Triggers (Oracle Version)

Often called event-condition-action rules.

- **Event** = a class of changes in the DB, e.g., “insert into Beers.”
- **Condition** = a test as in a where-clause for whether or not the trigger applies.
- **Action** = one or more SQL statements.
- Oracle version and SQL3 version; not in SQL2.
- Differ from checks or SQL2 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.
Example

Whenever we insert a new tuple into Sells, make sure the beer mentioned is also mentioned in Beers, and insert it (with a null manufacturer) if not.

```
Sells(bar, beer, price)

CREATE OR REPLACE TRIGGER BeerTrig
AFTER INSERT ON Sells
FOR EACH ROW
WHEN(new.beer NOT IN
    (SELECT name FROM Beers))
BEGIN
    INSERT INTO Beers(name)
    VALUES(:new.beer);
END;
```

run
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

- More information in on-line document or plsql.html

- 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.
• Dot and **run** cause the definition of the trigger to be stored in the database.

  ✦ Oracle 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.
Example

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

Sells(bar, beer, price)
RipoffBars(bar)

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
Modification to Views Via Triggers

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

Example

\[
\begin{align*}
\text{Likes}(\text{drinker}, \text{beer}) \\
\text{Sells}(\text{bar}, \text{beer}, \text{price}) \\
\text{Frequents}(\text{drinker}, \text{bar})
\end{align*}
\]

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;
.
run
SQL3 Triggers

- Read in text.
- Some differences, including:
  1. Position of \texttt{FOR EACH ROW}.
  2. The Oracle restriction about not modifying the relation of the trigger or other relations linked to it by constraints is not present in SQL3 (but Oracle is real; SQL3 is paper).
  3. The action in SQL3 is a list of SQL3 statements, not a PL/SQL statement.
SQL2 Assertions

- Database-schema constraint.
- Not present in Oracle.
- Checked whenever a mentioned relation changes.
- Syntax:
  
  ```sql
  CREATE ASSERTION <name>
  CHECK(<condition>);
  ```
Example
No bar may charge an average of more than $5 for beer.

Sells(bar, beer, price)

CREATE ASSERTION NoRipoffBars
CHECK(NOT EXISTS(
    SELECT bar
    FROM Sells
    GROUP BY bar
    HAVING 5.0 < AVG(price)
    )
));

- Checked whenever Sells changes.
Example

There cannot be more bars than drinkers.

Bars(name, addr, license)
Drinkers(name, addr, phone)

CREATE ASSERTION FewBar
CHECK(
    (SELECT COUNT(*) FROM Bars) <=
    (SELECT COUNT(*) FROM Drinkers)
);

• Checked whenever Bars or Drinkers changes.
Class Problem

Suppose we have our usual relations

\[
\text{Beers}(\text{name}, \text{manf}) \\
\text{Sells}(\text{bar}, \text{beer}, \text{price})
\]

and we want to maintain the foreign-key constraint that if you sell a beer, its name must appear in Beers.

1. If we don’t have foreign-key declarations available, how could we arrange for this constraint to be maintained:
   a) Using attribute-based constraints?
   b) Using SQL2 assertions?
   c) Using Oracle triggers?

2. What if we also want to make sure that each beer mentioned in Beers is sold at at least one bar?
PL/SQL

- Found only in the Oracle SQL processor (sqlplus).

- A compromise between completely procedural programming and SQL’s very high-level, but limited statements.

- Allows local variables, loops, procedures, examination of relations one tuple at a time.

- Rough form:

  ```
  DECLARE
      declarations
  BEGIN
      executable statements
  END;
  .
  run;
  ```

- DECLARE portion is optional.

- Dot and **run** (or a slash in place of **run;**) are needed to end the statement and execute it.
Simplest Form: Sequence of Modifications

Likes(drinker, beer)

BEGIN
   INSERT INTO Likes
      VALUES('Sally', 'Bud');
   DELETE FROM Likes
      WHERE drinker = 'Fred' AND beer = 'Miller';
END;

run;
Procedures

Stored database objects that use a PL/SQL statement in their body.

Procedure Declarations

CREATE OR REPLACE PROCEDURE
  <name>(<arglist>) AS
  <declarations>
  BEGIN
    <PL/SQL statements>
  END;

run;
• Argument list has name-mode-type triples.
  ✦ Mode: IN, OUT, or IN OUT for read-only, write-only, read/write, respectively.
  ✦ Types: standard SQL + generic types like \texttt{NUMBER} = any integer or real type.
  ✦ Since types in procedures must match their types in the DB schema, you should generally use an expression of the form

    \texttt{relation.attribute\%TYPE}

  to capture the type correctly.
Example

A procedure to take a beer and price and add it to Joe’s menu.

```
Sells(bar, beer, price)

CREATE PROCEDURE joeMenu(
    b IN Sells.beer%TYPE,
    p IN Sells.price%TYPE
) AS
    BEGIN
        INSERT INTO Sells
            VALUES(’Joe’’s Bar’, b, p);
    END;
.
run;
```

- Note “run” only stores the procedure; it doesn’t execute the procedure.
Invoking Procedures

A procedure call may appear in the body of a PL/SQL statement.

- Example:

  BEGIN
  
  joeMenu('Bud', 2.50);
  joeMenu('MooseDrool', 5.00);

  END;
  
  run;