

PL/SQL

Oracle's Version of Triggers and PSM

1

PL/SQL

- ◆ Oracle uses a variant of SQL/PSM which it calls PL/SQL.
- ◆ PL/SQL not only allows you to create and store procedures or functions, but it can be run from the generic query interface, like any SQL statement.
- ◆ Triggers are a part of PL/SQL.

2

Trigger Differences

- ◆ Compared with SQL standard triggers, Oracle has the following differences:
 1. Differences in order of elements.
 2. Action is a PL/SQL statement.
 3. New/old tuples referenced automatically.
 4. Strong constraints on trigger actions designed to make certain you can't fire off an infinite sequence of triggers.

3

Order of Oracle Trigger Elements

1. CREATE TRIGGER
2. Event, e.g., AFTER INSERT ...
3. FOR EACH ROW, if desired.
4. Condition.
5. Action.
6. A dot and the word "run". These cause the trigger to be installed in the database.

4

New/Old Tuples

- ◆ Instead of a REFERENCING clause, Oracle assumes that new tuples are referred to as "new" and old tuples by "old."
- ◆ Also, for statement-level triggers: "newtable" and "oldtable".
- ◆ In actions, *but not in conditions*, you must prefix "new," etc., by a colon.

5

Example: BeerTrig

- ◆ Recall our example BeerTrig, which inserted a beer name into Beers whenever a tuple was inserted into Sells with a beer that was not mentioned in Beers.
- ◆ Here's the Oracle version of that same trigger.

6

BeerTrig in Oracle SQL

```
CREATE OR REPLACE TRIGGER BeerTrig
  AFTER INSERT ON Sells
  WHEN (NOT IN
        (SELECT name FROM Beers))
  BEGIN
    INSERT INTO BEERS(name) VALUES ('');
  END;
```

Note position

Needed to store trigger as an element of the database

Notice "new" is understood. Also, colon used only in the action.

7

Another Example

- ◆ Recall PriceTrig, which stores in the relation Ripoffbars(bar) the name of any bar that raises the price of any beer by more than \$1.
- ◆ Here's the Oracle version.

PriceTrig in Oracle

```
CREATE OR REPLACE 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

9

Oracle Limitation on Relations Affected

- ◆ Each trigger is on some one relation R , mentioned in the event.
- ◆ The SQL standard puts no constraint on which relations, including R , can be modified in the action.
- ◆ As a result, infinite sequences of triggered events are possible.

Example: Infinite Triggering

- ◆ Let $R(x)$ be a unary relation that is a set of integers.
 - ◆ Easy to write a trigger with event INSERT ON R , that as action, inserts $i+1$ if i was the integer that awakened the trigger.
 - ◆ Results in a never-ending sequence of inserts.
- 11

Oracle Limitation

- ◆ Oracle is overly conservative about what relations can be changed when the event is on R .
 - ◆ R surely must not be subject to any modification in the action.
 - ◆ But much trickier: any relation that is linked to R by a chain of foreign-key constraints may not be changed either.
- 12

Example: Foreign-Key Chains

- ◆ Suppose $R.a$ is a foreign key, referencing $S.b$.
- ◆ Also, $T.c$ is a foreign key referencing $S.b$.
- ◆ Then in a trigger on relation R , neither T nor S may be modified.

13

PL/SQL

- ◆ In addition to stored procedures, one can write a PL/SQL statement that looks like the body of a procedure, but is executed once, like any SQL statement typed to the generic interface.
 - ◆ Oracle calls the generic interface "sqlplus."
 - ◆ PL/SQL is really the "plus."

14

Form of PL/SQL Statements

```
DECLARE
  <declarations>
BEGIN
  <statements>
END;
.
```

- ◆ The DECLARE section is optional.

15

Form of PL/SQL Procedure

```
CREATE OR REPLACE PROCEDURE
  <name> (<arguments>)
  <optional declarations>
BEGIN
  <PL/SQL statements>
END;
```

Notice AS needed here

Needed to store procedure in database

16

PL/SQL Declarations and Assignments

- ◆ The word DECLARE does not appear in front of each local declaration.
 - ◆ Just use the variable name and its type.
- ◆ There is no word SET in assignments, and := is used in place of =.
 - ◆ Example: $x := y;$

17

PL/SQL Procedure Parameters

- ◆ There are several differences in the forms of PL/SQL argument or local-variable declarations, compared with the SQL/PSM standard:
 1. Order is name-mode-type, not mode-name-type.
 2. INOUT is replaced by IN OUT in PL/SQL.
 3. Several new types.

18

PL/SQL Types

- ◆ In addition to the SQL types, NUMBER can be used to mean INT or REAL, as appropriate.
- ◆ You can refer to the type of attribute x of relation R by $R.x\%TYPE$.
 - ◆ Useful to avoid type mismatches.
 - ◆ Also, $R\%ROWTYPE$ is a tuple whose components have the types of R 's attributes.

19

Example: JoeMenu

- ◆ Recall the procedure $JoeMenu(b,p)$ that adds beer b at price p to the beers sold by Joe (in relation $Sells$).
- ◆ Here is the PL/SQL version.

20

Procedure JoeMenu in PL/SQL

```
CREATE OR REPLACE PROCEDURE JoeMenu (  
  b IN             
  p IN             
) AS  
  BEGIN  
    INSERT INTO Sells  
      VALUES ('Joe's Bar', b, p);  
  END;  
.  
run
```

Notice these types
have to be suitable
for the intended
uses of b and p .

21

PL/SQL Branching Statements

- ◆ Like IF ... in SQL/PSM, but:
- ◆ Use ELSIF in place of ELSEIF.
- ◆ Viz.: IF ... THEN ... ELSIF ... ELSIF ... ELSE ... END IF;

22

PL/SQL Loops

- ◆ LOOP ... END LOOP as in SQL/PSM.
- ◆ Instead of LEAVE ... , PL/SQL uses EXIT WHEN <condition>
- ◆ And when the condition is that cursor c has found no tuple, we can write $c\%NOTFOUND$ as the condition.

23

PL/SQL Cursors

- ◆ The form of a PL/SQL cursor declaration is:
CURSOR <name> IS <query>;
- ◆ To fetch from cursor c , say:
FETCH c INTO <variable(s)>;

24

Example: JoeGouge() in PL/SQL

- ◆ Recall JoeGouge() sends a cursor through the Joe's-Bar portion of Sells, and raises by \$1 the price of each beer Joe's Bar sells, if that price was initially under \$3.

25

Example: JoeGouge() Declarations

```
CREATE OR REPLACE PROCEDURE
  JoeGouge() AS
  theBeer Sells.beer%TYPE;
  thePrice Sells.price%TYPE;
  CURSOR c IS
    SELECT beer, price FROM Sells
    WHERE bar = 'Joe's Bar';
```

26

Example: JoeGouge Body

```
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO theBeer, thePrice;
    IF thePrice < 3.00 THEN
      UPDATE Sells
        WHERE bar = 'Joe's Bar' AND beer = theBeer;
    END IF;
  END LOOP;
  CLOSE c;
END;
```

How PL/SQL breaks a cursor loop

Note this is a SET clause in an UPDATE, not an assignment. PL/SQL uses := for assignments.

27

Tuple-Valued Variables

- ◆ PL/SQL allows a variable x to have a tuple type.
- ◆ x R%ROWTYPE gives x the type of R 's tuples.
- ◆ R could be either a relation or a cursor.
- ◆ $x.a$ gives the value of the component for attribute a in the tuple x .

28

Example: Tuple Type

- ◆ Here is the declarations of JoeGouge(), using a variable bp whose type is beer-price pairs, as returned by cursor c .

```
CREATE OR REPLACE PROCEDURE
  JoeGouge() AS
  CURSOR c IS
    SELECT beer, price FROM Sells
    WHERE bar = 'Joe's Bar';
  bp c%ROWTYPE;
```

29

JoeGouge Body Using bp

```
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO bp;
    EXIT WHEN c%NOTFOUND;
    IF bp.price < 3.00 THEN
      UPDATE Sells SET price = bp.price + 1.00;
      WHERE bar = 'Joe's Bar' AND beer = bp.beer;
    END IF;
  END LOOP;
  CLOSE c;
END;
```

Components of bp are obtained with a dot and the attribute name

30