Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Semantic Data Control
  - View Management
  - Data Security
  - Semantic Integrity Control
- Distributed Query Processing
- Distributed Transaction Management
- Distributed Database Operating Systems
- Parallel Database Systems
- Distributed Object DBMS
- Database Interoperability
- Current Issues
Semantic Data Control

- **Involves:**
  - View management
  - Security control
  - Integrity control

- **Objective:**
  - Insure that *authorized* users perform *correct* operations on the database, contributing to the maintenance of the database integrity.
View Management

View – virtual relation

- generated from base relation(s) by a query
- not stored as base relations

Example:

```sql
CREATE VIEW SYSAN (ENO, ENAME)
AS SELECT ENO, ENAME
FROM EMP
WHERE TITLE="Syst. Anal."
```

<table>
<thead>
<tr>
<th>EMP</th>
<th>ENO</th>
<th>ENAME</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E1</td>
<td>J. Doe</td>
<td>Elect. Eng</td>
</tr>
<tr>
<td></td>
<td>E2</td>
<td>M. Smith</td>
<td>Syst. Anal.</td>
</tr>
<tr>
<td></td>
<td>E4</td>
<td>J. Miller</td>
<td>Programmer</td>
</tr>
<tr>
<td></td>
<td>E5</td>
<td>B. Casey</td>
<td>Syst. Anal.</td>
</tr>
<tr>
<td></td>
<td>E8</td>
<td>J. Jones</td>
<td>Syst. Anal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYSAN</th>
<th>ENO</th>
<th>ENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E2</td>
<td>M. Smith</td>
</tr>
<tr>
<td></td>
<td>E5</td>
<td>B. Casey</td>
</tr>
<tr>
<td></td>
<td>E8</td>
<td>J. Jones</td>
</tr>
</tbody>
</table>
View Management

Views can be manipulated as base relations

Example:

SELECT ENAME, PNO, RESP
FROM SYSAN, ASG
WHERE SYSAN.ENO = ASG.ENO
Query Modification

queries expressed on views

queries expressed on base relations

Example:

\[
\text{SELECT ENAME, PNO, RESP} \\
\text{FROM SYSAN, ASG} \\
\text{WHERE SYSN.ENO = ASG.ENO}
\]

\[
\text{SELECT ENAME, PNO, RESP} \\
\text{FROM EMP, ASG} \\
\text{WHERE EMP.ENO = ASG.ENO} \text{ AND TITLE = “Syst. Anal.”}
\]

<table>
<thead>
<tr>
<th>ENAME</th>
<th>PNO</th>
<th>RESP</th>
</tr>
</thead>
<tbody>
<tr>
<td>M.Smith</td>
<td>P1</td>
<td>Analyst</td>
</tr>
<tr>
<td>M.Smith</td>
<td>P2</td>
<td>Analyst</td>
</tr>
<tr>
<td>B.Casey</td>
<td>P3</td>
<td>Manager</td>
</tr>
<tr>
<td>J.Jones</td>
<td>P4</td>
<td>Manager</td>
</tr>
</tbody>
</table>
View Management

- To restrict access

```
CREATE VIEW ESAME
AS
SELECT *
FROM EMP E1, EMP E2
WHERE E1.TITLE = E2.TITLE
AND E1.ENO = USER
```

- Query

```
SELECT *
FROM ESAME
```

<table>
<thead>
<tr>
<th>ENO</th>
<th>ENAME</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>J. Doe</td>
<td>Elect. Eng</td>
</tr>
<tr>
<td>E2</td>
<td>L. Chu</td>
<td>Elect. Eng</td>
</tr>
</tbody>
</table>
View Updates

- Updatable

```
CREATE VIEW SYSAN(ENO, ENAME)
AS SELECT ENO, ENAME
FROM EMP
WHERE TITLE = "Syst. Anal."
```

- Non-updatable

```
CREATE VIEW EG(ENAME, RESP)
AS SELECT ENAME, RESP
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO
```
View Management in DDBMS

- Views might be derived from fragments.
- View definition storage should be treated as database storage.
- Query modification results in a distributed query.
- View evaluations might be costly if base relations are distributed.
  - use snapshots
    - Static views - do not reflect the updates to the base relations
    - managed as temporary relations - only access path is sequential scan
    - bad selectivity - snapshots behave as pre-calculated answers
    - periodic recalculation
Data Security

- **Data protection**
  - prevent the physical content of data to be understood by unauthorized users
  - encryption/decryption
    - Data Encryption Standard
    - Public-key encryption

- **Authorization control**
  - only authorized users perform operations they are allowed to on the database
    - identification of subjects and objects
    - authentication of subjects
    - granting of rights (authorization matrix)
Semantic Integrity Control

- Maintain database consistency by enforcing a set of constraints defined on the database.

- Structural constraints
  - basic semantic properties inherent to a data model
    e.g., unique key constraint in relational model

- Behavioral constraints
  - regulate application behavior
    e.g., dependencies in the relational model

- Two components
  - Integrity constraint specification
  - Integrity constraint enforcement
Semantic Integrity Control

- **Procedural**
  - control embedded in each application program

- **Declarative**
  - assertions in predicate calculus
  - easy to define constraints
  - definition of database consistency clear
  - inefficient to check assertions for each update
    - limit the search space
    - decrease the number of data accesses/assertion
    - preventive strategies
    - checking at compile time
Constraint Specification Language

Predefined constraints

specify the more common constraints of the relational model

- **Not-null attribute**
  
  ENO NOT NULL IN EMP

- **Unique key**

  (ENO, PNO) UNIQUE IN ASG

- **Foreign key**

  A key in a relation \( R \) is a foreign key if it is a primary key of another relation \( S \) and the existence of any of its values in \( R \) is dependent upon the existence of the same value in \( S \)

  PNO IN ASG REFERENCES PNO IN PROJ

- **Functional dependency**

  ENO IN EMP DETERMINES ENAME
Precompiled constraints

Express preconditions that must be satisfied by all tuples in a relation for a given update type

(INSERT, DELETE, MODIFY)

NEW - ranges over new tuples to be inserted
OLD - ranges over old tuples to be deleted

General Form

CHECK ON <relation> [WHEN <update type>] <qualification>
Constraint Specification Language

Precompiled constraints

- Domain constraint

  \[
  \text{CHECK ON PROJ (BUDGET} \geq 500000 \text{ AND BUDGET} \leq 1000000)\]

- Domain constraint on deletion

  \[
  \text{CHECK ON PROJ WHEN DELETE (BUDGET} = 0)\]

- Transition constraint

  \[
  \text{CHECK ON PROJ (NEW.BUDGET} > \text{OLD.BUDGET AND NEW.PNO} = \text{OLD.PNO)}\]
General constraints

Constraints that must always be true. Formulae of tuple relational calculus where all variables are quantified.

General Form

\[
\text{CHECK ON } <\text{variable}>:<\text{relation}>,(<\text{qualification}>)
\]

- Functional dependency

\[
\text{CHECK ON } \text{e1:EMP, e2:EMP} \\
(e1.\text{ENAME} = e2.\text{ENAME} \text{ IF } e1.\text{ENO} = e2.\text{ENO})
\]

- Constraint with aggregate function

\[
\text{CHECK ON } \text{g:ASG, j:PROJ} \\
(\text{SUM}(g.\text{DUR} \text{ WHERE } g.\text{PNO} = j.\text{PNO}) < 100 \text{ IF } j.\text{PNAME} = \text{"CAD/CAM"})
\]
Integrity Enforcement

Two methods

- **Detection**
  
  Execute update $u: D \rightarrow D_u$
  
  If $D_u$ is inconsistent then
  
  compensate $D_u \rightarrow D_u'$
  
  else
  
  undo $D_u \rightarrow D$

- **Preventive**
  
  Execute $u: D \rightarrow D_u$ only if $D_u$ will be consistent
  
  ➔ Determine valid programs
  
  ➔ Determine valid states
Query Modification

- Preventive
- Add the assertion qualification to the update query
- Only applicable to tuple calculus formulae with universally quantified variables

```
UPDATE PROJ
SET BUDGET = BUDGET*1.1
WHERE PNAME = “CAD/CAM”
```

```
UPDATE PROJ
SET BUDGET = BUDGET*1.1
WHERE PNAME = “CAD/CAM”
AND NEW.BUDGET ≥ 500000
AND NEW.BUDGET ≤ 1000000
```
Compiled Assertions

Triple \((R,T,C)\) where
- \(R\) relation
- \(T\) update type (insert, delete, modify)
- \(C\) assertion on differential relations

Example: Foreign key assertion

\[\forall g \in \text{ASG}, \exists j \in \text{PROJ} : g\text{.PNO} = j\text{.PNO}\]

Compiled assertions:

\((\text{ASG}, \text{INSERT}, C1), (\text{PROJ}, \text{DELETE}, C2), (\text{PROJ}, \text{MODIFY}, C3)\)

where

- \(C1: \forall \text{NEW} \in \text{ASG+}, \exists j \in \text{PROJ} : \text{NEW}\text{.PNO} = j\text{.PNO}\)
- \(C2: \forall g \in \text{ASG}, \forall \text{OLD} \in \text{PROJ-} : g\text{.PNO} \neq \text{OLD}\text{.PNO}\)
- \(C3: \forall g \in \text{ASG}, \forall \text{OLD} \in \text{PROJ-}, \exists \text{NEW} \in \text{PROJ+} : g\text{.PNO} \neq \text{OLD}\text{.PNO} \text{ OR } \text{OLD}\text{.PNO} = \text{NEW}\text{.PNO}\)
Differential Relations

Given relation $R$ and update $u$

- $R^+$ contains tuples inserted by $u$
- $R^-$ contains tuples deleted by $u$

Type of $u$

- insert: $R^-$ empty
- delete: $R^+$ empty
- modify: $R^+ \cup (R - R^-)$
Algorithm

**Input:** Relation $R$, update $u$, compiled assertion $C_i$

**Step 1:** Generate differential relations $R+$ and $R–$

**Step 2:** Retrieve the tuples of $R+$ and $R–$ which do not satisfy $C_i$

**Step 3:** If retrieval is not successful, then the assertion is valid.

**Example:**

$u$ is delete on J. Enforcing (J, DELETE, C2) :

- retrieve all tuples of J-
- into RESULT
- where not(C2)

If RESULT = φ, the assertion is verified.
Distributed Integrity Control

Problems:

- Definition of constraints
  - consideration for fragments

- Where to store
  - replication
  - non-replicated: fragments

- Enforcement
  - minimize costs
Types of Distributed Assertions

- **Individual assertions**
  - single relation, single variable
  - domain constraint

- **Set oriented assertions**
  - single relation, multi-variable
    - functional dependency
  - multi-relation, multi-variable
    - foreign key

- **Assertions involving aggregates**
Distributed Integrity Control

- Assertion Definition
  - similar to the centralized techniques
  - transform the assertions to compiled assertions

- Assertion Storage
  - Individual assertions
    - one relation, only fragments
    - at each fragment site, check for compatibility
    - if compatible, store; otherwise reject
    - if all the sites reject, globally reject
  - Set-oriented assertions
    - involves joins (between fragments or relations)
    - maybe necessary to perform joins to check for compatibility
    - store if compatible
Distributed Integrity Control

- **Assertion Enforcement**
  - Where do you enforce each assertion?
    - type of assertion
    - type of update and where update is issued
  - **Individual Assertions**
    - update = insert
      - ✔ enforce at the site where the update is issued
    - update = qualified
      - ✔ send the assertions to all the sites involved
      - ✔ execute the qualification to obtain R+ and R-
      - ✔ each site enforce its own assertion
  - **Set-oriented Assertions**
    - single relation
      - ✔ similar to individual assertions with qualified updates
    - multi-relation
      - ✔ move data between sites to perform joins; then send the result to the query master site