Applications of Active Rules

- **Internal** to the database:
  - Integrity constraint maintenance
  - Support of data derivation
    (including data replication).

- **Extended** functionalities:
  - Workflow management systems
  - Version managers
  - Event tracking and logging
  - Security administration

- **Business Rules**:
  - Trading rules for the bond market
  - Warehouse and inventory management
  - Energy management rules
Internal and Extended Rules

- Perform classical DBMS functions
- Can be approached with structured approaches and techniques
- Can be automatically or semi-automatically generated
- Can be declaratively specified
Declarative Design of Active Rules for Integrity and View Maintenance

• Internal applications of active databases are:
  – Static
  – Declarative
  – High-level, easy to understand

• Approach
  – User specifies application at declarative (high) level
  – System derives low-level rules that implement it (automatically or semi-automatically)
Framework

- Rules should be programmed by DBA
- Rule programming should be assisted by rule design tools
- Rule derivation can be:
  - Completely automatic  
    (for few well-defined problems)
  - Partially automatic  
    (interactive system)
Integrity Constraint Maintenance

- Constraints are **static** conditions
  - Every employee’s department exists
  - Every employee’s salary is between 30 and 100

- Rules monitor **dynamic** database changes to enforce constraints
  - **when** change to employees or departments
    **if** an employee’s department doesn’t exist
    **then** fix the constraint
  - **when** change to employee salaries
    **if** a salary is not between 30 and 100
    **then** fix the constraint

- Generalizing:
  - **when** *potentially invalidating operations*
    **if** constraint violated
    **then** fix it
  - Constraint consistency points =
    Rule processing points
Integrity-Preserving Rules

- Constraint: condition $C$

- Rules(s):
  
  **when** operations that could make $C$ become false
  
  **if** $C$ is false
  
  **then** make $C$ true
  
  or abort transaction

- Example:

  $C =$ every employee’s department exists
  
  Operations = insert into emp,
  
  delete from dept,
  
  update to emp.deptno,
  
  update to dept.dno

- Condition:

  - There is some employee violating $C$ (due to above ops)

- Action: make $C$ true

  - Rollback insertion of emp
  
  - Rollback deletion of dept
  
  - Put emp into a dummy dept
Example: Referential Integrity

- Constraint:
  \[
  \text{EXISTS (SELECT * FROM Dept}
  \text{WHERE Dno = Emp.Deptno)}
  \]

- Denial form:
  \[
  \text{NOT EXISTS (SELECT * FROM Dept}
  \text{WHERE Dno = Emp.Deptno)}
  \]

- Abort Rules

  CREATE RULE DeptEmp1 ON Emp
  WHEN INSERTED, UPDATED(Deptno)
  IF \[
  \text{EXISTS (SELECT * FROM Emp}
  \text{WHERE NOT EXISTS}
  \text{(SELECT * FROM Dept}
  \text{WHERE Dno = Emp.DeptNo))}
  \] THEN ROLLBACK

  CREATE RULE DeptEmp2 ON Dept
  WHEN DELETED, UPDATED(Dno)
  IF \[
  \text{EXISTS (SELECT * FROM Emp}
  \text{WHERE NOT EXISTS}
  \text{(SELECT * FROM Dept}
  \text{WHERE Dno = Emp.DeptNo))}
  \] THEN ROLLBACK
Example: Repair Rules for EMP

CREATE RULE DeptEmp1 ON Emp
WHEN INSERTED
IF EXISTS (SELECT * FROM INSERTED
WHERE NOT EXISTS
(SELECT * FROM Dept
WHERE Dno ≠ Emp.DeptNo))
THEN UPDATE Emp
SET DeptNo = NULL
WHERE EmpNo IN
(SELECT EmpNo FROM INSERTED)
AND NOT EXISTS
(SELECT * FROM Dept
WHERE Dno = Emp.DeptNo))
Example: Repair Rules for EMP (2)

CREATE RULE DeptEmp2 ON Emp
WHEN UPDATED(Deptno)
IF EXISTS (SELECT * FROM NEW-UPDATED
WHERE NOT EXISTS
   (SELECT * FROM Dept
    WHERE Dno = Emp.DeptNo))
THEN UPDATE Emp
   SET DeptNo = 99
WHERE EmpNo IN
   (SELECT EmpNo FROM NEW-UPDATED)
AND NOT EXISTS
   (SELECT * FROM Dept
    WHERE Dno = Emp.DeptNo))
Example: Repair Rules for DEPT

Repair rules on table Dept
CREATE RULE DeptEmp3 ON Dept
WHEN DELETED
IF EXISTS (SELECT * FROM Emp WHERE EXISTS
(SELECT * FROM DELETED
  WHERE Dno = Emp.DeptNo))
THEN DELETE FROM Emp
  WHERE EXISTS
    (SELECT * FROM DELETED
     WHERE Dno = Emp.Deptno)

CREATE RULE DeptEmp4 ON Dept
WHEN UPDATED(Dno)
IF EXISTS (SELECT * FROM Emp WHERE EXISTS
(SELECT * FROM OLD-UPDATED
  WHERE Dno = Emp.Deptno))
THEN DELETE FROM Emp
  WHERE EXISTS
    (SELECT * FROM OLD-UPDATED
     WHERE Dno = Emp.DeptNo)
View Maintenance

- Logical tables derived from base tables
  - Portion of database specified by retrieval query
  - Used to provide different abstraction levels (or: *external schemas*)

- Referenced in retrieval queries

- *Virtual* views
  - Not physically stored
  - Implemented by query modification

- *Materialized* views
  - Physically stored
  - Kept consistent with base tables
Virtual Views

- Views define derived data by static database queries

  Table **high-paid** =
  All employees with high salaries

- Virtual views are not stored in the database

- Rules **dynamically** detect queries on virtual views and transform into queries on base tables

  when retrieve from high-paid
  then retrieve from emp
  where sal > X
Materialized Views

- View: $V = \text{query } Q$
- Rules(s): \textbf{when} operations that can change the result of $Q$
  \textbf{then} modify $V$
- How to generate rule(s) from view?
- Generate triggering operations by analyzing $Q$

\[
V = \text{all employees with high salaries}
\]

\[
\text{Ops = insert into emp, delete from emp, update emp.sal}
\]

- Generate action to modify $V$
  - Evaluate query $Q$, set $V = \text{result}$
  - Evaluate $Q$ using changed values, update $V$
  - Determine which by analyzing $Q$
Materialized Views and Rules

- SQL `select` expressions

```sql
define view V as
select Cols from Tables where Predicate
```

- Materialized initially, stored in database

- “Refreshed” at rule processing points

  Changes to base tables →
  Production rules modify view
View-Maintaining Rules

- Recomputation approach (easy but bad)

  \textbf{when} changes to base tables
  \textbf{then} recompute view

- Incremental approach (good but hard)

  \textbf{when} changes to base tables
  \textbf{then} change view

- Incremental rules is complicated for:
  
  - Views with duplicates
  - Certain base table operations
Example

- Relational view selecting departments with one employee who earns more than 50,000

```
DEFINE VIEW HighPaidDept AS
  (SELECT DISTINCT Dept.Name
   FROM Dept, Emp
   WHERE Dept.Dno = Emp.Deptno
   AND Emp.Sal > 50K)
```

- Critical events
  1. insertions into `Emp`
  2. insertions into `Dept`
  3. deletions from `Emp`
  4. deletions from `Dept`
  5. updates to `Emp.Deptno`
  6. updates to `Emp.Sal`
  7. updates to `Dept.Dno`
Refresh Rules written in Starburst

Refresh rules

CREATE RULE RefreshHighPaidDept1 ON Dept
WHEN INSERTED, DELETED,
    UPDATED(Deptno), UPDATED(Sal)
THEN DELETE * FROM HighPaidDept;
INSERT INTO HighPaidDept:
    (SELECT DISTINCT Dept.Name
     FROM Dept, Emp
     WHERE Dept.Dno = Emp.Deptno
     AND Emp.Sal > 50K)

CREATE RULE RefreshHighPaidDept2 ON Emp
WHEN INSERTED, DELETED, UPDATED(Dno)
THEN DELETE * FROM HighPaidDept;
INSERT INTO HighPaidDept:
    (SELECT DISTINCT Dept.Name
     FROM Dept, Emp
     WHERE Dept.Dno = Emp.Deptno
     AND Emp.Sal > 50K)
Incremental Rule for Insert on Dept

Incremental refresh rule
CREATE RULE IncrRefreshHighPaidDept1 ON Dept
WHEN INSERTED
THEN INSERT INTO HighPaidDept:
   (SELECT DISTINCT Dept.Name
    FROM INSERTED, Emp
    WHERE INSERTED.Dno = Emp.Deptno
    AND Emp.Sal > 50K)
Replication

- A special case of data derivation (identical copies).
- Main application: distributed systems (copies on different servers).
- Typical approach: asynchronous.
  - **Capture Step**: Active rules react to changes on one copy and collect changes into deltas.
  - **Apply step**: Deltas are propagated to other copies at the appropriate time.
- Alternatives:
  - Primary-Secondary
  - Symmetric
Active Rules for Replication

Capture rules CREATE RULE Capture1 ON Primary
WHEN INSERTED
THEN INSERT INTO PosDelta
   (SELECT * FROM INSERTED)

CREATE RULE Capture2 ON Primary
WHEN DELETED
THEN INSERT INTO NegDelta
   (SELECT * FROM DELETED)

CREATE RULE Capture3 ON Primary
WHEN UPDATED
THEN INSERT INTO PosDelta
   (SELECT * FROM NEW-UPDATED);
   INSERT INTO NegDelta
   (SELECT * FROM OLD-UPDATED)
Workflow Management

- A new paradigm for organizing the working activities within enterprise.

- Intrinsically reactive: workflow managers monitor events and perform the required event management activities.

- Events are:
  - **Internal**: generated from within the workflow manager while workflows are progressing.
  - **External**: representing the interaction of the workflow manager with the external world.

- The most significant application of rules: expressing exceptions to the normal flow.
Examples of Active Rules for Workflow Management

define trigger WF1 for Agent
   events modify(Agent.Availability)
   condition Agent(A), occurred(modify(Agent.Availability),A),
      A.Availability=FALSE, task(T), T.Responsible=A,
      T.Type='Urgent', Agent(B), A.Substitute=B,
      B.Availability=TRUE
   actions modify(Task.Responsible, T, B)
end;

define trigger WF2 for Accident
   events create(Accident)
   condition Accident(A), occurred(create, A),
      Booking(B), B.Car = A.DamagedCar,
   actions create(Warning,[B.Number,B.Agent],X)
end;
Business Rules

- Performing a part of the application-specific business.

- Examples:
  - Stock and bond trading in financial applications.
  - Airway assignment to flights in air traffic control systems.
  - Order management in inventory control systems.

- Key design principle:
  **knowledge independence**.
  
  - Factoring knowledge out of the applications.
  - Rules automatically shared by all applications.
  - Rules logically part of the database schema (designed by “DBA”).
  - Knowledge evolution feasible and controllable (changing rules without changing applications).
Energy Management System

- The ENEL Energy Management System uses a "radial topology" network (← forest), each "user" connected to a single "distributor" through a network of intermediate "nodes".

- The purpose of the network is to transfer the exact power from distributors to users through nodes and (directed) branches connecting pairs of nodes.

- A transaction changes the user’s profile, then the system finds the appropriate layout and power supply.
Active Rules

R1: If a new user requires power, connect it to the closest node
R2: If a user or a node requires less power, change the power of the user or node and propagate the change to its input branch
R3: If a branch requires less power, change the power of the branch and propagate the change to its input node
R4: If the power required from a distributor is decreased, change its output power accordingly
R5: If a user or node requires more power, change the power of the user or node and propagate the change to its input branch
R6: If a branch requires more power, change the power of the branch and propagate the change to its input node
R7: If the power required from a distributor is increased, change its output power accordingly
R8: If a distributor’s power exceeds its maximum, rollback the entire transaction
R9: If a branch’s power is changed, change the power of some of its wires accordingly
R10: If a wire’s power is above its threshold, change wire type
R11: If there’s no suitable wire type, add another wire to the branch
R12: If a wire is not included into a tube, add a tube around it
R13: If a tube is too small to fit all its wires, change it into a larger tube
R14: If a wire inside a tube is high voltage and the tube is not protected, change it into a protected tube
R15: If there’s no suitable tube, split the branch into two branches