





# Triggers

0

## Triggers


- Active Database Systems
- Oracle Triggers
- Differences between Oracle and DB2 Triggers
- Guidelines in writing triggers in Oracle
- Trigger Design



1

# Active Database Systems


Triggers



2

## 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



3

## Active Database Systems


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



4

## 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



5

## 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



6

6

## 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
- Action
  - Issue a new order of a pre-determined quantity for product x



7

7

## 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

8

## 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



9

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



10

10

## 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



11

11

## 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

DBG

12

12

## 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

DBG

13

13

## 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

DBG

14

14

## Triggers in Oracle

Triggers

DBG

15

15

## Trigger syntax

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

DBG

16

16

## Trigger syntax

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

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

DBG

17

17

## 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)*

DBG

18

18

## 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

DBG

19

19

## 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*

DBG

20

20

## 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

DBG

21

21

## 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

DBG

22

22

## 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

DBG

23

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
4. After statement triggers are executed



24

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



25

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

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
  - *may only* be accessed in statement triggers
- Limited access on mutating tables only characterizes Oracle applications
  - accessing mutating tables is *always* allowed in SQL3



27

27

## 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 (Part#, QtyOnHand, ThresholdQty, ReorderQty)
  - PendingOrders(Part#, OrderDate, OrderedQty)



28

28

## 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



29

29

### Trigger example

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



30

30

### 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



31

31

### Trigger example

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



32

32

### 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



33

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;
```



34

34

### 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;
```



35

35

## Concise comparison between Oracle and DB2 Triggers



36

36

## Differences between Oracle and DB2

	Oracle	DB2
Reference to Old_Table and New_Table in statement triggers	No	Yes
When clause in statement triggers	No	Yes
Execution order between row and statement triggers with same mode	Specified	Arbitrary
Execution order between triggers with same event, mode and granularity	Unspecified	Creation Order
More than one triggering event allowed	Yes	No
Forbidden access to the mutating table	Yes for row triggers	No
Availability of the instead semantics	Yes	No
Database modifications allowed in before triggers	Yes	Only NEW variables



37

37

## Guidelines in writing triggers in Oracle



38

38

## 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 allowed in before triggers
  - Other databases modifications apart those reported in the previous point are not allowed on before triggers
  - Before triggers cannot trigger other triggers



39

39

## Triggers Design



40

40

## Trigger design

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



41

41

## Trigger design

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

DBG

42

42

## 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

DBG

43

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 graph
  - a node for each trigger
  - a directed edge  $T_i \rightarrow T_j$  if trigger  $T_i$  is performing an action triggering trigger  $T_j$
- A cycle in the graph shows potential non terminating executions



DBG

44

44

## Example

- Trigger managing salary amounts
  - When a given average salary value is exceeded, a salary reduction is automatically enforced
- The following table is given
  - Employee (EmpID, Ename, ..., Salary)
- Execution semantics
  - After the modification events
  - Separate execution for each update instruction
- No condition for execution

DBG

45

45

## Example

```
CREATE TRIGGER SalaryMonitor
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 = 0.9      execution terminates  
 K = 1.1      infinite execution



DBG

46

46

## Trigger applications

- 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

DBG

47

47



## Triggers for constraint management

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

DBG

48

48

## Design 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

DBG

49

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

DBG

50

50

## Design example (1)

- Execution semantics
  - *after* the modification
  - *statement level*
    - to capture the effect of the entire modification
    - (Oracle) to allow access to the mutating table
- (Oracle) No condition
  - The condition cannot be specified in the WHEN clause
  - It is checked in the trigger body
- Design for Oracle trigger semantics

DBG

51

51

## Design 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

DBG

52

52

## 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;
```

DBG

53

53

### Design example (2)

- The following tables are given
  - Supplier S (S#, SName, ...)
  - Part P (P#, PName, ...)
  - Supply SP (S#, P#, 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



54

54

### 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



55

55

### Design 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



56

56

### 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

57

### 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



58

58

### 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;
```



59

59

### 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
  - 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



60

60

### Design example (3)

- Database schema
  - S (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

61

### 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



62

62

### 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



63

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;
```



64

64

### 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;
```



65

65

## 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

DBG

66

66

## 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;
```

DBG

67

67

## 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;
```

DBG

68

68

## 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

DBG

69

69

## 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;
```

DBG

70

70

## 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;
```

DBG

71

71

### 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;

```



72

72

### 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;

```



73

73