Database Systems

Triggers

Triggers

- Active Database Systems
- Oracle Triggers
- Triggers for materialized view managament

Database Management Systems

Active Database Systems

Active database systems

- Traditional DBMS operation is *passive*
  - Queries and updates are explicitly requested by users
  - The knowledge of processes operating on data is typically embedded into applications
- *Active* database systems
  - Reactivity is a service provided by a normal DBMS
  - Reactivity *monitors* specific database events and *triggers* actions in response

Active database systems

- Reactivity is provided by automatically executing rules
- Rules are in the form
  - Event
  - Condition
  - Action
- Also called active or ECA rules

Active rules

- Event
  - Database modification operation
- Condition
  - Predicate on the database state
  - If the condition is true, the action is executed
- Action
  - Sequence of SQL instructions or application procedure

Rule engine

- Component of the DBMS, in charge of
  - Tracking events
  - Executing rules when appropriate
    - based on the execution strategy of the DBMS
- Rule execution is interleaved with traditional transaction execution

Example

- The active rule manages reorder in an inventory stock
  - when the stocked quantity of a product goes below a given threshold
  - a new order for the product should be issued
- Event
  - Update of the quantity on hand for product x
  - Insert of a new product x

Example

- The active rule manages reorder in an inventory stock
  - when the stocked quantity of a product goes below a given threshold
  - a new order for the product should be issued
- Condition
  - The quantity on hand is below a given threshold *and* there are no pending orders for product x
- Action
  - Issue an order with given reorder quantity for product x

Applications of active rules

- Internal applications
  - maintenance of complex integrity constraints
  - replication management
  - materialized view maintenance
- Business Rules
  - Incorporate into the DBMS application knowledge
    - E.g., reorder rule
- Alerters
  - widely used for notification

Triggers

- Commercial products implement active rules by means of *triggers*
- SQL provides instructions for defining triggers
  - Triggers are defined by means of the DDL instruction CREATE TRIGGER
- Trigger syntax and semantics are covered in the SQL3 standard
  - Some commercial products implement different features with respect to the standard

Trigger structure

- Event
  - Insert, delete, update of a table
  - Each trigger can only monitor events on a *single* table
- Condition
  - SQL predicate (it is optional)
- Action
  - Sequence of SQL instructions
  - Proprietary programming language blocks
    - e.g. Oracle PL/SQL
  - Java block

Execution process

When the events take place	[triggering]
If the condition is true	[evaluation]
Then the action is executed	[execution]

- Seems very simple but…
  - Execution modes
  - Execution granularity

Execution mode

- Immediate
  - The trigger is executed *immediately before* or *after* the triggering statement
- Deferred
  - The trigger is executed immediately *before commit*
- Only the immediate option is available in commercial systems

Execution granularity

- Tuple (or row level)
  - One separate execution of the trigger *for each tuple* affected by the triggering statement
- Statement
  - One single trigger execution *for all tuples* affected by the triggering statement

Granularity example

- Table T

A	B
1	5
2	9
8	20

- Transaction statement

```
UPDATE T
SET A=A+1
WHERE B<10;
```

- Trigger execution
  - A row level trigger executes twice
  - A statement level trigger executes once

Database Management Systems

Oracle Triggers

Trigger syntax

CREATE TRIGGER *TriggerName*

Mode Event {OR *Event* }

ON *TargetTable*

[[REFERENCING *ReferenceName*]

FOR EACH ROW

[WHEN *Predicate*]]

PL/SQL Block

Trigger syntax

```
CREATE TRIGGER TriggerName
Mode Event {OR Event }
ON TargetTable
[[ REFERENCING ReferenceName]
FOR EACH ROW
[WHEN Predicate]]
PL/SQL Block
```

- *Mode* is BEFORE or AFTER
  - Also INSTEAD OF but **it should be avoided**

Trigger syntax

```
CREATE TRIGGER TriggerName
Mode Event {OR Event }
ON TargetTable
[[ REFERENCING ReferenceName]
FOR EACH ROW
[WHEN Predicate]]
PL/SQL Block
```

- *Event* ON *TargetTable* is
  - INSERT
  - DELETE
  - UPDATE [OF *ColumnName*]

Trigger syntax

CREATE TRIGGER *TriggerName*
Mode Event {OR *Event* }
ON *TargetTable*
[[REFERENCING *ReferenceName*]
FOR EACH ROW
[WHEN *Predicate*]]
PL/SQL Block

- FOR EACH ROW specifies row level execution semantics
  - If omitted, the execution semantics is statement level

DBMG

Trigger syntax

```
CREATE TRIGGER TriggerName
Mode Event {OR Event }
ON TargetTable
[[ REFERENCING ReferenceName]
FOR EACH ROW
[WHEN Predicate]]
PL/SQL Block
```

- The old and new states of the row triggering a *row level* trigger may be accessed by means of the
  - OLD.*ColumnName* variable
  - NEW.*ColumnName* variable

Trigger syntax

CREATE TRIGGER *TriggerName*
Mode Event {OR *Event* }
ON *TargetTable*
[[REFERENCING *ReferenceName*]
FOR EACH ROW
[WHEN *Predicate*]]
PL/SQL Block

- To rename the state variables
  - REFERENCING OLD AS *OldVariableName*
    - similarly for NEW

Trigger syntax

CREATE TRIGGER *TriggerName*
Mode Event {OR *Event* }
ON *TargetTable*
[[REFERENCING *ReferenceName*]
FOR EACH ROW
[WHEN *Predicate*]]
PL/SQL Block

- *Only* for row level execution semantics (i.e., FOR EACH ROW)
  - A condition may be optionally specified
  - The old and new state variables may be accessed

Trigger syntax

CREATE TRIGGER *TriggerName*
Mode Event {OR *Event* }
ON *TargetTable*
[[REFERENCING *ReferenceName*]
FOR EACH ROW
[WHEN *Predicate*]]
PL/SQL Block

- The action is
  - a sequence of SQL instructions
  - a PL/SQL block
- *No* transactional and DDL instructions

Trigger semantics

- Execution modes
  - immediate before
  - immediate after
- Granularity is
  - row (tuple)
  - statement
- Execution is triggered by insert, delete, or update statements in a transaction

Execution algorithm

1. Before statement triggers are executed
2. For each tuple in *TargetTable* affected by the triggering statement
   a) Before row triggers are executed
   b) The triggering statement is executed
      + integrity constraints are checked on tuples
   c) After row triggers are executed
3. Integrity constraints on tables are checked
4. After statement triggers are executed

Example

- Trigger to manage reorder in an inventory stock
  - when the stocked quantity of a product goes below a given threshold
  - a new order for the product should be issued
- The following database schema is given

  Inventory (<u>Part#</u>, QtyOnHand, ThresholdQty, ReorderQty)

  PendingOrders(<u>Part#</u>, OrderDate, OrderedQty)

Example

- Trigger to manage reorder in an inventory stock
  - when the stocked quantity of a product goes below a given threshold
  - a new order for the product should be issued
- Event
  - Update of the quantity on hand for product x
  - Insert of a new product x
- Execution semantics
  - After the modification event
  - Separate execution for each row of the Inventory table

Trigger example

```
CREATE TRIGGER Reorder
AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory
FOR EACH ROW
```

Example

- Trigger to manage reorder in an inventory stock
  - when the stocked quantity of a product goes below a given threshold
  - a new order for the product should be issued
- Condition
  - The quantity on hand is below a given threshold

Trigger example

```
CREATE TRIGGER Reorder
AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory
FOR EACH ROW
WHEN (NEW.QtyOnHand < NEW.ThresholdQty)
```

Example

- Trigger to manage reorder in an inventory stock
  - when the stocked quantity of a product goes below a given threshold
  - a new order for the product should be issued
- Condition
  - The quantity on hand is below a given threshold

    and there are no pending orders for product x
    - This part cannot be introduced into the WHEN clause
- Action
  - Issue an order with given reorder quantity for product x

Example: Trigger body

```
DECLARE
 N number;
BEGIN
 select count(*) into N
 from PendingOrders
 where Part# = :NEW.Part#;
 If (N=0) then
    insert into PendingOrders(Part#,OrderedQty,OrderDate)
    values (:NEW.Part#, :NEW.ReorderQty, SYSDATE);
 end if;
END;
```

Complete trigger example

```
CREATE TRIGGER Reorder
AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory
FOR EACH ROW
WHEN (NEW.QtyOnHand < NEW. ThresholdQty)
DECLARE
 N number;
BEGIN
 select count(*) into N
 from PendingOrders
 where Part# = :NEW.Part#;
 If (N=0) then
    insert into PendingOrders(Part#,OrderedQty,OrderDate)
    values (:NEW.Part#, :NEW.ReorderQty, SYSDATE);
 end if;
END;
```

Triggers for materialized view maintenance

DBMG

Triggers for materialized view maintenance

- Materialized views are queries persistently stored in the database
  - provide increased performance
  - contain redundant information
    - e.g., aggregate computations
- Triggers are exploited to maintain redundant data
  - Propagate data modifications on tables to materialized view

Design example (3)

- Tables
  - Student S (SId, SName, DCId)
  - Degree course DC (DCId, DCName)
- Materialized view
  - Enrolled students ES (DCId, TotalStudents)
    - For each degree course, TotalStudents counts the total number of enrolled students
    - Defined by query

```
SELECT DCId, COUNT(*)
FROM S
GROUP BY DCId;
```

DBMG

Design example (3)

- Tables
  - Student S (<u>SId</u>, SName, DCId)
  - Degree course DC (<u>DCId</u>, DCName)
- Materialized view
  - Enrolled students ES (<u>DCId</u>, TotalStudents)
    - For each degree course, TotalStudents counts the total number of enrolled students
  - A new degree course is inserted in materialized view ES when the first student is enrolled in it
  - A degree course is deleted from ES when the last student quits it

Design example (3)

- Database schema

  S (<u>SId</u>, SName, DCId)

  DC (<u>DCId</u>, DCName)

  ES (<u>DCId</u>, TotalStudents)

- Propagate modifications on table S to materialized view (table) ES
  - Inserting new tuples into S
  - Deleting tuples from S
  - Updating the DCId attribute in one or more tuples of S

Design example (3)

- Design three triggers to manage separately each data modification
  - Insert trigger, delete trigger, update trigger
  - All triggers share the same execution semantics
- Execution semantics
  - *after* the modification takes place
    - Table ES is updated after table S has been modified
  - *row level*
    - Separate execution for each tuple of table S
      - significantly simpler to implement

Insert trigger (3)

- Event
  - insert on S
- No condition
  - It is always executed
- Action
  - if table ES contains the DCId in which the student is enrolled
    - increment TotalStudents
  - otherwise
    - add a new tuple in table ES for the degree course, with TotalStudents set to 1

Insert trigger (3)

```
CREATE TRIGGER InsertNewStudent
AFTER INSERT ON S
FOR EACH ROW
DECLARE N number;
BEGIN
 --- check if table ES contains the tuple for the degree course
 --- NEW.DCId in which the student enrolls -> COUNT the number of
 --- tuple and store the result into N
if (N <> 0) then
       --- the tuple for the NEW.DCId degree course is available in
       --- ES → UPDATE ES
       else
       --- no tuple for the NEW.DCId degree course available in ES
       --- → INSERT INTO ES
end if;
END;
```

Insert trigger (3)

```
CREATE TRIGGER InsertNewStudent
AFTER INSERT ON S
FOR EACH ROW
DECLARE
 N number;
BEGIN
 --- check if table ES contains the tuple for the degree
 --- course NEW.DCId in which the student enrolls
 select count(*) into N
 from ES
 where DCId = :NEW. DCId;
```

Insert trigger (3)

```
if (N <> 0) then
    --- the tuple for the NEW.DCId degree course is
    --- available in ES
    update ES
    set TotalStudents = TotalStudents +1
    where DCId = :NEW.DCId;
else
    --- no tuple for the NEW.DCId degree course is
    --- available in ES
    insert into ES (DCId, TotalStudents)
    values (:NEW.DCId, 1);
end if;
END;
```

Delete trigger (3)

- Event
  - delete from S
- No condition
  - It is always executed
- Action
  - if the student was the only student enrolled in the degree course
    - delete the corresponding tuple from ES
  - otherwise
    - decrement TotalStudents

Delete trigger (3)

```
CREATE TRIGGER DeleteStudent
AFTER DELETE ON S
FOR EACH ROW
DECLARE
 N number;
BEGIN
 --- read the number of students enrolled on the degree course
 --- OLD.DCId and store it into N
if (N > 1) then
      --- there are many enrolled students -> UPDATE ES
else
      --- there is a single enrolled student -> DELETE the tuple FROM ES
end if;
END;
```

Delete trigger (3)

```
CREATE TRIGGER DeleteStudent
AFTER DELETE ON S
FOR EACH ROW
DECLARE
 N number;
BEGIN
 --- read the number of students enrolled on
 --- the degree course OLD.DCId
 select TotalStudents into N
 from ES
 where DCId = :OLD.DCId;
```

Delete trigger (3)

```
 if (N > 1) then
      --- there are many enrolled students
      update ES
      set TotalStudents = TotalStudents – 1
      where DCId = :OLD.DCId;
else
      --- there is a single enrolled student
      delete from ES
      where DCId = :OLD.DCId;
end if;
END;
```

Update trigger (3)

- Event
  - Update of DCId on S
- No condition
  - It is always executed
- Action
  - update table ES for the degree course where the student *was* enrolled
    - decrement TotalStudents, or delete tuple if last student
  - update table ES for the degree course where the student *is currently* enrolled
    - increment TotalStudents, or insert new tuple if first student

Update trigger (3)

```
CREATE TRIGGER UpdateDegreeCourse
AFTER UPDATE OF DCId ON S
FOR EACH ROW
DECLARE
 N number;
BEGIN
--- read the number of students enrolled in
--- degree course OLD.DCId
 select TotalStudents into N
 from ES
 where DCId = :OLD.DCId;
```

DBMG

Update trigger (3)

```
if (N > 1) then
    --- there are many enrolled students
    update ES
    set TotalStudents = TotalStudents – 1
    where DCId = :OLD.DCId;
else
    --- there is a single enrolled student
    delete from ES
    where DCId = :OLD.DCId;
end if;
```

Update trigger (3)

```
--- check if table ES contains the tuple for the degree
--- course NEW.DCId in which the student is enrolled
 select count(*) into N
 from ES
 where DCId = :NEW. DCId;
```

Update trigger (3)

```
if (N <> 0) then
 --- the tuple for the NEW.DCId degree course is available in ES
        update ES
        set TotalStudents = TotalStudents +1
        where DCId = :NEW.DCId;
else
 --- no tuple for the NEW.DCId degree course is available in ES
        insert into ES (DCId, TotalStudents)
        values (:NEW.DCId, 1);
end if;
END;
```