Active Databases

General principles of Conventional Database Systems

SQL Schema Definitions

DBMS

Query Results

SQL Queries and Updates

Conventional (Passive) Databases

- Data model, usually relational
- Transaction model
  - Passive update principle
  - Inventory control
    - Reordering of items when quantity in stock falls below threshold.
  - Travel waiting list
    - Book ticket as soon as right kind is available
  - Stock market
    - Buy/sell stocks when price below/above threshold

Active Databases

Recognize predefined situations
- In database
- Trigger predefined actions when situations occur

Actions are usually database updates

Conventional Databases

Passive DBMS

T1: 25 copies of Elmasri/Navathe sold

Less than 5 copies of S. in stock?
- Frequent polling => expensive
- Infrequent polling => might miss the right time to react
Problem is that DBMS does not know that application is polling.
### Active Databases

**General idea**
- ADBMS provides:
  - Regular DBMS primitives
  - definition of application-defined situations
  - triggering of application-defined reactions

**ADBMS schema situation/action rules**

**Results**
- Program invocations
- Manipulation operators
- Situation notification (use updates)

---

### Active Databases

**Further applications of ADBMS**
- Consistency enforcement:
  - Reaction to violations
    - e.g. ROLLBACK when constraint violated
  - Connection to time
    - e.g. check some constraint violations every midnight
- Computation of derived data
  - View materialization of derived data
    - e.g. incremental recomputation of view of sum of salaries per department,
      \( \text{salsum}(dno, \text{total}) \)
    - computed from \( \text{employee}(ssn, dno, salary) \)
    - or invalidation of materialized view when relevant update (e.g. salary) occurs
      \( \Rightarrow \) Rematerialize view when accessed next time if materialized view invalid

---

### Active Databases

**Semantics of ECA rules**
- Most common model presently
- Event Condition Action:
  WHEN event occurs (usually update of row in database table)
  IF condition holds (usually SQL query returning non-empty result)
  DO execute action (usually SQL update statements referencing the triggering event)

**Example, no condition part (EA-rule), SQL-99 (no Condition)**

```sql
CREATE TRIGGER EMPLOYEE_SALARY_MATERIALIZATION
AFTER UPDATE OF SALARY
ON EMPLOYEE
REFERENCING NEW ROW AS NROW,
OLD ROW AS OROW
FOR EACH ROW
BEGIN
UPDATE SALSUM S
SET TOTAL = TOTAL - OROW.SALARY
WHERE S.DNO = OROW.DNO
UPDATE SALSUM S
SET TOTAL = TOTAL + NROW.SALARY
WHERE S.DNO = NROW.DNO
END
```
Active Databases (ECA)

- **Event:**
  Update of a single database record
  Parameterised using pseudo tables with one row specified by REFERENCING clause.

- **Condition:**
  Query on database state,
  e.g. a database query
  - empty result => condition is FALSE
  - non-empty result => condition is TRUE

- **Action:**
  Database update statement(s)
  Call stored procedure
  - Unconditioned (EA) rules, as in example:
    ON ... DO
  - Condition/Action (CA) rules
    Not used in databases
    Difficult to identify situation when rule triggered

Active Databases

- Example of triggers (ECA) for maintaining constraints, Oracle

```
CREATE TRIGGER SALARY_SITUATION1
AFTER UPDATE OF SALARY ON EMPLOYEE
REFERENCING NEW ROW AS NROW, OLD ROW AS OROW
FOR EACH ROW
WHEN NROW.SALARY > (SELECT M.SALARY FROM EMPLOYEE M, DEPARTMENT D WHERE NROW.DNO = D.DNO AND D.MGRSSN = M.SSN)
BEGIN
UPDATE EMPLOYEE E
SET SALARY = OROW.SALARY*0.9
END
```

- NOTICE!
  SALARY_CONSTRAINT needed for managers:

```
CREATE TRIGGER SALARY_SITUATION2
AFTER UPDATE OF SALARY ON EMPLOYEE
REFERENCING NEW ROW AS NROW, OLD ROW AS OROW
FOR EACH ROW
WHEN NROW.SALARY < (SELECT E.SALARY FROM EMPLOYEE E, DEPARTMENT D WHERE E.DNO = D.DNO AND D.MGRSSN = NROW.SSN)
BEGIN
ROLLBACK
END
```

- NOTICE!
  SALARY_SITUATION3 needed for departments too in case employee promoted to manager!

- Possible catch-all solution: Integrity constraints.

Active Databases

- Advanced level SQL99 has assertions too:

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
(SELECT *
FROM EMPLOYEE E, EMPLOYEE M,
DEPARTMENT D
WHERE E.SALARY > M.SALARY AND
E.DNO = D.DNO AND
D.MGRSSN = M.SSN))
```

- Notice:
  Advanced integrity constraints may be allowed or may be implemented very inefficiently!
  Naive implementation would check above constraint after each update to any of the tables EMPLOYEE or DEPARTMENT.
  Integrity constraints cannot make different compensating actions depending on situation, as triggers can!
Active Databases

Cautions:

• Very powerful mechanism:
  Small statement => massive behaviour changes.
  Rope for programmer.
  Requires careful design
• Trace consequences of rule specification/changes.
  Make sure indefinite triggering or undesired cascading triggering cannot happen.
• Avoid using triggers unless really needed.
  Use queries or stored procedures instead if possible.

SUMMARY

• Active DBMSs embed situation-action rules in database
• Support many functionalities:
  E.g. Integrity control, access control, monitoring, derived data,
  change notification
• Some ADBMS functionality commercially available in SQL:99 as *triggers*: