Constraints and Triggers

Intro >> Lecture's Map

Learning Maps

Sequence	Title
1	Introduction to databases
2	Relational Databases
3	Relational Algebra
4	Structured Query Language – Part 1
5	Structured Query Language – Part 2
6	Constraints and Triggers
7	Entity Relationship Model
8	Functional Dependency
9	Normalization
10	Storage - Indexing
11	Query Processing
12	Transaction Management – Part 1
13	Transaction Management – Part 2

Intro > Overview

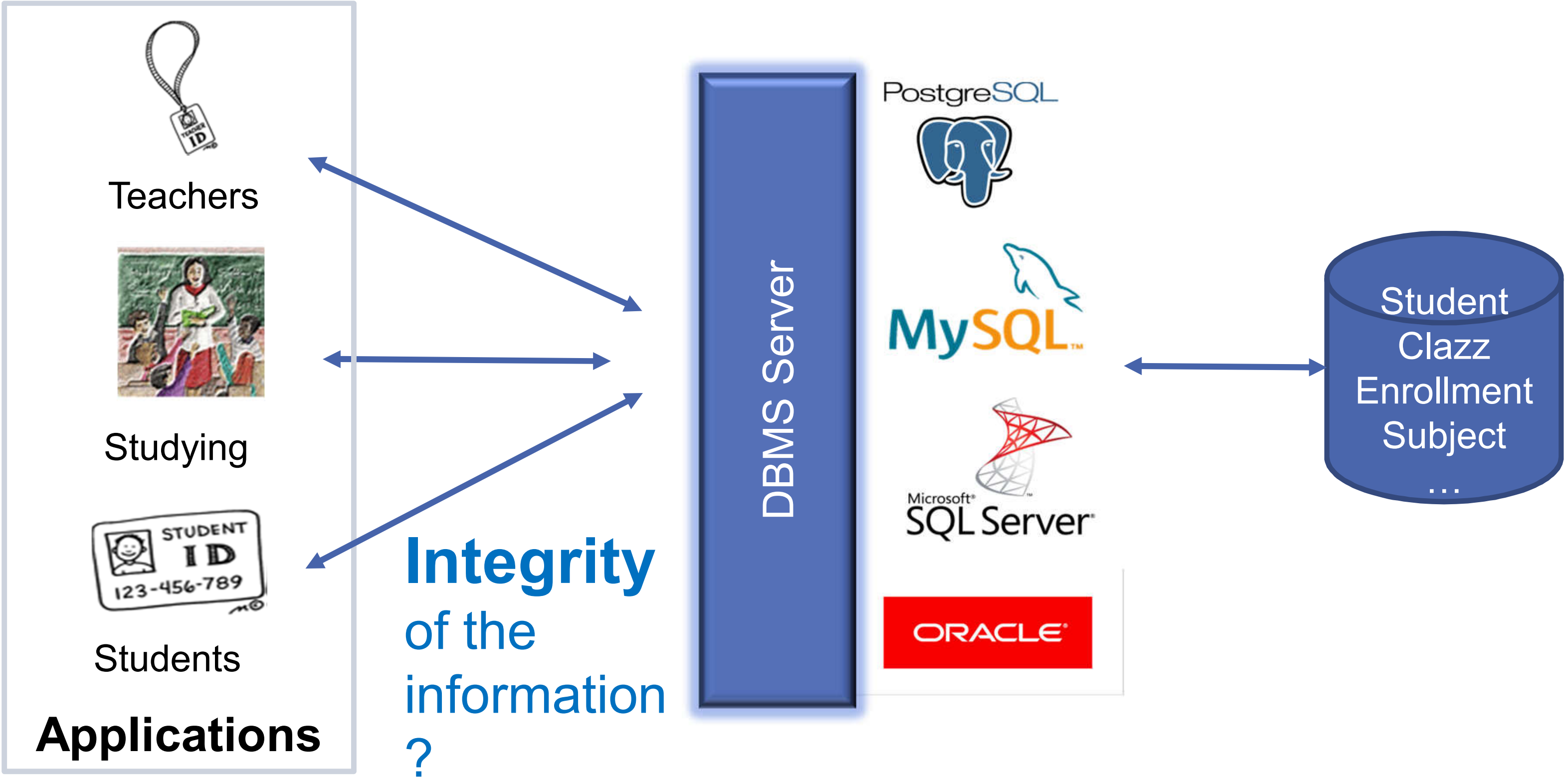
□ A : Voice and PPT Overview
□ B : Text-based Overview
■ C : Video and PPT Overview

Opening Message	→ In this lesson, we will study.
Lesson topic	1. …. 2. …. 3. …
Learning Goals	Upon completion of this lesson, students will be able to: 1. …… 2. ……

Intro > Keywords

Keyword	Description
Constraints	Constraints are the rules enforced on the data columns of a table. Constraints could be either on a column level or a table level
Triggers	A trigger is a SQL procedure that initiates an action (i.e., fires an action) when an event (INSERT, DELETE or UPDATE) occurs. They are stored in and managed by the DBMS
PL/SQL	Procedural Language/Structured Query Language is Oracle Corporation's procedural extension for SQL and the Oracle relational database

Lesson > Topic 1: Introduction



Integrity of the information ?

Database Schema

student(**student_id**, first_name,last_name, dob, gender,
address, note, email, *clazz_id*)

clazz(**clazz_id**, name, *lecturer_id*, *monitor_id,* number_students)

subject(**subject_id**, name, credit, percentage_final_exam)

enrollment(***student_id*****,** ***subject_id*****, semester**, midterm_score, final_score)

lecturer(**lecturer_id**, first_name, last_name, dob, gender, address, email)

teaching(***subject_id, lecturer_id***)

grade(**code**, from_score, to_score)

1. ....

▶ 1.1 Constraints and Triggers

....

- A *constraint* is a relationship among data elements that the DBMS is required to enforce
  - Example: key constraints

- *Triggers* are only executed when a specified condition occurs, e.g., insertion of a tuple
  - Easier to implement than complex constraints

Lesson > Topic 2: Constraints

Slide 8

O1 Oanh, 27/08/2018

2. ....

▶ Kinds of Constraints

- Keys
- Foreign-key, or referential-integrity
- Value-based constraints
  - Constrain values of a particular attribute.
- Tuple-based constraints
  - Relationship among components
- Assertions: any SQL boolean expression

2. ....

► 2.1 Keys : PRIMARY KEY vs. UNIQUE

- Declaring: similar syntax as primary key
- Example:

```
CREATE TABLE student (
      student_id CHAR(8) NOT NULL,
      first_name VARCHAR(20) NOT NULL,
      last_name VARCHAR(20) NOT NULL,
      ...
      email varchar(50) UNIQUE,
      clazz_id CHAR(8),
      CONSTRAINT student_pk PRIMARY KEY (student_id));
```

2. ....

2.1 Keys : PRIMARY KEY vs. UNIQUE

	PRIMARY KEY	UNIQUE KEY
Number defined on table	One	Multiple
Null columns allowed	No	Yes
Default index	CLUSTERED	NON-CLUSTERED
Purpose	Enforce Entity Integrity	Enforce Unique Data
Number of columns	One or more columns	One or more columns
Referenced by a Foreign Key Constraint	Yes	Yes

2. ....

▶ 2.2 Foreign keys

Expressing Foreign Keys

- Use keyword REFERENCES, either:
  1. After an attribute (for one-attribute keys)
  2. As an element of the schema:

     [CONSTRAINT <name>] FOREIGN KEY (<list of attributes>)

     REFERENCES <relation> (<attributes>)

- Referenced attributes must be declared PRIMARY KEY or UNIQUE

2. ....

▶ 2.2 Foreign keys

Example

```
CREATE TABLE clazz (
      clazz_id CHAR(8) NOT NULL PRIMARY KEY,
      name VARCHAR(20), … );

CREATE TABLE student (
      student_id CHAR(8) NOT NULL,
      ... ,
      clazz_id CHAR(8),
      CONSTRAINT student_pk PRIMARY KEY (student_id));

ALTER TABLE student ADD CONSTRAINT student_fk_class
FOREIGN KEY (clazz_id) REFERENCES clazz(clazz_id);
```

2. ....

► 2.2 Foreign keys

Enforcing constraint

- An insert or update to **student** that introduces a non-existent clazz_id (clazz_id value is not found in **clazz**)

  ➔ **Reject**

- A deletion or update to **clazz** that removes a clazz_id value found in some tuples of **student** ?
  - Default: reject the modification
  - Cascade: make the same changes in **student**
  - Set NULL: change clazz_id in **student** to NULL

2. ....

▶ 2.2 Foreign keys

Choosing policy

```
ALTER TABLE student
        ADD CONSTRAINT student_fk_class FOREIGN KEY
(clazz_id) REFERENCES clazz(clazz_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE;
```

2. ....

► 2.3 Attribute-based checks

Declaring

- Constraints on the value of a particular attribute
  - Add CHECK(<condition>) to the declaration for the attribute or add as **relation-schema element**
  - The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery
- Example:

```
CREATE TABLE student (
    student_id CHAR(8) NOT NULL PRIMARY KEY, ...,
    gender CHAR(1),
    clazz_id CHAR(8) CHECK (clazz_id IN (SELECT clazz_id FROM
clazz)),
    CONSTRAINT student_chk_gender CHECK (gender = 'F' OR
gender = 'M') );
```

2. ....

▶ 2.3 Attribute-based checks

Timing of checks

- Only when a value for that attribute is inserted or updated

```
CREATE TABLE student (
     student_id CHAR(8) NOT NULL PRIMARY KEY, ...,
     gender CHAR(1),
     clazz_id CHAR(8) CHECK (clazz_id IN (SELECT
clazz_id FROM clazz)),
     CONSTRAINT student_chk_gender CHECK (gender = 'F'
OR gender = 'M') );
```

Not checked if a class is deleted from **clazz**

2. ....

▶ 2.4 Tuple-based checks

- CHECK (<condition>) may be added as a **relation-schema element**.
- The condition may refer to any attribute of the relation
  - But other attributes or relations require a subquery
- Timing of checks: on insert or update only.

```
CREATE TABLE grade(
          code CHAR(1) NOT NULL,
          from_score DECIMAL(3,1) NOT NULL,
          to_score DECIMAL(3,1) NOT NULL, ...,
          CONSTRAINT grade_chk_toScore CHECK (to_score >
from_score) );
```

2...

▶ 2.5 Assertions

Declaring

- Database-schema elements, like relations or views
- Defined by:

```
CREATE ASSERTION <name>
        CHECK (<condition>);
```

- Condition may refer to any relation or attribute in the database schema
- Drop an assertion:

```
DROP ASSERTION <assertion name>;
```

2...

▶ 2.5 Assertions

Example

```
CREATE ASSERTION teachingSubject CHECK (
       (SELECT COUNT(*) FROM teaching) >=
       (SELECT COUNT(*) FROM subject) );

CREATE ASSERTION numberStdInClass CHECK (
       NOT EXISTS (
              SELECT * FROM clazz c
              WHERE number_students <>
                     (SELECT count(*) FROM student
                      WHERE clazz_id = c.clazz_id)
       )
);
```

Timing of Assertion Checks

- In principle, we must check every assertion after every modification to any relation of the database

- A clever system can observe that only certain changes could cause a given assertion to be violated
  - No change to **student** can affect **teachingSubject**
  - Neither can an insertion to **teaching**

- Very hard to implement assertions efficiently

3. Triggers

▶ 3.1 What is a database trigger?

Motivation

- **Assertions**
  - powerful,
  - but the DBMS often can’t tell when they need to be checked
- **Attribute- and tuple-based checks**
  - checked at known times,
  - but are not powerful
- **Triggers** let the **user decide** when to check for any condition

► 3.1 What is a database trigger?

ECA Rules

- A trigger defines an operation that is performed when a specific event occurs on a relation:
  - inserts a new record / updates an existing record / deletes a record
- Trigger functions have access to special variables from the data base engine
- Called also ECA rules (**Event**-**Condition**-**Action**)
  - Event: type of database modification
  - Condition: Any SQL Boolean-valued expression
  - Action: Any SQL statements

3. Triggers

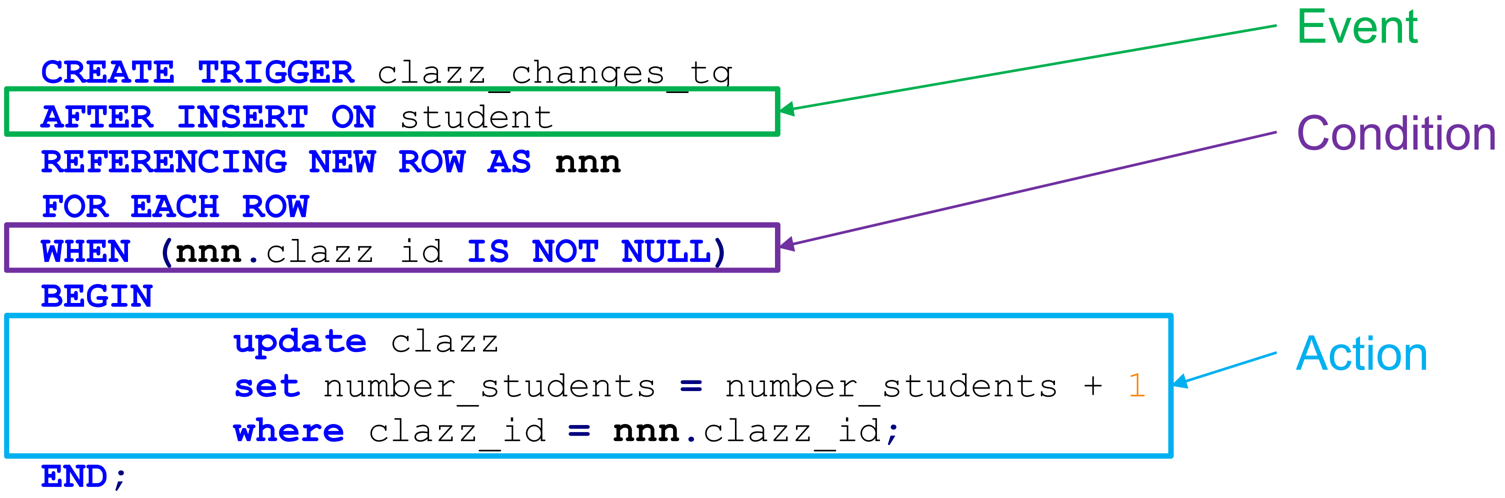
► 3.1 What is a database trigger?

Example

Constraint: when a new student is inserted into student relation, the number of students in his class must be increased

student(**<u>student_id</u>**, first_name,last_name, dob, gender, address, note, email, *clazz_id*)

clazz(**<u>clazz_id</u>**, name, *lecturer_id*, *monitor_id,* number_students)



```
CREATE TRIGGER clazz_changes_tg
AFTER INSERT ON student
REFERENCING NEW ROW AS nnn
FOR EACH ROW
WHEN (nnn.clazz_id IS NOT NULL)
BEGIN
        update clazz
        set number_students = number_students + 1
        where clazz_id = nnn.clazz_id;
END;
```

3. Triggers

▶ 3.2 Trigger Definition

Syntax

- Creating a trigger:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
       {BEFORE | AFTER | INSTEAD OF }
       {INSERT | DELETE | UPDATE [OF <attribute_name>]}
       ON <table_name>
       REFERENCING {NEW | OLD} {ROW | TABLE} AS <name>
       [FOR EACH ROW ]
       [WHEN (<condition>) ]
       BEGIN
               <trigger body goes here >
       END;
```

- Dropping a trigger:

```
DROP TRIGGER <trigger_name>;
```

3. Triggers

► 3.2 Trigger Definition

Event

- **AFTER, BEFORE, INSTEAD OF**:
  - AFTER, BEFORE: used for tables / views
  - INSTEAD OF: used only for views
    - A way to execute view modifications: triggers translate them to appropriate modifications on the base tables
- **INSERT, DELETE, UPDATE , UPDATE OF**
  - UPDATE OF <colums> : update on a particular column

3. Triggers

▶ 3.2 Trigger Definition

Triggers level

- Row-level trigger:
  - Indicated by option FOR EACH ROW
  - Trigger executes once for each modified tuple
- Statement-level trigger:
  - Without option FOR EACH ROW or with FOR EACH STATEMENT
  - Trigger execute once for a SQL statement, regardless how many tuples are modified

3. Triggers

► 3.2 Trigger Definition

REFERENCING

- **INSERT** statements imply a new tuple (for row-level) or new table (for statement-level)
  - The table is the set of inserted tuples
- **DELETE** implies an old tuple or table
- **UPDATE** implies both
- Refer to these by

  REFERENCING [NEW | OLD] [TUPLE | TABLE] AS <name>
- Each DBMS has its own implementation, REFERENCING may not used:
  - Access directly to special variables from the database engine: NEW, OLD,…

3. Triggers

► 3.2 Trigger Definition

Condition

- Any boolean-valued condition
- Evaluated on the database as it would exist before or after the triggering event, depending on whether BEFORE or AFTER is used.
  - But always before the changes take effect.
- Access the new/old tuple/table through the names in the REFERENCING clause

3. Triggers

► 3.2 Trigger Definition

Action

- Can be more than one SQL statement:
  - Surround by BEGIN .. END
- Language:
  - Simple SQL statements
  - Extention of SQL: procedural languages, depends on each DBMD
    - PL/SQL (Oracle), PL/pgSQL (PostgreSQL), T-SQL(SQL Server) ,..

3. Triggers

► 3.3 Using triggers

When?

- Auditing data modification (keeping history of data), providing transparent event logging
- Validation and business security checking if so is desired
  - Eg. column formatting before and after inserts into database
- Enforcing complex integrity constraints
- Enforcing complex business rules
- Maintaining replicate tables
- Building complex views that are updatable
- …

3. Triggers

3.3 Using triggers

Guidelines for designing triggers

- Do not define triggers that duplicate database features
  - do not define triggers to reject bad data if you can do the same checking through constraints
- Use triggers only for centralized, global operations that must fire for the triggering statement, regardless of which user or database application issues the statement
- Do not create recursive triggers
- Use triggers on DATABASE judiciously (servererror, logon, logoff,…):
  - they are executed for *every user every time* the event occurs on which the trigger is created

3. Triggers

▶ 3.4 Examples

Oracle

- Add a new column in **clazz** relation

```
alter table clazz
add column number_students integer not null default 0;
```

- Create a trigger on **student** relation

```
CREATE TRIGGER clazz_changes_tg
AFTER UPDATE ON student
FOR EACH ROW
WHEN (:NEW.clazz_id <> :OLD.clazz_id)
BEGIN
        update clazz set number_students= number_students+1
        where clazz_id = :NEW.clazz_id;
        update clazz set number_students = number_students-1
        where clazz_id = :OLD.clazz_id;
END;
```

3. Triggers

▶ 3.4 Examples

PostgreSQL

```
CREATE FUNCTION public.tg_fnc_change_clazz()
    RETURNS trigger LANGUAGE 'plpgsql' AS $$
BEGIN
   update clazz set number_students = number_students+1
       where clazz_id = new.clazz_id;
   update clazz set number_students = number_students-1
       where clazz_id = old.clazz_id;
   return new;
END; $$

CREATE TRIGGER tg_af_update_clazz
    AFTER UPDATE OF clazz_id
    ON public.student
    FOR EACH ROW
    EXECUTE PROCEDURE public.tg_fnc_change_clazz();
```

Remarks

- Constraints, Assertions, Triggers:
  - How to declare
  - Timing of checks
  - Differences
- Only use them if you really need to, especially triggers
- Each DBMS has its own variation in implementation:
  - Options
  - Syntax: triggers as an example

  ➔ Reading documentation for each DBMS used

Outro > Summary

No	Topic	Summary
1	Introduction	- Why we need constraints and triggers?
2	Constraints	- Keys - Foreigne key - Check constraints - Assertion
3	Triggers	- Motivation - Triggers definition - Using triggers

You’ve just have an overview of …...

Next lesson:

Entity Relationship Model

1. ……..
2. …..
3. ….