(255);
        IF NEW.relationship NOT IN ('Spouse', 'Daughter', 'Son')
        THEN /* Cause Error Message */
        SET msg = 'Please, provide correct relationship (Spouse, Son or Daughter).';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
        END IF;
END$$
DELIMITER ;
```

TRIGGERS: HANDS ON (3)

- Write a trigger to create a default project for each new department inserted into database.

TRIGGERS: HANDS ON (3)

- Write a trigger to create a default project for each new department inserted into database.

```
DELIMITER $$
CREATE TRIGGER new_dept
AFTER INSERT ON department
FOR EACH ROW
BEGIN
        INSERT INTO project
        VALUES(CONCAT(NEW.dname, ' - Initialization'), CONCAT(NEW.dnumber, '1'), 'Houston',
        NEW.dnumber);
END$$
DELIMITER ;
```

VIEWS: INTRO

- **View** is a **single table** that is derived from **other tables** (*base tables* or *other views*).
  - Does not have to **exist** in **physical form** – considered to be a **virtual table**.
    - **Updates** are **limited**, but no **limitations** on **queries**.
- **View** is a way to specify (*joined*) **table** that is **referenced frequently** but does not need to be **defined physically**.
- Company database **example**:
  - Frequently referencing to **employee name** & **project name** he/she works on.
    - Rather than specifying a join of **EMPLOYEE**, **WORKS_ON** and **PROJECT** relation every time we issue the query, it is easier to define a **view** that specifies the **result** of this join.

VIEWS: SYNTAX

- CREATE VIEW is used to **define** new **view**.
  - Must provide a (*virtual*) **table name** (*view name*), a **list** of **attribute** names, and a **query** to specify the **contents**.
    - If some of the view attributes are **derived** by functions or arithmetic operations, then **renaming** those attributes is recommended.
- **Example**:
  - Create a view that provides information about the **employees** and the **projects** that they **work on**.

```
CREATE VIEW works_on_ext
AS SELECT e.fname, e.lname, p.pname, w.hours
FROM employee e, project p, works_on w
WHERE e.ssn = w.essn AND w.pno = p.pnumber;
```

VIEWS: IMPLEMENTATION & UPDATES

- Two main approaches for **views implementation**.
  - **Query modification**.
    - View query is transformed into a query that joins underlying base tables.
  - **View materialization**.
    - Temporary physical view table is created when the view is first created or queried.
- View always contains **updated data**.
  - **Immediate update**.
    - View is updated as soon as base tables are changed.
  - **Lazy update**.
    - View is updated when needed by a view query.

VIEWS: HANDS ON

- Create a view that displays the number of employees and the total salary paid in each department.

VIEWS: HANDS ON

- Create a view that displays the number of employees and the total salary paid in each department.
  - CREATE VIEW dept_info(dept_name, no_of_emps, total_sal)
    AS SELECT dname, COUNT(*), SUM(salary)
    FROM department d, employee e
    WHERE d.dnumber = e.dno
    GROUP BY dname;

SUMMARY

- Purpose of triggers.
- Triggers syntax.
- Purpose of views.
- Views syntax.
- Views implementation & updates.