Tore Risch

Dept. of Information Technology

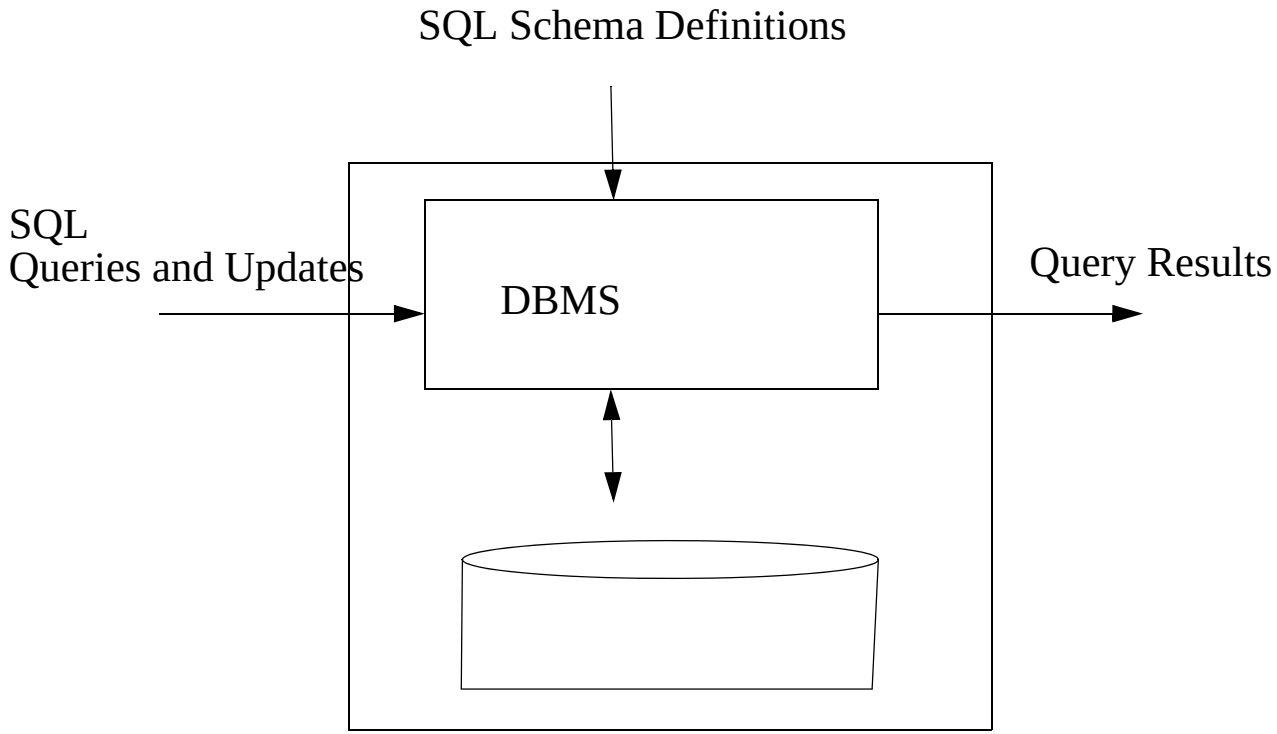
Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

General principles of Conventional Database Systems



Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Conventional (Passive) Databases

- Data model, usually relational
- Transaction model
  - *Passive* update principle - client controls DBMS updates

Example of real world problem not well suited for passive update principle:

- Inventory control

  - reordering 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

Tore Risch

Dept. of Information Technology

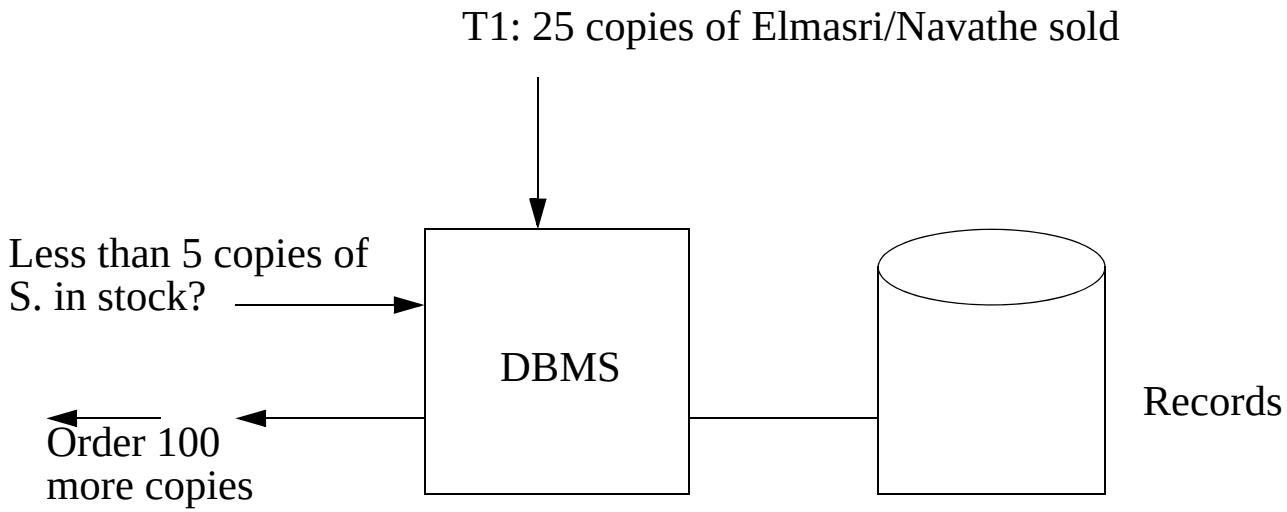
Uppsala University

Sweden

Tore.Risch@it.uu.se

Conventional Databases

Passive DBMS



- Periodical polling of database by application
  - Frequent polling => expensive
  - Infrequent polling => might miss the right time to react
- The polling has to be done for all items in stock and can be expensive.
- Problem is that DBMS does not know that application is polling.

Tore Risch

Dept. of Information Technology

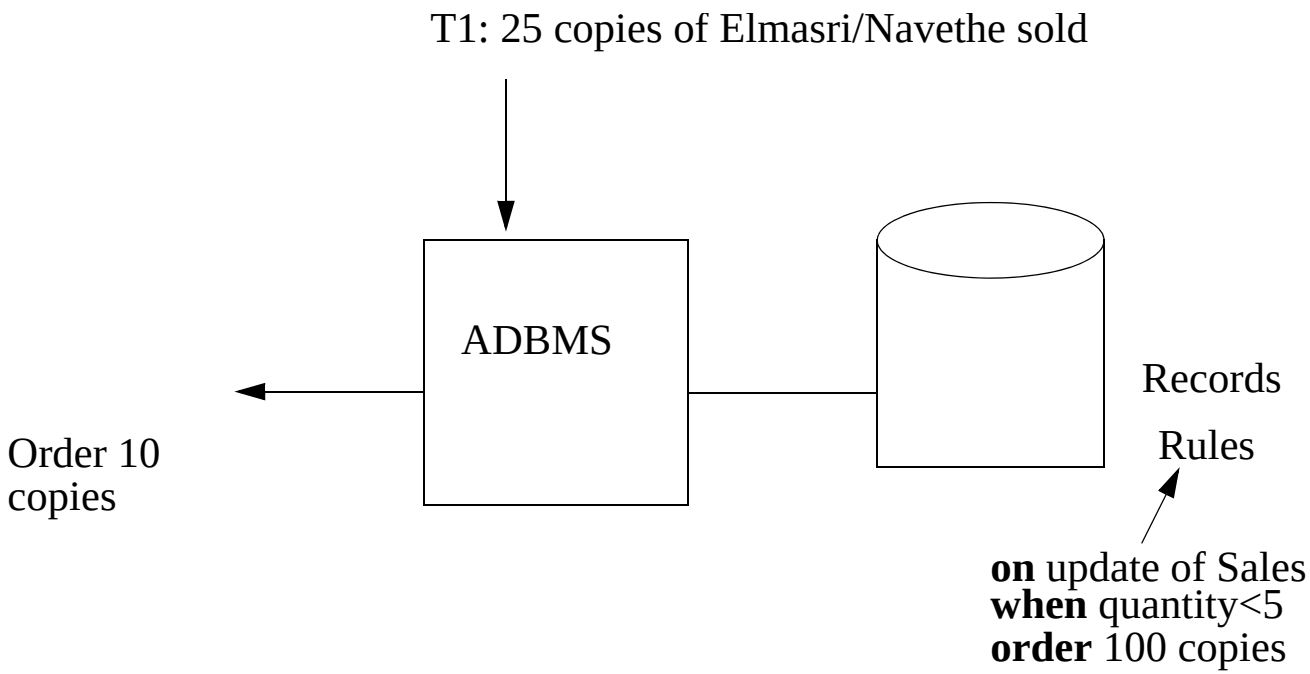
Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

Active DBMS



- Recognize predefined *situations* in database
- *Trigger* predefined actions when situations occur

  Actions are usually database updates, not calls to external programs to e.g. order items.

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

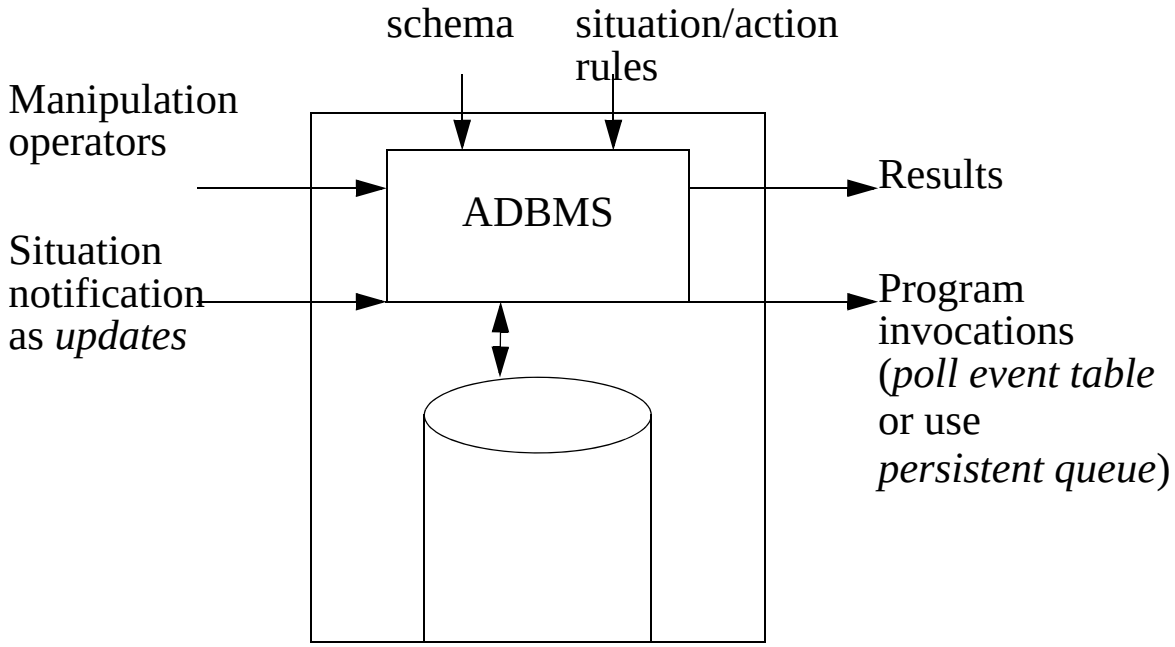
General idea

- ADBMS provides:

  Regular DBMS primitives

  + definition of application-defined `situations`

  + triggering of application-defined `reactions`



Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

Can be used for computation of derived data

View materialization of derived data

e.g. incremental recomputation of view of sum of salaries per department,

`salsum(dno,total),`

computed from `employee(ssn,dno,salary)`

or invalidation of materialized view when relevant update (e.g. salary) occurs

=> Rematerialize view when accessed next time if materialized view invalid

In Oracle materialized views can be specified declaratively as

```
create materialized view salsum
as select dno, sum(salary) as total
   from employee
```

However, this is not standard SQL! Triggers provide an alternative.

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

Semantics of ECA rules

- Most common model presently
- **E**vent **C**ondition **A**ction:

  WHEN event occurs

  Usually update of single row in database table)

  IF condition holds

  Usually SQL query joining the triggered row with database table.
  Condition is considered true if query returns non-empty result)

  DO execute action

  Usually SQL update statements or call to stored procedure
  referencing the updated row

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

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

```
EMPLOYEE(SSN, DNO, SALARY)
SALSUM(DNO, TOTAL) <= Materialized from EMPLOYEE
CREATE TRIGGER EMPLOYEE_SALARY_MATERIALIZATION
   AFTER UPDATE OF SALARY ON EMPLOYEE<--- Event
   REFERENCING NEW ROW AS NROW, OLD ROW AS OROW
   FOR EACH ROW          <--- per single updated row
   BEGIN                 <--- Action
      UPDATE SALSUM S
         SET TOTAL = TOTAL - OROW.SALARY FROM OROW
         WHERE S.DNO = OROW.DNO
      UPDATE SALSUM S
         SET TOTAL = TOTAL + NROW.SALARY FROM NROW
         WHERE S.DNO = NROW.DNO
   END
```

This does not cover changes to DNO! More triggers may be needed.

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases (ECA)

- *Event*:

  Update of a single database record

  Parameterized using pseudo tables with a single row (added, updated, or deleted) 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 both for user and DBMS.

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

- Example of triggers (ECA) for maintaining constraints, SQL:99:
  Department table with number and manager's SSN:
  DEPARTMENT(DNO, MGRSSN)

```
CREATE TRIGGER SALARY_SITUATION1
   AFTER UPDATE OF SALARY ON EMPLOYEE<--- Event
   REFERENCING NEW ROW AS NROW, OLD ROW AS OROW
   FOR EACH ROW          <--- C and A per updated row
   IF NROW.SALARY >(SELECT M.SALARY   <--- Condition
         FROM EMPLOYEE M,DEPARTMENT D,NROW
         WHERE
            NROW.DNO = D.DNO AND
            D.MGRSSN = M.SSN)
   THEN BEGIN                              <--- Action
      UPDATE EMPLOYEE E
      SET SALARY = OROW.SALARY*0.9 FROM OROW
   END
```

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

- 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
   IF NROW.SALARY < (SELECT E.SALARY
         FROM EMPLOYEE E,DEPARTMENT D, NROW
         WHEREE.DNO = D.DNO AND
            D.MGRSSN = NROW.SSN )
   THEN
      ROLLBACK
```

- NOTICE! SALARY_SITUATION3 needed for departments too in case employee promoted to manager!
- Possible catch-all solution: Integrity constraints, *assertions*.

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

- Advanced level SQL:99 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 assertions may not be supported by the DBMS or may be implemented very inefficiently! Check manual for when they are efficient.

Naive implementation would check above constraint after each update to any of the tables EMPLOYEE or DEPARTMENT, which is very inefficient (does not scale).

Assertions cannot make different compensating actions depending on situation, as triggers can!

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

Cautions:

- Very powerful mechanism:

  Small statement => massive behavior 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*, *view materialization* statements, *referential integrity constraints*, or *stored procedures* instead if possible.

Tore Risch

Dept. of Information Technology

Uppsala University

Sweden

Tore.Risch@it.uu.se

Active Databases

SUMMARY

- Active DBMSs embed situation-action rules in database
- Support many functionalities:

  E.g. Integrity control, derived data, change notification
- ADBMS functionality commercially available in SQL:99 as *triggers*: