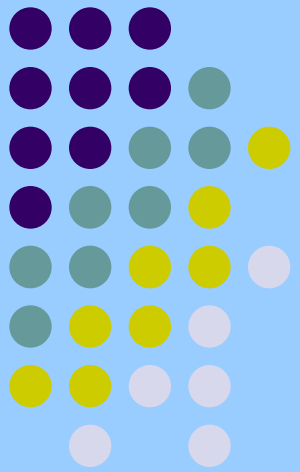
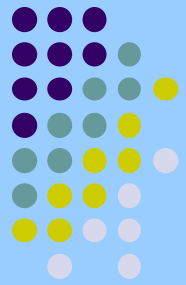
Advanced SQL: Triggers & Assertions



Triggers

Triggers: Introduction

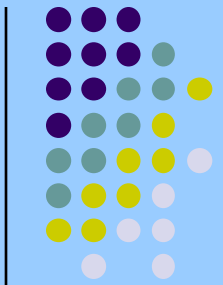


- The application constraints need to be captured inside the database
- **Some constraints can be captured by:**
  - Primary Keys, Foreign Keys, Unique, Not NULL, and domain constraints

```
CREATE TABLE Students
        (sid: CHAR(20),
         name: CHAR(20) NOT NULL,
         login: CHAR(10),
         age: INTEGER,
         gpa: REAL Default 0,
  Constraint pk Primary Key (sid),
   Constraint u1 Unique (login),
Constraint gpaMax check (gpa <= 4.0) );
```

These constraints are defined in ***CREATE TABLE*** or ***ALTER TABLE***

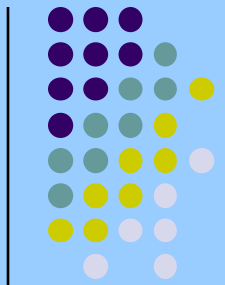
Triggers: Introduction



- **Other application constraints are more complex**
  - Need for assertions and triggers

- **Examples:**
  - Sum of loans taken by a customer does not exceed 100,000
  - Student cannot take the same course after getting a pass grade in it
  - Age field is derived automatically from the Date-of-Birth field

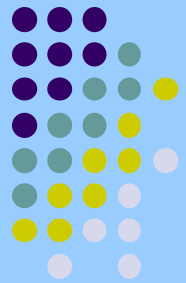
Triggers



- A procedure that runs automatically when a certain ***event*** occurs in the DBMS

- **The procedure performs some actions, e.g.,**
  - Check certain values
  - Fill in some values
  - Inserts/deletes/updates other records
  - Check that some business constraints are satisfied
  - Commit (approve the transaction) or roll back (cancel the transaction)

Trigger Components

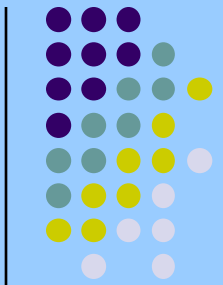


- **Three components**
  - **Event:** When this event happens, the trigger is activated
  - **Condition (optional):** If the condition is true, the trigger executes, otherwise skipped
  - **Action:** The actions performed by the trigger

- **Semantics**
  - When the **Event** occurs and **Condition** is true, execute the **Action**

Lets see how to define these components

Trigger: Events



- **Three event types**
  - Insert
  - Update
  - Delete
- **Two triggering times**
  - Before the event
  - After the event
- **Two granularities**
  - Execute for each row
  - Execute for each statement

1) Trigger: Event

Trigger name

Create Trigger *<name>*
Before|After **Insert|Update|Delete** **ON** <tablename>

That is the event

….

- Example

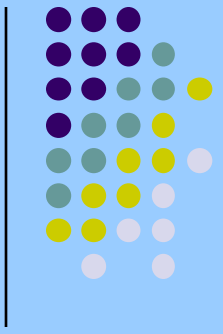
Create Trigger *ABC*
Before **Insert On** Students
….

This trigger is activated when an insert statement is issued, but before the new record is inserted

Create Trigger *XYZ*
After Update On Students
….

This trigger is activated when an update statement is issued and after the update is executed

Granularity of Event



- A single SQL statement may update, delete, or insert many records at the same time
  - **E.g., Update student set gpa = gpa x 0.8;**

- **Does the trigger execute for each updated or deleted record, or once for the entire statement ?**
  - **We define such granularity**

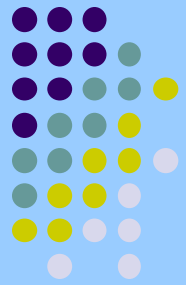
Create Trigger *<name>*
Before| After Insert| Update| Delete

For Each Row | For Each Statement
….

This is the event

This is the granularity

Example: Granularity of Event



Create Trigger *XYZ*
After Update ON <tablename>

For each statement
….

This trigger is activated once (per UPDATE statement) after all records are updated

Create Trigger *XYZ*
Before Delete ON <tablename>

For each row
….

This trigger is activated before deleting each record

2) Trigger: Condition

- This component is **optional**

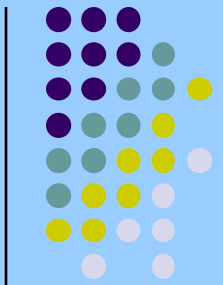
Create Trigger *<name>*
Before| After **Insert| Update| Delete On** *<tableName>*
For Each Row | For Each Statement

When *<condition>* ← That is the condition
…

If the employee salary > 150,000 then some actions will be taken

Create Trigger *EmpSal*
After Insert or Update On *Employee*
For Each Row
When *(new.salary >150,000)*
…

3) Trigger: Action



- **Action depends on what you want to do, e.g.:**
  - Check certain values
  - Fill in some values
  - Inserts/deletes/updates other records
  - Check that some business constraints are satisfied
  - Commit (approve the transaction) or roll back (cancel the transaction)

- **In the action, you may want to reference:**
  - The new values of inserted or updated records ***(:new)***
  - The old values of deleted or updated records ***(:old)***

Trigger: Referencing Values

- **In the action, you may want to reference:**
  - The new values of inserted or updated records ***(:new)***
  - The old values of deleted or updated records ***(:old)***

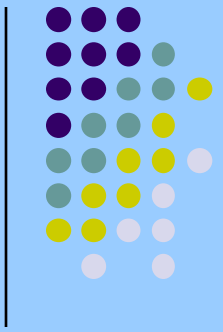
Trigger body

```
Create Trigger EmpSal
After Insert or Update On Employee
For Each Row
When (new.salary >150,000)
Begin
        if (:new.salary < 100,000) …
End;
```

Inside "When", the "new" and "old" should not have ":"

Inside the trigger body, they should have ":"

Trigger: Referencing Values (Cont'd)



- **Insert Event**
  - Has only :new defined
- **Delete Event**
  - Has only :old defined
- **Update Event**
  - Has both :new and :old defined
- **Before triggering (for insert/update)**
  - Can update the values in :new
  - Changing :old values does not make sense
- **After triggering**
  - Should not change :new because the event is already done

Example 1

If the employee salary increased by more than 10%, make sure the 'rank' field is not empty and its value has changed, otherwise reject the update

If the trigger exists, then drop it first

```
Create or Replace Trigger EmpSal
Before Update On Employee
For Each Row
Begin
      IF (:new.salary > (:old.salary * 1.1)) Then
          IF (:new.rank is null or  :new.rank = :old.rank) Then
              RAISE_APPLICATION_ERROR(-20004, 'rank field not correct');
          End IF;
      End IF;
End;
/
```

Compare the old and new salaries

Make sure to have the "/" to run the command

Example 2

If the employee salary increased by more than 10%, then increment the rank field by 1.

In the case of **Update** event only, we can specify which columns

```
Create or Replace Trigger EmpSal
Before Update Of salary On Employee
For Each Row
Begin
      IF (:new.salary > (:old.salary * 1.1)) Then
          :new.rank := :old.rank + 1;
      End IF;
End;
/
```

We changed the new value of ***rank*** field

The assignment operator has ":"

Example 3: Using Temp Variable

If the newly inserted record in employee has null hireDate field, fill it in with the current date

```
Create Trigger EmpDate
Before Insert On Employee
For Each Row
Declare
      temp date;
Begin
       Select sysdate into temp from dual;
       IF (:new.hireDate is null) Then
             :new.hireDate := temp;
       End IF;
End;
/
```

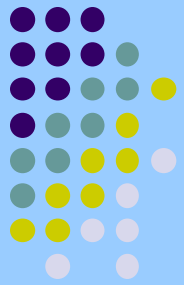
Since we need to change values, then it should be "Before" event

Declare section to define variables

Oracle way to select the current date

Updating the new value of hireDate before inserting it

Example 4: Maintenance of Derived Attributes



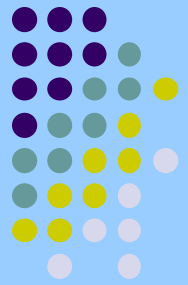
Keep the bonus attribute in Employee table always 3% of the salary attribute

```
Create Trigger EmpBonus
Before Insert Or Update On Employee
For Each Row
Begin
    :new.bonus := :new.salary * 0.03;
End;
/
```

Indicate two events at the same time

The bonus value is always computed automatically

Row-Level vs. Statement-Level Triggers



- **Example:** *Update emp set salary = 1.1 * salary;*
  - Changes many rows (records)

- **Row-level triggers**
  - Check individual values and can update them
  - Have access to ***:new*** and ***:old*** vectors

- **Statement-level triggers**
  - Do not have access to ***:new*** or ***:old*** vectors (only for row-level)
  - Execute once for the entire statement regardless how many records are affected
  - Used for verification before or after the statement

Example 5: Statement-level Trigger

Store the count of employees having salary > 100,000 in table R

Indicate three events at the same time

```
Create Trigger EmpBonus
After Insert Or Update of salary Or Delete On Employee
For Each Statement
Begin
    delete from R;
    insert into R(cnt) Select count(*) from employee where salary > 100,000;
End;
/
```

Delete the existing record in R, and then insert the new count.

Order Of Trigger Firing

Some Other Operations

- **Dropping Trigger**

```
SQL> Create Trigger <trigger name>;
```

- **If creating trigger with errors**

```
SQL > Show errors;
```

Assertions

Assertions

- An expression that should be always true
- When created, the expression must be true
- DBMS checks the assertion after any change that may violate the expression

create assertion <assertion-name> **check** <predicate>

Must return True or False

Example 1

- branch (branch_name, branch_city, assets)
- customer (customer_name, customer_street, customer_city)
- account (account_number, branch_name, balance)
- loan (loan_number, branch_name, amount)
- depositor (customer_name, account_number)
- borrower (customer_name, loan_number)

Sum of loans taken by a customer does not exceed 100,000

Must return True or False (not a relation)

```
Create Assertion SumLoans Check
    (   100,000 >= ALL
        Select   Sum(amount)
        From     borrower B , loan L
        Where  B.loan_number = L.loan_number
        Group By customer_name );
```

Example 2

- *branch (branch_name, branch_city, assets)*
- *customer (customer_name, customer_street, customer_city)*
- *account (account_number, branch_name, balance)*
- *loan (loan_number, branch_name, amount)*
- *depositor (customer_name, account_number)*
- *borrower (customer_name, loan_number)*

Number of accounts for each customer in a given branch is at most two

```
Create Assertion NumAccounts Check
    (   2 >= ALL
        Select   count(*)
        From     account A , depositor D
        Where  A.account_number = D.account_number
        Group By customer_name, branch_name );
```

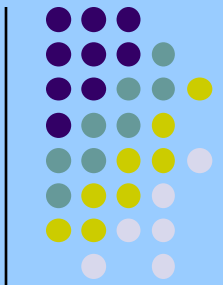
Example 3

- *branch (branch_name, branch_city, assets)*
- *customer (customer_name, customer_street, customer_city)*
- *account (account_number, branch_name, balance)*
- *loan (loan_number, branch_name, amount)*
- *depositor (customer_name, account_number)*
- *borrower (customer_name, loan_number)*

Customer city is always not null

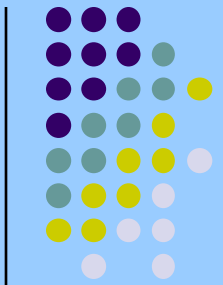
```
Create Assertion CityCheck Check
    (   NOT EXISTS (
        Select   *
        From     customer
        Where  customer_city is null));
```

Assertions vs. Triggers



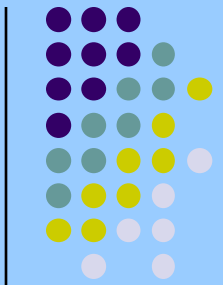
- Assertions do not modify the data, they only check certain conditions
- Triggers are more powerful because the can check conditions and also modify the data
- Assertions are not linked to specific tables in the database and not linked to specific events
- Triggers are linked to specific tables and specific events

Assertions vs. Triggers (Cont'd)



- All assertions can be implemented as triggers (one or more)
- Not all triggers can be implemented as assertions
- Oracle does not have assertions

Example: Trigger vs. Assertion



All new customers opening an account must have opening balance >= $100.
However, once the account is opened their balance can fall below that amount.

We need triggers, assertions cannot be used

Trigger Event: Before Insert

```
Create Trigger OpeningBal
Before Insert On Customer
For Each Row
Begin
    IF (:new.balance is null or :new.balance < 100) Then
        RAISE_APPLICATION_ERROR(-20004, 'Balance should be >= $100');
    End IF;
End;
```