Tuple-Based Checks

Separate element of table declaration.

• Form: like attribute-based check.
• But condition can refer to any attribute of the relation.
  – Or to other relations/attributes in subqueries.
  – Again: Oracle forbids the use of subqueries.
• Checked whenever a tuple is inserted or updated.
Example

Only graduate students can take 600 level courses

CREATE TABLE student (
    level CHAR(2),
    dept CHAR(4),
    coursenum CHAR(20),
    CHECK(level = 'GS' or coursenum < 60000)
);

SQL Assertions

• Database-schema constraint.
• Not present in Oracle.
• Checked whenever a mentioned relation changes.
• Syntax:

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

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
  - Syntax illustrated here may not work exactly on your database system; check the system manuals.

Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - For example, **after update of takes on grade**
- Values of attributes before and after an update can be referenced
  - **referencing old row as** : for deletes and updates
  - **referencing new row as** : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. For example, convert blank grades to null.

```sql
create trigger setnull_trigger before update of takes
referencing new row as nrow
for each row
when (nrow.grade = ' ')
begin atomic
    set nrow.grade = null;
end;
```
Trigger to Maintain credits_earned value

- create trigger credits_earned after update of takes on (grade)
  referencing new row as nrow
  referencing old row as orow
  for each row
  when nrow.grade <> 'F' and nrow.grade is not null
  and (orow.grade = 'F' or orow.grade is null)
  begin atomic
    update student
    set tot_cred = tot_cred +
    (select credits
     from course
     where course.course_id = nrow.course_id)
    where student.id = nrow.id;
  end;

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 (Oracle Version)

Often called event-condition-action rules.

- **Event** = a class of changes in the DB, *e.g.*, “insertions into **Beers**.”

- **Condition** = a test as in a where-clause for whether or not the trigger applies.

- **Action** = one or more SQL statements.

- Differ from checks or SQL assertions in that:
  1. Triggers invoked by the event; the system doesn’t have to figure out when a trigger could be violated.
  2. Condition not available in checks.

Modification to Views Via Triggers

Oracle allows us to “intercept” a modification to a view through an instead-of trigger.

**Example**

```sql
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)
```

CREATE VIEW Synergy AS
SELECT Likes.drinker, Likes.beer,
Sells.bar
FROM Likes, Sells, Frequents
WHERE Likes.drinker = Frequents.drinker AND
      Likes.beer = Sells.beer AND
      Sells.bar = Frequents.bar;
```
CREATE TRIGGER ViewTrig INSTEAD OF INSERT ON Synergy FOR EACH ROW BEGIN
    INSERT INTO Likes VALUES (:new.drinker, :new.beer);
    INSERT INTO Sells(bar, beer) VALUES (:new.bar, :new.beer);
    INSERT INTO Frequents VALUES (:new.drinker, :new.bar);
END;

Options

1. Can omit OR REPLACE. But if you do, it is an error if a trigger of this name exists.
2. AFTER can be BEFORE.
3. If the relation is a view, AFTER can be INSTEAD OF.
   - Useful for allowing "modifications" to a view; you modify the underlying relations instead.
4. INSERT can be DELETE or UPDATE OF <attribute>.
   - Also, several conditions like INSERT ON Sells can be connected by OR.
5. FOR EACH ROW can be omitted, with an important effect: the action is done once for the relation(s) consisting of all changes.
Notes

• There are two special variables `new` and `old`, representing the new and old tuple in the change.
  - `old` makes no sense in an insert, and `new` makes no sense in a delete.
• Notice: in `WHEN` we use `new` and `old` without a colon, but in actions, a preceding colon is needed.
• The action is a PL/SQL statement.
  - Simplest form: surround one or more SQL statements with `BEGIN` and `END`.
  - However, select-from-where has a limited form.

Example

Maintain a list of all the bars that raise their price for some beer by more than $1.

```sql
CREATE TRIGGER PriceTrig
AFTER UPDATE OF price ON Sells
FOR EACH ROW
WHEN(new.price > old.price + 1.00)
BEGIN
  INSERT INTO RipoffBars
  VALUES(:new.bar);
END;
```

. run
• Triggers are part of the database schema, like tables or views.
• Important Oracle constraint: the action cannot change the relation that triggers the action.
  – Worse, the action cannot even change a relation connected to the triggering relation by a constraint, e.g., a foreign-key constraint.

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