Introduction to Database Systems

2023-Fall

5. The Security and Integrity Constraints (2/2)

Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Three parts:
  - **E**vent (activates the trigger)
  - **C**ondition (tests whether the triggers should run)
  - **A**ction (what happens if the trigger runs)
- Active database rules (ECA rules)

Triggers

- Synchronization of the Trigger with the activating statement (DB modification)
  - Before
  - After
  - Instead of
  - Deferred (at end of transaction).
- ***Update events*** may specify a particular column or set of columns.
- A ***condition*** is specified with a WHEN clause.
- Number of ***Activations*** of the Trigger
  - Once per modified tuple
    (FOR EACH ROW)
  - Once per activating statement
    (default).

Triggers

- Options for the **REFERENCING** clause:
  - NEW TABLE: the set (!) of tuples newly inserted (INSERT).
  - OLD TABLE: the set (!) of deleted or old versions of tuples (DELETE / UPDATE).
  - OLD ROW: the old version of the tuple (FOR EACH ROW UPDATE).
  - NEW ROW: the new version of the tuple (FOR EACH ROW UPDATE).
- The action of a trigger can consist of multiple SQL statements, surrounded by **BEGIN . . . END**.

Example: Row Level Trigger

```
CREATE TRIGGER    NoLowerPrices
AFTER UPDATE OF  price  ON Product
REFERENCING
        OLD  AS OldTuple
        NEW  AS  NewTuple
WHEN (OldTuple.price > NewTuple.price)
        UPDATE  Product
        SET  price = OldTuple.price
        WHERE  name = NewTuple.name
FOR EACH ROW
```

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use **for each statement** instead of **for each row**
  - Use **referencing old table** or **referencing new table** to refer to temporary tables (called ***transition tables***) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

Triggers: Example

```
CREATE TRIGGER youngSailorUpdate
AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
INSERT
        INTO YoungSailors(sid, name, age, rating)
        SELECT sid, name, age, rating
        FROM NewSailors N
        WHERE N.age <= 18
```

- This trigger inserts young sailors into a separate table.
- It has no (i.e., an empty, always true) condition.

Triggers: Example

```
CREATE TRIGGER notTooManyReservations
   AFTER INSERT ON Reserves                          /* Event */
   REFERENCING NEW ROW NewReservation
   FOR EACH ROW
   WHEN (10 <= (SELECT COUNT(*) FROM Reserves
            WHERE sid =NewReservation.sid))      /* Condition */
     DELETE FROM Reserves R
     WHERE R.sid= NewReservation.sid                 /* Action */
           AND day=
        (SELECT MIN(day) FROM Reserves R2 WHERE R2.sid=R.sid);
```

- This trigger makes sure that a sailor has less than 10 reservations, deleting the oldest reservation of a given sailor, if necessary.
- It has a non-empty condition (WHEN).

Triggers vs. General Constraints

- Triggers can be harder to understand.
  - Several triggers can be activated by one SQL statement (*arbitrary order*!).
  - A trigger may activate other triggers (*chain activation*).
- Triggers are procedural.
  - Assertions react on any database modification, trigger only specified event.
  - *Trigger execution cannot be optimized by DBMS*.
- Triggers have more applications than constraints.
  - monitor integrity constraints,
  - construct a log,
  - gather database statistics, etc.

Assuming the following three relations exist:

Sailors(sid, sname, rating, age, master)

/* where sid is the sailor's ID, sname is the name, rating is the level, age is the birth date, and master is the ID of the sailor's master, who is also a sailor /

Boats(bid, bname, color)

/ where bid is the boat's ID, bname is the name, and color is the color of the boat /

Reserves(sid, bid, day)

/ where sid is the ID of the sailor who reserved a boat, bid is the ID of the reserved boat, and day is the date of reservation */

CASE

Implementation of Referencing Integrity Rules

Taking Sailors, Boats, and Reserves as examples, write the rules for implementing referential integrity constraints.

What actions can affect the referential integrity between the three tables?

- **Reserves Insert**
- **Sailors Delete**
- **Boats Delete**
- **Reserves Update**
- **Sailors Update**
- **Boats Update**

Does the Update operation on all attributes affect referential integrity?

Rule 1

Create a trigger to monitor the **Insert operation** of the Reserves table. ***If*** the foreign key attribute of the inserted tuple does not exist in the Sailors and Boats tables, ***roll back*** the operation of inserting the record.

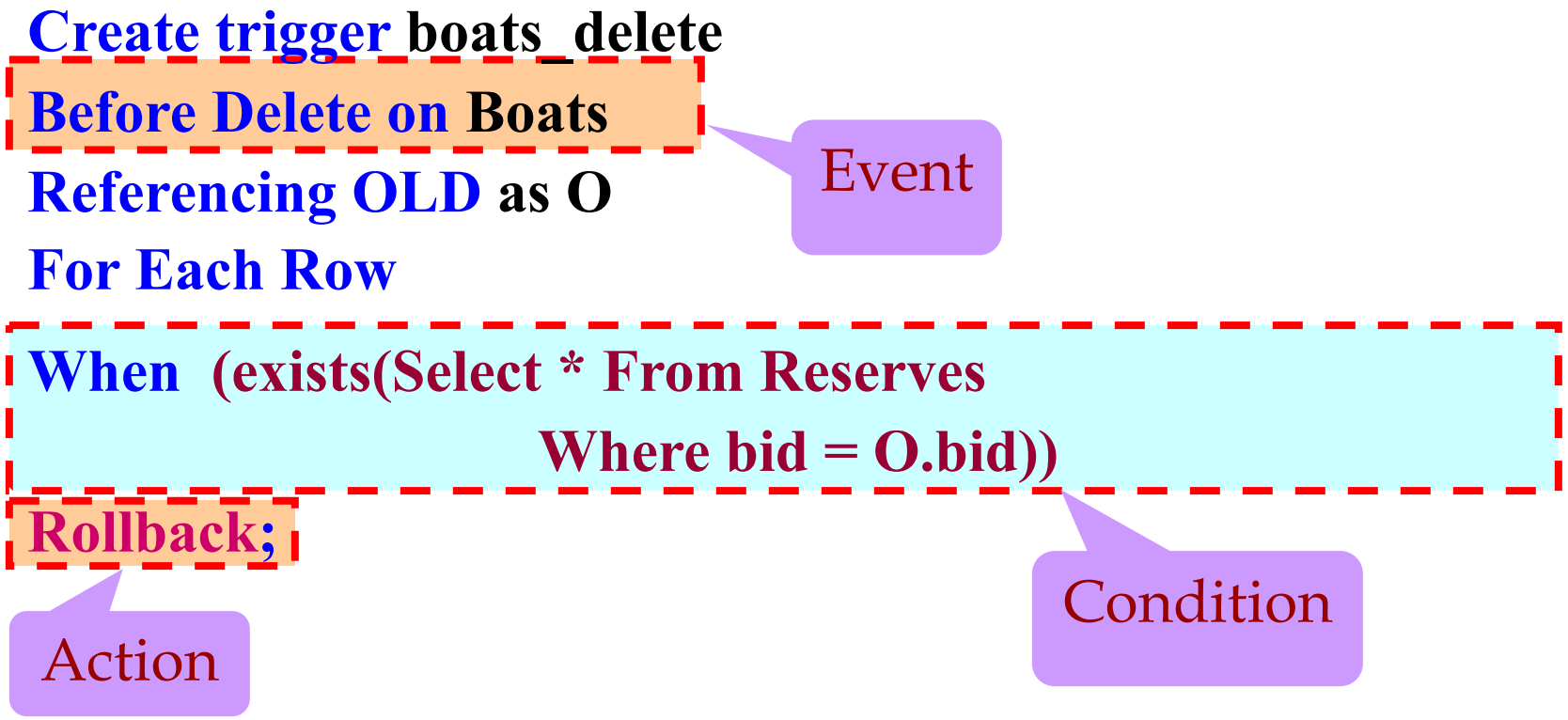
```
Create trigger referential_integrity_check
Before Insert on Reserves
Referencing NEW as N
For Each Row
When (not (exists(Select * From Sailors Where sid = N.sid)
        and
        (exists(Select * From Boats Where bid = N.bid))
        )
Rollback;
```

Condition

Action

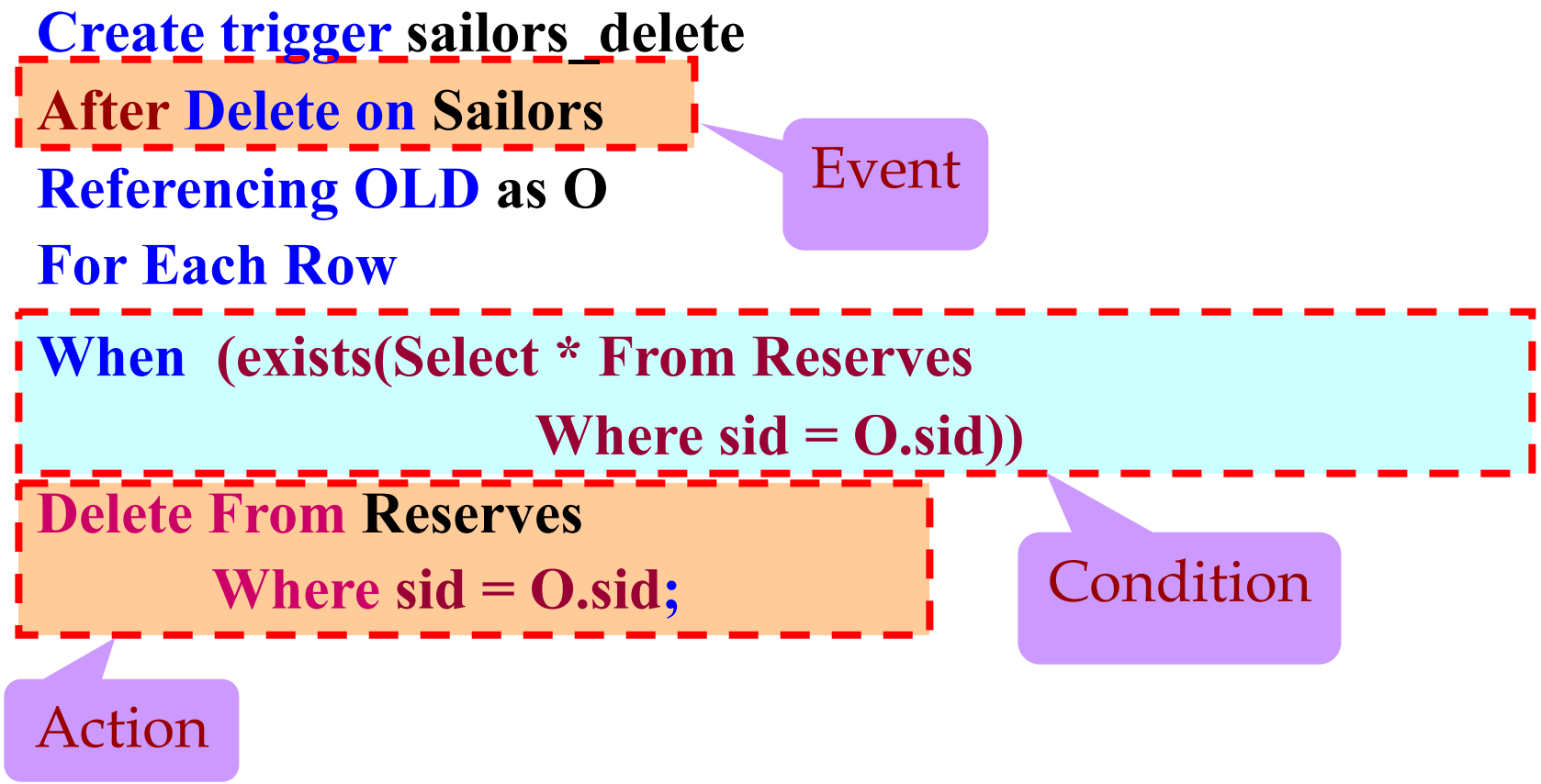
Rule 2

Create a trigger to monitor the **Delete operation** on the Boats table. ***If*** the primary key of the deleted tuple is a foreign key in the Reserves table, ***roll back*** the operation to delete the record.



Rule 3

Create a trigger to monitor the **Delete operation** on the Sailors table. ***If*** the primary key of the deleted tuple is a foreign key in the Reserves table, the relevant records in the Reserves table will be ***deleted***.



Rule 4

Create a trigger to monitor the **Update operation** on the Reserves table. ***If*** the modified tuple sid and bid attribute values do not exist in the Sailors and Boats tables, ***roll back*** the operation to modify the record.

```
Create trigger referential_integrity_check
Before Update of sid,bid on Reserves
Referencing NEW as N
For Each Row
When (not (exists(Select * From Sailors Where sid = N.sid)
          and
          (exists(Select * From Boats Where bid = N.bid))
          )
Rollback;
```

Event

Condition

Action

Rule 5

Create a trigger to monitor the **Update operation** on the Sailors table. ***If*** there is a tuple in the Reserves table that references the pre modified sid value as a foreign key, ***roll back*** this modification operation.

```
Create trigger sailors_sid_update
Before Update of sid on Sailors
Referencing Old as O
For Each Row
When (exists(Select * From Reserves
        Where sid = O.sid))
Rollback;
```

When Not To Use Triggers

- Triggers were used earlier for tasks such as
  - Maintaining summary data (e.g., total salary of each department)
  - Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
  - Databases today provide built in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger

When Not To Use Triggers (Cont.)

- Risk of unintended execution of triggers, for example, when
  - Loading data from a backup copy
  - Replicating updates at a remote site
  - Trigger execution can be disabled before such actions.
- Other risks with triggers:
  - Error leading to failure of critical transactions that set off the trigger
  - Cascading execution

Execution of Rules

- ***Immediate execution***
- ***Deferred execution***
- ***Decoupled or detached mode***
- ***Cascading trigger***
  - Control nested execution of rules
  - Prevent nontermination
    - Triggering graph
    - Specify the upper limit of cascading times
  - So triggers should be used reasonably

Implementation of ECA

- Loosely coupling
- Tightly coupling (DB2, Oracle, etc.)
- Nested method

  The rules are nested into transaction and executed by DBMS as a part of the transaction.
  - Grafting method
  - Query modification method

Summary

- SQL allows specification of rich integrity constraints (ICs): attribute-based, tuple-based CHECK and assertions (table-independent).
- CHECK constraints are activated only by modifications of the table they are based on, ASSERTIONs are activated by any modification that can possibly violate them.
- Choice of the most appropriate method for a particular IC is up to the DBA.
- Triggers respond to changes in the database. Can also be used to represent ICs.