Applications

- Production control, e.g., power plants, ...
- Maintenance tasks, e.g., inventory control, ...
- Financial applications, e.g., stock & bond trading, ...
- Network management
- Air traffic control
- Program trading
- Computer Integrated Manufacturing (CIM)
“Passive” Database Systems

- Example: Traveling

3 complaints about Tunesia

DBMS

3 complaints > 10% of the reservations?

YES

database

complaints

Tunisia 2013

“Passive” Database Systems - 2

Two approaches:

- Polling (periodical queries)
  - too infrequently: missing the response time window
  - too frequently: flooding the systems with queries returning empty answers

- Augmentation of every program updating the database to check the situation being monitored
  - software modularity is compromised
  - what happens in case of “ad hoc” updates?
**Motivation**

- Overcome the strict separation between application programs and DBS.

- Usually only a small part of the real-world semantics can be modeled in the DBS.

- Object-oriented DBS are not enough => add active (and deductive) mechanisms to model more semantics (especially dynamic behavior) of the applications in DBS.

---

**Active Database Systems**

- Example: Traveling

![Diagram showing an example of active database system for travel complaints in Tunisia.](image)
Active Database Systems - 2

• General Idea

![Active Database System](image)

in addition to the capabilities of passive database systems

• monitor specified **situations** (events & conditions) in the database or its environment

• invoke specified **reactions** whenever a situation occurs programs containing, e.g., database operations

---

**Definition: Active DBS**

An active database system (ADBS) is a DBS that monitors situations of interest and, when they occur, triggers an appropriate response in a timely manner.

The desired behavior is expressed in **production rules** (also called **event-condition-action rules**), which are defined and stored in the DBS.

This has the benefits that the rules can be shared by many application programs, and the DBS can optimize their implementation.
Active DBS

Rule Models and Languages

- Event Specification
- Condition Specification
- Action Specification
- Event-Condition-Action Binding
- Rule Ordering
- Rule Organization

affects data model and transaction management
Production Rules (Event-Condition-Action Rules)

ON event has occurred
IF condition holds
THEN execute action

“situation”
insert into complaints list
#complaints > 0.1*(#reservations)

“reaction”
reduce price

Event-Condition-Action Rules

Event Specification

- relational DBS:
  define rule MonitorNewEmps
  on insert to employee
  if ... then ...

- OODBS:
  define rule CheckRaise
  on employee.salary-raise()
  if ... then ...

- rule triggered by data retrieval:
  define rule MonitorSalAccess
  on retrieve salary
  from employee
  if ... then ...
Knowledge Model: Semantics of ECA Rules

**primitive events:**
- begin/end of database operations (depending on data model)
  - ON AFTER insert (complaints)
  - ON BEFORE Jur_Person.buy_share_from
- begin/end of DBMS operations
  - ON BOT
  - ON EOT
- temporal events - time specifications (absolute, relative, periodical/repeated)
  - ON 21/11-96, 18:15
  - ON E1 + 10
  - ON EVERY DAY 16:00
- abstract events (define, raise)
  - explicitly defined by user (specified for one application)
    - define program_start
    - ON program_start
    - raise program_start

Knowledge Model: Semantics of ECA Rules - 2

**composite events (-> algebra):**
- disjunction (OR): E1 | E2
- sequence (ordered): E1; E2
- conjunction (AND): E1, E2
- negation (NOT): NOT E <time_interval>
- history operator: TIMES (n, E) <time_interval>

Example:
- ((E1, E2) | E3); (E4 | E5); NOT E6
Knowledge Model: Semantics of ECA Rules - 3

ON event has occurred

IF condition holds

DO execute action

predicate (WHERE) over database state or query

programs, including database operations or notifications to the users

Condition Specification

The condition part of a rule specifies a predicate or query over the data in the database.

The condition is satisfied if the predicate is true or if the query returns a nonempty answer.

explicit events: condition may often be omitted (in which case it is always satisfied).

transition conditions: allow to express conditions over changes in the database state. Example ->

```
define rule MonitorRaise
  on update to employee.salary
  if employee.salary > 1.1 * old employee.salary
  then ...
```
Action Specification

The action part of a production rule specifies the operations to be performed when the rule is triggered and its condition is satisfied.

define rule FavorNewEmps
  on insert to employee
  then delete employee e where e.name = employee.name

Example of Rule Implementation (INGRES)

CREATE RULE ComplaintCheck <- name
  AFTER INSERT OF complaints <- event
  EXECUTE PROCEDURE InsertComplaint (mdestination = new.destination);

CREATE PROCEDURE InsertComplaint (mdestination VARCHAR (30))
  DECLARE check_complaint, check_book INTEGER;
  msg VARCHAR (80) NOT NULL;
  BEGIN
    SELECT COUNT (*) INTO check_complaint <- condition
    FROM complaints WHERE destination = mdestination;
    SELECT COUNT (*) INTO check_book
    FROM reservations WHERE destination = mdestination;
    IF check_complaint > check_book/10 THEN <- action
      msg = "Error1: destination + :mdestination +
      has the minimum of complaints exceeded";
      MESSAGE :msg;
      INSERT INTO destination_log VALUES (:msg);
    ENDIF;
    UPDATE destination SET price =: price * 3/4
    WHERE name = mdestination;
  END;
## Execution of Rules

- When should a rule be executed?
  - **coupling modes**:
    - immediate
    - deferred
    - decoupled

- Execution order of multiple rules can be specified by **priorities**

=> Extension of rule definition with definition of coupling modes and priorities

## Rule Ordering

different ways of conflict resolution for rule processing:

- only 1 rule can be triggered (at a time)
  => no conflict resolution needed

- n rules can be triggered (at a time):
  - arbitrary
  - numeric priorities
  - exception hierarchies
  - partially ordered
  - multiple rules can be executed concurrently
  - ...

©2002 Vera Goebel
Rules in DBS (Active DBS)
Rule Organization
The number of rules in an active DBS can get very large.

=> mechanisms needed for:

- organization and management of collections of rules,
- selective activation/deactivation of rules.

Rule Execution Semantics
The rule execution semantics determine how rule processing will take place at runtime once a set of rules has been defined, including how rules will interact with the arbitrary database operations and transactions that are submitted by users and application programs.

Precise execution semantics are important because even for relatively small rule sets, rule behavior can be complex and unpredictable.

Alternatives for rule execution:
- granularity of rule processing
- 1 event => 1-n rules triggered
- interplay between rule execution and execution of user-initiated transactions
Implementation Issues

Alternatives to realize a DBS with rule support:

- **Loose coupling**: extension of interpreter of a rule language with DBMS calls on top of a DBMS.

- **Tight coupling**: interface (data model) to users and application programs only rule language. DBMS “hidden” from the user/app.

- **Integration**: DBMS internally extended with rule definition language and inference mechanism. This requires the modification of the source code of an existing DBMS or totally new implementation of a DBMS.

Systems

- **Relational active DBS**: Ariel, Postgres, STARBURST, ... *(research prototypes)*
  Ingres, InterBase, Oracle, Rdb, Sybase, ... *(commercial products)*

- **Object-oriented active DBS**: HiPAC, DOM, Ode, Adam, Sentinel, SAMOS, ... *(research prototypes)*
  ... *(commercial products)*
Active DBS: Relational Standards (SQL2, SQL3)

- both permit tuple-level and set-level processing of assertions and triggers
  (the choice is made at rule definition time by specifying “for each row”)

- assertions can be defined with a database modification command as the triggering
  event or with the special event “before commit”, in both cases: condition evaluation is
  done over the current state of the database + the changes of the current transaction

- triggers are initiated by database modifications (can refer to the old or the new
  database state)

- rule processing is strictly sequential (no conflict resolution necessary because no 2
  rules can be defined to have the same triggering event)
  +
  additional syntactic restrictions on rule definitions ensure that the same table cannot be
  modified multiple times in a sequence of rule firings, thereby ensuring termination.

Note: none of the other systems guarantees termination.

Conclusions

- **Open issues:**
  - error recovery
  - rule programming support
  - rule termination

- **Future directions:**
  - support for application development
  - increasing the expressive power of rules
  - improved algorithms
  - distribution and parallelism