

Triggers >Active Database Systems ➤ Oracle Triggers ➤ Differences between Oracle and DB2 Triggers ➤ Guidelides in writing triggers in Oracle ➤Trigger Design

1

Active Database Systems Triggers

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 \bullet Reactivity is a service provided by a normal DBMS \bullet Reactivity ${\it monitors}$ specific database events and ${\it triggers}$ 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 • Sequence of SQL instructions or application procedure

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

D₿G

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 stocked quantity for product x
 - · Insert of a new product x
- Condition
 - The stocked quantity of product x is below a given threshold and there are no pending orders for product x
- and the Action
 - Issue a new order of a pre-determined quantity for product x

7

6

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

8

Triggers

- Commercial products implement active rules by means of triggers
- SQL provides instructions for defining triggers
 - \bullet 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

9

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

DM

Execution process

When the events take place [triggering]

If the condition is true [evaluation]

Then the action is executed [execution]

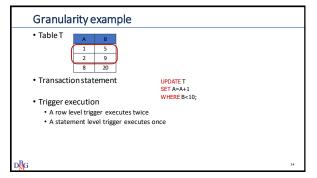
- Seems very simple but...
 - Execution modes
 - Execution granularity

₿G

• 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

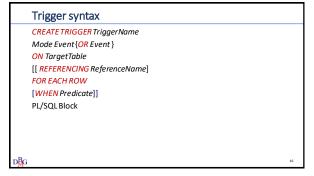
• Tuple (or row level)
• One separate execution of the trigger for each tuple affected by the triggering statement
• Statement
• One single trigger execution for all tuples affected by the triggering statement

12 13



Triggers in Oracle
Triggers

14 15



Trigger syntax

CREATE TRIGGER TriggerName

Mode Event {OR Event }
ON TargetTable

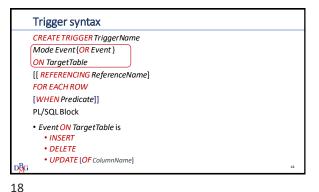
[[REFERENCING ReferenceName]
FOR EACH ROW

[WHEN Predicate]]
PL/SQL Block

• Mode is BEFORE or AFTER

• Also INSTEAD OF but it should be avoided

16 17



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

19

Trigger syntax CREATE TRIGGER Trigger Name 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 $\ensuremath{\textit{NEW}}$

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

21 20

Trigger syntax CREATE TRIGGER Trigger Name 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

22 23

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
- ${\it 4. \ \, After \, statement \, triggers \, are \, executed}$

24

Trigger semantics

- The execution order for triggers with the same event, mode and granularity is not specified
 - · it is a source of nondeterminism
- When an error occurs
 - · rollback of all operations performed by the triggers
 - rollback of the triggering statement in the triggering transaction

<u>p</u>g 25

Non termination

- Trigger execution may activate other triggers
 - Cascaded trigger activation may lead to non termination of trigger execution
- A maximum length for the cascading trigger execution may be set • default = 32 triggers
- If the maximum is exceeded
 - an execution error is returned

26

Mutating tables

- A mutating table is the table modified by the statement (i.e., event)
- triggering the trigger
- The mutating table
 - cannot be accessed in row level triggers
 - \bullet may $\emph{only}~$ be accessed in statement triggers
- $\bullet \ Limited\ access\ on\ mutating\ tables\ only\ characterizes\ Oracle\ applications$
 - accessing mutating tables is *always* allowed in SQL3

27

Example

- Trigger to manage reorder in an inventory stock
 - when the stocked quantity of a product goes below a given threshold
 - $\ensuremath{\bullet}$ 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)

1.2

Example

- $\bullet \ \mathsf{Trigger} \ \mathsf{to} \ \mathsf{manage} \ \mathsf{reorder} \ \mathsf{in} \ \mathsf{an} \ \mathsf{inventory} \ \mathsf{stock}$
 - $\ensuremath{^{\bullet}}$ 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

D₿G

Trigger example

CREATETRIGGER Reorder

AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory

FOR EACH ROW

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

 D_{G}^{B}

30 31

Trigger example

CREATE TRIGGER Reorder

AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory
FOR EACHROW

WHEN (NEW.QtyOnHand < NEW.ThresholdQty)

• 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

32 33

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 QLYONHAND OR INSERT ON Inventory
FOR EACH ROW

WHEN (INEW QLYONHAND < 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:

34 35

Concise comparison between Oracle and DB2 Triggers

Differences between Oracle and DB2 Yes Arbitrary More than one triggering event allowed No No

Guidelines in writing triggers in Oracle

38

36

Triggers Design

40

Guidelines in writing triggers in Oracle

- Execution Mode INSTEAD OF is allowed in Oracle but it should be avoided
- Usage of before triggers in Oracle to be compliant with the standard
 - Modifications of the NEW variable in tuples affected by the triggering statement are Modulinations of the New York of the New
 - allowed on before triggers
- · Before triggers cannot trigger other triggers

37

Trigger design

- The design of a single trigger is usually simple
 - Identify
 - execution semantics
 - event
 condition (optional)

action

Trigger design • Understanding *mutual* interactions among triggers is more complex • The action of one trigger may be the event of a different trigger • If mutual triggering occurs • Infinite execution is possible

Trigger execution properties

- Termination
- For an arbitrary database state and user transaction, trigger execution terminates in a final state (also after an abort)
- Confluence
 - For an arbitrary database state and user transaction, trigger execution terminates in a unique final state, independently of the execution order of triggers
- · Termination is the most important property
- · Confluence is enforced by deterministic trigger execution

 $D_{G}^{B}G$

42 43

Guaranteeing termination

- Termination is guaranteed at run time by aborting trigger execution after a given cascading length
- Termination may be verified at design time by means of the triggering
 - a node for each trigger

 D_{G}^{B}

- a directed edge T_i T_i if trigger T_i is performing an action triggering trigger T_i
- A cycle in the graph shows potential non terminating executions



Example

- Trigger managing salary amounts
 - · When a given average salary value is exceeded, a salary reduction is automatically
- $\bullet \ \ \text{The following table is given}$
- Employee (Emp#, Ename, ..., Salary)
- Execution semantics
 - · After the modification events · Separate execution for each update instruction
- No condition for execution

45

44

Example

CREATE TRIGGER Salary Monitor

AFTER UPDATE OF Salary ON Employee

FOR EACH STATEMENT

BEGIN

update Employee

set Salary = Salary * K

where 2500 < (select AVG (Salary) from Employee);

END;

The value of K may be

K = 0.9execution terminates K = 1.1infinite execution



Trigger applications

· replication management

• E.g., reorder rule

· widely used for notification

· materialized view maintenance

• maintenance of complex integrity constraints

· Incorporate into the DBMS application knowledge

Internal applications

• Business Rules

Alerters

47

Triggers for constraint management Triggers are exploited to enforce complex integrity constraints Design procedure Write the constraint as a SQL predicate It provides a condition for the trigger execution Identify the events which may violate the constraint I.e., the condition Define the constraint management technique in the action

 $D_{G}^{B}G$

Pesign example (1)

• The following tables are given

• Supplier S (S#, SName, ...)

• Part P (P#, PName, ...)

• Supply SP (S#, P#, Qty)

• Constraint to be enforced

• A part may be supplied by at most 10 different suppliers

48 49

Design example (1) • Constraint predicate select P# from SP group by P# having count(*) > 10 • set of parts violating the constraint • Events • insert on SP • update of P# on SP • Action • reject the violating transaction

Design example (1)

• Execution semantics

• ofter the modification

• statement level

• to capture the effect of the entire modification

• (Oracle) No condition

• The condition cannot be specified in the WHEN clause

• It is checked in the trigger body

• Design for Oracle trigger semantics

50 51

```
• The following tables are given
• Supplier S (S.B. SName, ...)
• Part P (P.B. PName, ...)
• Supply SP (S.B. P.B. Qty)
• Constraint to be enforced
• A part may be supplied by at most 10 different suppliers
```

```
Design example (1)

CREATE TRIGGER TOOMANYSuppliers

AFTER UPDATE OF P# OR INSERT ON SP

DECLARE

N number;

BEGIN
select count(*) into N
from SP
where P# IN (select P# from SP
group by P#
having count(*) > 10);

if (N <> 0) then
raise_application_error (xxx, 'constraint violated');
end if;

END;
```

PBG • The following tables are given • Supplier S (5世, SName, ...) • Part P (P世, PName, ...) • Supply SP (5世, P世, Qty) • Constraint to be enforced • The quantity of a product supply cannot be larger than 1000. If it is larger, trim it to 1000. • Check constraints do not allow compensating actions • Implement with a trigger

Design example (2)

• Constraint predicate
• Qty > 1000
• It is also the trigger condition

• Events
• insert on SP
• update of Qty on SP

• Action
• Qty = 1000

54 55

```
Pesign example (2)

Execution semantics

before the modification takes place
Its effect can be changed before the constraint is checked

row level

each tuple is modified separately
```

Design example (2)

CREATE TRIGGER ExcessiveQty
BEFORE UPDATE OF Qty OR INSERT ON SP
FOR EACH ROW
WHEN (NEW.Qty > 1000)
BEGIN
:NEW.Qty := 1000;
END;

57

56

```
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(*)

FROMS
GROUP BY DCId;
```

Posign example (3) Tables Student S(Sid, SName, DCId) Degree course DC (DCId, DCName) Materialized view Enrolled students S(DCId, 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)

Design example (3)

Design example (3)

Sid. SName, DCId)
DC (DCId. DCName)
ES (DCId. 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

61

60

```
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

ofter 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 DCld 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

62 63

```
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

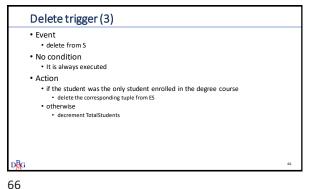
--- available in ES

insert into ES (DCId, TotalStudents)

values (:NEW.DCId, 1);

end if;

END;
```



Delete trigger (3) **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;

67

```
Delete trigger (3)
if (N > 1) then
   --- there are many enrolled students
   update ES
   set TotalStudents = TotalStudents - 1
   where DCId = : OLD. DCId;
   --- there is a single enrolled student
   delete from ES
   where DCId = : OLD. DCId;
end if;
END;
```

Update trigger (3) 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

68 69

```
Update trigger (3)
CREATE TRIGGER Update Degree Course
AFTER UPDATE OF DCId ONS
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;
```

```
Update trigger (3)
if (N > 1) then
   --- there are many enrolled students
   update ES
   set TotalStudents = TotalStudents - 1
   where DCId =: OLD. DCId;
   --- 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;
```