

## Other constraints

- SQL lets you specify a variety of other constraints:
  - Local (refer to a tuple)
  - global (refer to tables)
- These constraints are enforced by a DBMS, that is, they are checked when a database is modify.
- Local constraints occur in the CREATE TABLE statement and use the keyword CHECK after attribute declaration:

```
CREATE TABLE T (...
    ....
    A <type> CHECK <condition>,
    B <type> CHECK <condition>,
    ....
)
```

## Local constraints

- Example: the value of attribute Rank must be between 1 and 5:  

```
Rank INT CHECK (1 <= Rank AND Rank <= 5)
```
- Example: the value of attribute A must be less than 10, and occur as a value of attribute C of relation R:

```
A INT CHECK (A IN
    SELECT R.C FROM R WHERE R.C < 10)
```

- Example: each value of attribute Name occurs precisely once as a value of attribute LastName in relation S:

```
Name VARCHAR(20) CHECK (1 = SELECT COUNT(*)
    FROM S
    WHERE Name=S.LastName)
```

## Assertions

- These assert some conditions that must be satisfied by the whole database.
- Typically assertions say that all elements in a database satisfy certain condition.
- All salaries are at least 10K:

```
CREATE ASSERTION A1 CHECK
    ( (SELECT MIN (Empl.Salary) FROM Empl >= 10000) )
```

- Most assertions use NOT EXISTS in them:  
SQL way of saying

“every  $x$  satisfies  $F$ ”

is to say

“does not exist  $x$  that satisfies  $\neg F$ ”

## Assertions cont'd

- Example: all employees of department 'sales' have salary at least 20K:

```
CREATE ASSERTION A2 CHECK
    ( NOT EXISTS (SELECT * FROM Empl
    WHERE Dept='sales' AND Salary < 20000) )
```

- All theaters play movies that are at most 3hrs long:

```
CREATE ASSERTION A2 CHECK
    ( NOT EXISTS (SELECT * FROM Schedule S, Movies M
    WHERE S.Title=M.Title
    AND M.Length > 180) )
```

## Assertions cont'd

- Some assertions use counting. For example, to ensure that table T is never empty, use:

```
CREATE ASSERTION T_notempty CHECK
  ( 0 <> (SELECT COUNT(*) FROM T) )
```

- Assertions are not forever: they can be created, and dropped later:  
DROP ASSERTION T\_nonempty.

## Triggers

- They specify a set of actions to be taken if certain event(s) took place.
- Follow the *event-condition-action* scheme.
- Less declarative and more procedural than assertions.
- Example: If an attempt is made to change the length of a movie, it should not go through.

```
CREATE TRIGGER NoLengthUpdate
AFTER UPDATE OF Length ON Movies
REFERENCING
  OLD ROW AS OldTuple
  NEW ROW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.Length <> NewTuple.Length)
SET Length = OldTuple.Length
WHERE title = NewTuple.title
```

## Analysis of the trigger

- AFTER UPDATE OF Length ON Movies specifies the **event**: Relation Movies was modified, and attribute Length changed its value.
- REFERENCING  
 OLD ROW AS OldTuple  
 NEW ROW AS NewTuple  
 says how we refer to tuples before and after the update.
- FOR EACH ROW – the trigger is executed once for each update row.
- WHEN (OldTuple.Length <> NewTuple.Length) – **condition** for the trigger to be executed (Length changed its value).
- SET Length = OldTuple.Length  
 WHERE title = NewTuple.title  
 is the **action**: Length must be restored to its previous value.

## Another trigger example

- Table Empl(emp\_id,rank,salary)
- Requirement: the average salary of managers should never go below \$100,000.
- Problem: suppose there is a complicated update operation that affects many tuples. It may initially decrease the average, and then increase it again. Thus, we want the trigger to run after *all* the statements of the update operation have been executed.
- Hence, FOR EACH ROW trigger cannot work here.

### Another trigger example cont'd

Trigger for the previous example:

```
CREATE TRIGGER MaintainAvgSal
AFTER UPDATE OF Salary ON Empl
REFERENCING
    OLD TABLE AS OldTuples
    NEW TABLE AS NewTuples
FOR EACH STATEMENT
WHEN ( 100000 > (SELECT AVG(Salary)
                FROM Empl WHERE Rank='manager') )
BEGIN
    DELETE FROM Empl
    WHERE (emp_id, rank, salary) in NewTuples;
    INSERT INTO Empl
        (SELECT * FROM OldTuples)
END;
```

### Analysis of the trigger

- AFTER UPDATE OF Salary ON Empl specifies the **event**: Relation Empl was modified, and attribute Salary changed its value.
- REFERENCING  
    OLD TABLE AS OldTuples  
    NEW TABLE AS NewTuples  
says that we refer to the set of tuples that were inserted into Empl as NewTuples and to the set of tuples that were deleted from Empl as OldTuples.
- FOR EACH STATEMENT – the trigger is executed once for the entire update operation, not once for each updated row.
- WHEN ( 100000 > (SELECT AVG(Salary)  
                    FROM Empl WHERE Rank='manager') )  
is the **condition** for the trigger to be executed: after the entire update, the average Salary of managers is less than \$100K.

### Analysis of the trigger cont'd

```
BEGIN
    DELETE FROM Empl
    WHERE (emp_id, rank, salary) in NewTuples;
    INSERT INTO Empl
        (SELECT * FROM OldTuples)
END;
```

is the **action**, that consists of two updates between BEGIN and END.

```
    DELETE FROM Empl
    WHERE (emp_id, rank, salary) in NewTuples;
deletes all the tuples that were inserted in the illegal update, and

    INSERT INTO Empl
        (SELECT * FROM OldTuples)
```

re-inserts all the tuples that were deleted in that update.