Events and active concepts in database systems

Trigger in SQL:1999 (Oracle)
Active Database systems
Events in DBS: Motivation

- Assumption: Data stored in DB
- Architectures:

1. Alternative
   - New book
   - Amazon Web-interface

2. Alternative
   - New book
   - Amazon Web-interface
   - ENS
Events in DBS: Motivation

- Idea: Active Database System (ADBS)

  - Active database
    - Relational or object oriented database
    - Triggers actions in reaction on (system)events
    - ECA-Rules specify event, condition, action

New book
Events in DBS:

- **Conceivable events:**
  - Database state transitions
  - Temporal events
  - Abstract or external events

- **Examples:**
  - Relational: (Sybase, Ingres)
  - Object-relational: Postgres
  - Object-oriented: Sentinel, Ode [agr89], Samos[dit00]

- Simple triggers also in SQL:1999 (Oracle)
Events in DBS:

- **Argumentation for aDBS:**
  - central management of semantic in DB
  - optimization of processing
  - higher DBMS functionality
  - support of time-dependent requirements

- **Typical applications:**
  - monitoring of integrity constraints
  - access control
  - service for derived data within database (view update)
  - but also: trigger of external actions (notify administrator about weird actions on data)
Events in DBS: Roadmap

- Trigger in SQL:1999
  - Standard
  - Example: Oracle

- Active Database Systems
Trigger in SQL1999: Introduction

- Simplified integrity rules
- Simple conditions

CREATE TRIGGER <TriggerName>
 {before|after } {insert|delete|update}
 ON <RelationName>
 [referencing old as <OldName>, new as <NewName>]
 [WHEN <Condition>]
 [for each row|statement]
 <SQLStatement>

- called on insert/update/delete on specified relation
- references: binds variables to old/new tuples of a relation
- for each row: activates action for all selected tuples
- for each statement: activates action once for each condition
- Only for single table
Trigger in SQL1999: Trigger Example

- Example:
  - count the number of inserted books

```sql
create trigger BookCountPlus
  on insert on Book
  referencing old as Old
  new as New
  update BookCount
  set New.Number = Old.Number + 1
  for each row

create trigger BookCountMinus
  ...

  set New.Number = Old.Number - 1
```
Example:

- check that the budget holds
- just once per update statement

```
create trigger BudgetTest
    after update on Salary
    for each statement
    when (SumSalary > 200.000)
    signal "Budget overflow!"
```

Note: SumSalary is a function that has been predefined and is just used here
Trigger in SQL1999: Trigger in Oracle

— similar to stored procedures
— written in PL/SQL, Java (stored internally), or C (stored externally)

PL/SQL Block Structure:

DECLARE – Optional
  Variables, cursors, user-defined exceptions
BEGIN – Mandatory
  SQL statements
  PL/SQL statements
EXCEPTION – Optional
  Actions to perform when errors occur
END; – Mandatory

Stored procedure example:

DECLARE
  v_variable  VARCHAR2(5);
BEGIN
  SELECT column_name INTO v_variable FROM table_name;
  WHEN exception_name THEN
    ...
END;

PL/SQL variable
Bind variable: declared in host environment (e.g. SQL*Plus), reference in PL/SQL: :variable
Trigger in SQL1999: Trigger in Oracle

- **Supported Events:**
  - DML statements that modify data in a table (INSERT, UPDATE, or DELETE)
  - DDL statements
  - System events (startup, shutdown, and error messages)
  - User events such as logon and logoff

- **Syntax:**

```sql
CREATE [OR REPLACE] TRIGGER <trigger_name>
  {BEFORE|AFTER|INSTEAD OF}
  {INSERT|DELETE|UPDATE [OF column_list]}
  [[REFERENCING [NEW AS <new_row_name>]]
   [OLD AS <old_row_name>]]
  [FOR EACH ROW [WHEN (condition)]]
DECLARE <declarations>
BEGIN
  PL/SQL code
END;
```
Trigger in SQL1999: Trigger in Oracle

Trigger Applications:

- Integrity Test
- Referential Integrity Test
- Event Logging
- User Auditing
- Maintain table replicas
- Gather statistics
- Modify tables according to DML statements against views
- Security authorization
- Publish events to applications
Examples of Entity Integrity Triggers

CREATE OR REPLACE TRIGGER CUSTOMER_GET_KEY
BEFORE INSERT ON CUSTOMER FOR EACH ROW
DECLARE
    NEW_ID NUMBER;
BEGIN
    SELECT MAX(CUSTOMER_ID_NO) INTO NEW_ID FROM CUSTOMER;
    NEW.CUSTOMER_ID_NO := NEW_ID + 1;
END;

CREATE OR REPLACE TRIGGER customer_name_upper
BEFORE INSERT OR UPDATE OF name ON customer
FOR EACH ROW
BEGIN
    :new.name := UPPER(:new.name) ;
END;
Example of a Referential Integrity Trigger

```sql
CREATE OR REPLACE TRIGGER CUSTOMER_DEL_CHECK
BEFORE DELETE ON CUSTOMER FOR EACH ROW
DECLARE
    CUSTOMER_COUNT NUMBER;
BEGIN
    SELECT COUNT(CUSTOMER_ID_NO) INTO CUSTOMER_COUNT FROM SALE
    WHERE CUSTOMER_ID_NO = :OLD.CUSTOMER_ID_NO;
    IF (CUSTOMER_COUNT > 0) THEN
        RAISE_APPLICATION_ERROR(-20000,
            'Cannot delete customer because it has ' ||
            TO_CHAR(CUSTOMER_COUNT,'99999') || ' Sales.'));
    END IF;
END;
```

Non-PL/SQL variable
Example of a Integrity Constraint Trigger (using exception!)

```sql
CREATE OR REPLACE TRIGGER pos_cust_bal
    BEFORE INSERT OR UPDATE ON cust FOR EACH ROW
DECLARE
    neg_bal_error EXCEPTION;
BEGIN
    IF :new.balance < 0 THEN
        RAISE neg_bal_error ;
    END IF;
EXCEPTION
    WHEN neg_bal_error THEN
        RAISE_APPLICATION_ERROR
          (-20001, 'Negative Balance not allowed.');
END;
```
Trigger in SQL1999: Trigger in Oracle

Triggers vs. integrity constraints:

- **Oracle recommends to use triggers only to enforce:**
  - referential integrity when child and parent tables are on different nodes of a distributed database
  - complex business rules that are not definable using integrity constraints

- **Reason:**
  - integrity constrains are “all SQL” - easier, less errors
  - triggers are more complex to evaluate
  - Integrity constraints have better performance due to better optimization
Example of User Auditing Trigger

```sql
CREATE OR REPLACE TRIGGER audit_cust_trigger
  BEFORE UPDATE ON cust
  FOR EACH ROW
  WHEN (new.balance <> old.balance)
  BEGIN
    INSERT INTO audit_cust
    VALUES (:old.cust_no, :old.name,
            :old.address, :old.balance, sysdate);
  END;
```
Trigger in SQL1999: Trigger in Oracle

- Compiling Triggers: different to PL/SQL blocks

- PL/SQL block
  - compiled each time loaded into memory:
    - 1. Syntax checking + parse tree generation
    - 2. Semantic checking: Type checking etc. on the parse tree
    - 3. Code generation (pcode)

- Trigger
  - fully compiled at creation time (pcode then stored in the data dictionary)
  - firing: no opening of cursor, but direct execution
  - errors during compilation do not stop trigger creation (trigger firing fails + calling action fails)
Trigger in SQL1999: Trigger in Oracle

- **Modifying Triggers**
  - replace with new definition, or
  - drop and rerun `create trigger`.

- **Enabling and Disabling Triggers**
  - two modes: enabled (default) / disabled
  - Enabled: trigger executes its body if triggering statement is entered + trigger restriction is TRUE.
  - Disabled: trigger does not execute trigger body.

Disable or Re-enable a database trigger:

```
ALTER TRIGGER trigger_name DISABLE | ENABLE
```

Disable or Re-enable all triggers for a table:

```
ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS
```
Trigger in SQL1999: Trigger in Oracle

- Trigger Dependencies:
  - trigger become invalid if depended-on object is modified
  - depended-on objects: stored procedure or function called from the trigger body, other functions or packages
  - invalid triggers are recompiled when next invoked
  - if recompilation fails (object dropped) trigger becomes VALID WITH ERRORS

- Recompiling Triggers
  - manually: ALTER TRIGGER statement

```
ALTER TRIGGER trigger_name COMPILE
```
Triggers for remote sites:
- compiles at creation time
- fails if remote site not available at execution time

```
CREATE OR REPLACE TRIGGER Example
AFTER INSERT ON Emp_tab
FOR EACH ROW
BEGIN
    INSERT INTO Emp_tab@Remote VALUES ('x');
    EXCEPTION
    WHEN OTHERS THEN
        INSERT INTO Emp_log VALUES ('x');
END;
```

compilation fails here when dblink is inaccessible
Trigger in SQL1999: Trigger in Oracle

- Trigger publish events to applications:
  - support of database events:
    - DML on tables,
    - system events on Database and schema
  - users specify procedure to be run when the event occurs
  - uses Advanced Queueing publish/subscribe engine
    (based on JMS)
Trigger in SQL1999: Trigger in Oracle

- Drawback of Oracle/SQL3 Triggers:
  - no time events
  - restricted abstractions,
  - definition complicated, error source
  - difficult validation
  - SQL3: definition for single operation leads to multiple definition (test for budget on update AND insert necessary)
  - no deferred execution possible
  - cascading triggers, cycles possible
  - SQL:99 only primitive events on single tables
  - SQL:99 no events based on select-operations
  - SQL:99 conflict resolution problematic
Valuation (DB with SQL:99 Triggers)
- Often no sophisticated profiles possible
- No event patterns, events from different sources
- Restricted actions

- Use as simple ENS or in ENS-Application:

Functionality for ENS:
- Object/Event/Profile/Notification Repository
- Trigger for AS (active observer on suppliers side)
Events in DBS: Roadmap

- Trigger in SQL:1999
  - Standard
  - Example: Oracle

- Active Database Systems
Active DBS: Introduction

- Extension of triggers: active database rules
- ADB rules defined as ECA-rules:
  - Event - Condition - Action

  When \(<event\) expression >
  If \(<condition\) expression >
  Then \(<action\>
  Attributes \{priority,…\}

- Different implementations
  - Sybase: event and action part
  - Postgres: separate event, condition, action parts
  - Sentinel: separate event, condition, action parts
Active DBS: ECA rules

- "real" active Databases

A DBS is called active, if it is, additionally to common DBMS-functionality, capable to detect defined situations within the database (and outside) and to trigger defined reactions. [dit00]

- Basic ECA-Rule:

```
DEFINE RULE <rule_name>
    ON <event_clause>
    IF <cond_clause>
    DO <action_clause>
    <execution constraints>
```
Example: After 3 inserts on the same account transfer the money according defined strategy to 3 (depot) accounts

```
DEFINE EVENT EventInv_BankAccountTransfer
  TIMES (3, Kunde.AccountInsert):SAME OBJECT

DEFINE RULE EventInv_BankAccountTransfer
  ON EventInv_BankAccountTransfer
  DO {
    S = BankAccount.saldo + Inv_Account1.saldo + 
    Inv_Account1.saldo;
    Insert_1 = (S* BankAccount.ratio * 0.01)- BankAccount.saldo;
    Insert_2 = (S* Inv_Account1.ratio * 0.01)- Inv_Account1.saldo;
    Insert_3 = (S* Inv_Account2.ratio * 0.01)- Inv_Account2.saldo;
    AccountInsert(Insert_1);
    Inv_Account1Insert(Insert_2);
    Inv_Account1Insert(Insert_3);
  }
```
Active DBS: ECA rules

- **Management of rules:**
  - define parts separately
  - delete rule or parts
  - switch status of rule

---

**DEFINE EVENT** `<event_name>`

**DEFINE CONDITION** `<cond_name>`

**DEFINE ACTION** `<actions_name>`

**DELETE RULE** `<rule_name>`

**DELETE EVENT** `<event_name>`

**DELETE CONDITION** `<cond_name>`

**DELETE ACTION** `<actions_name>`

**DISABLE RULE** `<rule_name>`

**ENABLE RULE** `<rule_name>`

---

**e.g** disable rule for bulk load or backup

```
DISABLE RULE R1
EVERY WEEK [Fr 18:00, Mo 08:00]
```
Active DBS: Active DBS vs Triggers

Difference to triggers?

- Complex event definitions (e.g. on several tables)
- time events
- composite events
- external events
- conflict resolution strategy
- various coupling modes
- ...

Active DBS: Event Specification

- Event class/type: rules define event types

- Event Instance:
  - actual occurrence of event of event class
  - specification describes event (what happened)
  - has occurrence time (when)

\[
\text{Event} = (\text{event specification}, \text{event time})
\]

Note:
- recorded occurrence time depends on system clock
- occurrence time describes time of event detection
- explicit \((what \ and \ when)\) vs. implicit \((what)\) event description
- primitive or composed events
Active DBS: Event Specification

**Primitive Events:**

- **DB events** = begin or end of an DB operation execution DB (select, insert, delete update in relational DBMS)

  ```plaintext
  DEFINE EVENT E1 BEFORE INSERT
  ```

- **DBMS events** = begin or end of operations on DB (transaction start, end, abort; user login, ...)

  ```plaintext
  DEFINE EVENT E2 BankAccountInsert.COMMIT_TA
  ```

- **Time events** = absolute, periodically repeated or relative definition of point in time

  ```plaintext
  DEFINE EVENT E3 EVERY 10 DAYS 16:15
  ```

- **Abstract events** = external events in system applications, defined by identifier, explicitly announced to the DB

  ```plaintext
  DECLARE AccountCheck
  RAISE AccountCheck
  ```
Active DBS: Event Specification

Composite events:

based on primitive events, defined by event algebra

- **Disjunction** $E = E_1 \mid E_2$:
  - $E$ occurs if either $E_1$ or $E_2$ occurs, $E.time = \min(E_1.time \text{ or } E_2.time)$

- **Sequence** $E = E_1 ; E_2$:
  - $E_2$ occurs after $E_1$ ($E_1.time < E_2.time$), $E.time = E_2.time$

- **Conjunction** $E = E_1 , E_2$:
  - $E_1$ and $E_2$ occur, $E.time = \max(E_1,E_2)$

- **Negation** $\text{NOT } E$:
  - $E$ did not occur within interval $[t\_start,t\_end]$, $E.time = t\_end$

- **Reduction**:
  - $^*E$, $^E \text{ WITHIN } I$, TIMES($n,E$),
  - TIMES($n\_min,n\_max,E$) WITHIN $I$
Active DBS: Event Specification

Semantics of Event Order:

Rule: “fire if e1; e2”

Traces:
- e1 e3 e2 - fire?
- e1 e1 e2 - fire once or twice?
- e1 e2 e2 - fire once or twice?
- e1 e2 e1 e2 - fire two or three times?

- one trace for DBMS / transaction / user?
- Rule execution needs clear semantic
- State: rule executed differently in different aDBMS
Active DBS: Condition and Action

**Condition:**
- predicate over database state
- defined as SQL-query, WHERE-clause, method or application-procedure
- reference to old and new state

**Action:**
- program part with operations on DB and/or other operations
- DB operations (e.g. update, insert, select, delete),
- DBMS operations (e.g. abort transaction)
- stored procedures
- method call or application-operations
- alternative (do instead action_clause)
- rule operations (e.g. definition, change, activation of rules)
Active DBS: Condition and Action

Example:

```plaintext
DEFINE EVENT EventAccountInsert Kunde.AccountInsert

DEFINE EVENT EventSavings
  ( NOT EventAccountInsert
      WITHIN [EventAccountInsert, EventAccountInsert+ 1 MONTH]): SAME OBJECT

DEFINE RULE SavingsProblems
  ON EventSavings
  DO { alerter ("Warning: You have not saved much") }  
```

If after the last account insert nothing has been inserted for one month the customer is to be notified personally.
Active DBS: Rule execution

- Rule execution Model:
  - Activity 1: infinite loop for event detection and rule triggering
  - Activity 2: if triggered rule
    choose rule R
    evaluate condition C of R
    if C true, do action A of R

- Semantic of execution (directions):
  - granularity: for each tuple, operation, transaction,...
  - element or set-based execution (e.g. each row)
  - conflict resolution
  - coupling modes
  - iterative or recursive execution
  - cascading rules
Active DBS: Conflict Resolving [dia97]

- **Conflict:** More than one rule for one event
  - fundamental to controlling of aDBMS
  - strongly influences result

- **Problems:**
  - rule order influences final DB state,
  - rule order impact on performance
  - Goal: ensuring confluent rule sets = order of firing without impact on final DB state
Active DBS: Conflict Resolving

- **Clear Conflict resolution mechanism needed:**
  - large # rules -> complex and complicated interactions
  - large # users to create rules: need for guidelines and control
  - varying applications impose distinct strategies for conflict resolution

- **Strategies:**
  - random rule order
  - order by criteria such as creation time, ...
  - execution priority defined by user
  - concept-based rules: (different priority levels based on functionality)

- **Status:** no flexible conflict resolution mechanisms
Active DBS: Conflict Resolving

- Example: execution priority defined by user:
  - supported in most aDBMS
  - for each rule, defined when created

- Problems:
  - different priority criteria by authors

  Rule: if constraint violated reject update
  Criterion: order based on complexity of constraint (performance!)
  Problem: user-dependency, multi-user environment problematic

  - Per-rule priority mechanism too low-level for large rule sets
  - definition at rule-creation time: parameters sometimes known at execution time,
    e.g. workload in real-time DB decisive for rule-order
Active DBS: Coupling

Coupling modes: ECA-rules and Transactions

- temporal coupling (immediate)

- temporal coupling (deferred)

- contextual coupling (decoupled)
Active DBS:

**Coupling modes: E-C / C-A** (any combination possible)

- **immediate/deferred**
  - Condition evaluation
  - Action

- **deferred/decoupled**
  - Condition evaluation
  - Action

- **deferred/decoupled**
  - Condition evaluation
  - Action
Active DBS: SQL99-Execution Modes

- Cascading rule execution
  - Events occur during action and trigger new rules
  - When is the new rule executed?

- Iterative execution:
  - complete action
  - insert new rules (according priority) in list of “waiting” rules

- Recursive execution:
  - interrupt action
  - execute all immediate rules
Active DBS: SQL99-Cascading Rules

- Problem: no termination of recursive rule triggering

- Avoid during rule definition
  - complicated,
  - needs methodology
  - models: finite state machine, petri-nets
  - and tool-support, e.g. rule analyzer

- Use system-internal restrictions
  - syntactical rule restriction,
  - abort if \#cycles > threshold
Active DBS: Rule optimization

- **Situation:** temporally persistent rules, evaluated many times

- **Problem:**
  - potentially large set of predefined queries
  - possible overhead on every event

- **Approaches:**
  - conventional query optimization
  - grouping of rules (multiple query optimization)
  - materializing intermediate results
  - rule buffering strategies for real-time applications
  - parallelism
Active DBS: Evaluation Criteria [cha93]:

- Rule expressiveness
  - Supported events, Event operators

- Execution semantics
  - Coupling modes, Cascaded rule execution, Multiple rule execution, Priorities,

- Optimization of rules

- Architectural approach

Postgres
- database events: retrieve, replace, delete, append, new, old
- temporal events: time and date
- disjunction operator
- immediate reaction on event
- cascaded rule execution supported
- multiple rule execution with user defined priorities
- rule optimization (e.g. query rewrite)
- object-relational architecture
Active DBS: Problems

- Definition complicated, error source
- Difficult validation
- Only restricted abstractions supported
- Definition for single operation may lead to multiple definitions
- No deferred execution possible
- Cascading triggers, cycles possible
- Conflict resolution problematic
Do we need all that active features within the DB?

- Start 1953: trigger model for System R (first RDBS)
- 1982: term "active DB" introduced for system with automatic view update

**RDBMS:**
- restricted events (only DB operations)
- different approaches for rule execution
- specialized for integrity constrains, view update, ...

**ODBMS:**
- most concepts first proposed in HiPAC project [day88]
- support of special events in oo-context
- method events, internal and external rules for classes
- Encapsulation and Inheritance for rules
Active DBS: Current State + Research

Do we need all that active features within the DB?

Trend: unbundling [gep98]

Idea:

- DBMS with simple basic functionality
- additional features in separate cooperating components, e.g. Event/Reaction-Service
- possible interaction of DBMS and service:

Event-Action Service
(extension of ENS)