



## Constraints and Triggers

---

---

---

---

---

---

---

---



## Topics discussed earlier

- basic constraints
  - entity integrity (not null)
  - referential integrity (foreign key)
  - key constraint (primary key)
  - candidate key constraint (uniqueness)
- Naming constraints
  - to add, modify, drop
  - deferrable constraints can be deferred

See Section 7.1 and 7.3 for details

---

---

---

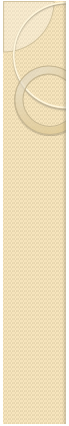
---

---

---

---

---



## Checks and Assertions

CHECK constraints  
attribute-level  
tuple-level

ASSERTIONS  
can include multiple tables

---

---

---

---

---

---

---

---

## Attribute Level Checks

```
create table enrolled (
  StudentID number(5),
  CourseID number(4),
  Quarter varchar(6)
  CHECK(quarter in ('Fall','Winter','Spring')),
  Year number(4), ...

create table memberof (
  StudentID number(5),
  GroupName varchar(40),
  Joined number(4)
  CHECK(Joined >= (SELECT Started
                    FROM student
                    WHERE studentID = SID)), ...
```

- has to be true or unknown (compare WHERE)
- checks get evaluated on a row when row is inserted/updated
  - so: checks may get violated
- subqueries not allowed in Oracle checks

---

---

---

---

---

---

---

---

---

---

## Tuple Level Checks

```
create table course (
  CID number(4),
  CourseName varchar(40),
  Department varchar(4),
  CourseNr char(3),

  primary key (CID),

  check (department <> 'CSC' OR CourseNR > 100)
);
```

- same as attribute level check, just different placement

---

---

---

---

---

---

---

---

---

---

## Examples

- enforce the following course ranges:  
CSC: 200-600, IT 100-500, GAM: 200-500
- a student must have either a first or last name
- we don't accept any undergraduate COMP-GPH and IT students after (and including) 2015
- only graduate students can be PhD students
- use to implement sub-classing

EER-modeling problem: Employees can be hourly, in which case we want their ID, name, Address, the day they were hired, and the rate at which they were hired. For salaried employees we want to store their ID, name, address, the day they were hired, and their annual salary and stock options. Consultants also get an ID, and we store their name, address, hiring date, contract number, and billing rate.

---

---

---

---

---

---

---

---

---

---

## Assertions

```
CREATE ASSERTION joined
CHECK (NOT EXISTS
      (SELECT *
       FROM student, memberof
       WHERE SID = StudentID and Joined < Started));
```

Example: ugrad/grad students can enroll in at most 2/4 courses a quarter

- not supported by anybody?
- can be mimicked using materialized views and/or triggers (procedural vs. declarative)

---

---

---

---

---

---

---

---

## Triggers

```
CREATE OR REPLACE TRIGGER started
BEFORE UPDATE OF started ON student
FOR EACH ROW
WHEN (new.started < old.started)
BEGIN
  :new.started := :old.started;
  DBMS_OUTPUT.PUT_LINE('Rejected change of started');
END;
/
```

```
SET SERVEROUTPUT ON;

UPDATE student
SET Started = 2001;

SELECT * FROM student;
```

---

---

---

---

---

---

---

---

## Triggers

```
CREATE OR REPLACE TRIGGER started
BEFORE UPDATE OF started ON student
FOR EACH ROW
WHEN (new.started < old.started)
BEGIN
  :new.started := :old.started;
  DBMS_OUTPUT.PUT_LINE('Rejected change of
started');
END;
/
```

← Create trigger

← triggering event

← attribute/table

← row trigger

← trigger restriction  
old: row before update  
new: row after update

← trigger action  
(in PL/SQL)

---

---

---

---

---

---

---

---

## Triggering Events

When do we trigger:

- before
- after
- instead of  
(only for views)

```
CREATE OR REPLACE TRIGGER started
BEFORE UPDATE OF started ON student
FOR EACH ROW
WHEN (new.started < old.started)
BEGIN
    :new.started := :old.started;
    DBMS_OUTPUT.PUT_LINE('Rejected
change of started');
END;
```

What is doing the triggering:

- insert, update, delete
- system events

---

---

---

---

---

---

---

---

## row/statement trigger

```
CREATE OR REPLACE TRIGGER started
BEFORE UPDATE OF started ON student
FOR EACH ROW
WHEN (new.started < old.started)
BEGIN
    :new.started := :old.started;
    DBMS_OUTPUT.PUT_LINE('Rejected
change of started');
END;
```

WHEN only  
for row-level  
triggers

:new/:old only  
for row-level  
triggers

vs

```
CREATE OR REPLACE TRIGGER started
AFTER UPDATE ON student
BEGIN
    DBMS_OUTPUT.PUT_LINE('Student
Table updated');
END;
```

---

---

---

---

---

---

---

---

## Restriction (WHEN)

- old  
(before change)
- new  
(after change)

```
CREATE OR REPLACE TRIGGER started
BEFORE UPDATE OF started ON student
FOR EACH ROW
WHEN (new.started < old.started)
BEGIN
    :new.started := :old.started;
    DBMS_OUTPUT.PUT_LINE('Rejected
change of started');
END;
```

---

---

---

---

---

---

---

---

## Trigger Action

- **BEGIN**  
*p/sql block*  
**END;**
- **old, :new**  
variables
- **dbms\_output**

```
CREATE OR REPLACE TRIGGER started
BEFORE UPDATE OF started ON student
FOR EACH ROW
WHEN (new.started < old.started)
BEGIN
  :new.started := :old.started;
  DBMS_OUTPUT.PUT_LINE('Rejected
change of started');
END;
```

---

---

---

---

---

---

---

---

---

---

## Example: Logging

```
CREATE OR REPLACE TRIGGER studentlog
AFTER INSERT OR UPDATE OR DELETE ON student
BEGIN
  DBMS_OUTPUT.PUT_LINE('Insert/Delete/Update on
Student Table');
END;
```

```
CREATE OR REPLACE TRIGGER started
AFTER INSERT OR UPDATE OR DELETE ON student
BEGIN
  IF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('Update on Student');
  ELSIF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('Insert on Student');
  ELSIF DELETING THEN
    DBMS_OUTPUT.PUT_LINE('Delete on Student');
  END IF;
END;
```

---

---

---

---

---

---

---

---

---

---

## Example: Unique IDs

ID fields often needed in tables

- Combinations of attributes can be unhandy
- No natural keys in the relation

Database support through **sequences**

- Each access gives new ID (by increasing the value)
- Microsoft Access: AutoNumber field (counter datatype)
- Oracle: sequence object

---

---

---

---

---

---

---

---

---

---

## Sequences in Oracle

```
--- create a new sequence for student table
CREATE SEQUENCE SEQ_STUDENT_ID
INCREMENT BY 1
START WITH 1;

--- example application
INSERT INTO student(SID, LastName,FirstName)
VALUES(seq_student_id.nextval, 'Pendleton', 'Gabriela');

--- drop sequence
DROP seq_student_id;
```

---

---

---

---

---

---

---

---

## Using sequence with trigger

```
--- create a new sequence for student table
CREATE SEQUENCE SEQ_STUDENT_ID
INCREMENT BY 1
START WITH 1;

--- create trigger to insert new ID automatically
CREATE OR REPLACE TRIGGER student_id_trigger
BEFORE INSERT ON student
FOR EACH ROW
BEGIN
    SELECT seq_student_id.nextval
    INTO :new.SID
    FROM dual;
END;
/
```

---

---

---

---

---

---

---

---

## Example: Logging into table

```
CREATE OR REPLACE TRIGGER studentlog
AFTER INSERT ON student
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('Insert on Student Table');
    insert into elog
    values(seq_student_id.nextval, :new.SID, 'I',
           systimestamp);
END;
/

CREATE TABLE elog(
    eid NUMBER,
    esid NUMBER(5),
    etype CHAR,
    etime DATE,
    PRIMARY KEY(eid)
);
```

---

---

---

---

---

---

---

---

## Example 7.13

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING
    OLD AS Oldtuple
    NEW AS Newtuple
FOR EACH ROW
WHEN (Oldtuple.networth > Newtuple.networth)
BEGIN
    UPDATE MovieExec
    SET netWorth = Oldtuple.netWorth
    WHERE cert# = Newtuple.cert#
END;
/
```

Not Oracle Syntax

potentially problematic code, why?

---

---

---

---

---

---

---

---

---

---

## Example 7.13

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING
    OLD AS Oldtuple
    NEW AS Newtuple
FOR EACH ROW
WHEN (Oldtuple.networth > Newtuple.networth)
BEGIN
    UPDATE MovieExec
    SET netWorth = Oldtuple.netWorth
    WHERE cert# = Newtuple.cert#
END;
/
```

Oracle compiles similar example, but rejects at runtime:

```
SQL Error: ORA-04091: table MSCHAEFER.STUDENT is mutating, trigger/function may
ORA-06512: at "MSCHAEFER.STARTED1", line 2
ORA-04088: error during execution of trigger 'MSCHAEFER.STARTED1'
04091. 00000 - "table %s.%s is mutating, trigger/function may not see it"
*Cause: A trigger (or a user defined plsql function that is referenced in
this statement) attempted to look at (or modify) a table that was
in the middle of being modified by the statement which fired it.
*Action: Rewrite the trigger (or function) so it does not read that table.
```

---

---

---

---

---

---

---

---

---

---

## Examples

- Extend the logging-into-table example, so it also logs updates and deletes
- write a trigger that cancels all deletions on the student table and writes a warning message that a deletion was attempted (need RAISE\_APPLICATION\_ERROR)
- if a student's program is PhD (update or insert), ensure the career is GRD (change if necessary)
- if a student is inserted without SSN, automatically assign a unique SSN starting with 900 (those SSNs are not currently in use)
- if a course is inserted with coursennr 666, allow the insert, but null the coursennr and issue a warning

---

---

---

---

---

---

---

---

---

---