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
- Open Systems and Interoperability
- Parallel Database Systems
- Distributed Object Management
- Concluding Remarks

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 :

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

E

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

SYSAN

ENO	ENAME
E2	M.Smith
E5	B.Casey
E8	J.Jones

View Management

Views can be manipulated as base relations

Example :

```
SELECT   ENAME, JNO, RESP
FROM     SYSAN, G
WHERE    SYSAN.ENO = G.ENO
```

Query Modification

queries expressed on views

⇓

queries expresed on base relations

Example :

```
SELECT ENAME, JNO, RESP
FROM   SYSAN, G
WHERE  SYSN.ENO = G.ENO
       ⇓
SELECT ENAME,JNO,RESP
FROM   E, G
WHERE  E.ENO = G.ENO
AND    TITLE = "Syst. Anal."
```

ENAME	PNO	RESP
M.Smith	P1	Analyst
M.Smith	P2	Analyst
B.Casey	P3	Manager
J.Jones	P4	Manager

View Management

- To restrict access

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

- Query

```
SELECT  *
FROM    ESAME
```

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	L. Chu	Elect. Eng

View Updates

- Updatable

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

- Non-updatable

```
CREATE VIEW   EG(ENAME,RESP)
AS     SELECT ENAME,RESP
       FROM   E, G
       WHERE  E.ENO=G.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** E

- Unique key

  (ENO, JNO) **UNIQUE IN** G

- 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*

  JNO **IN** G **REFERENCES** JNO **IN** J

- Functional dependency

  ENO **IN** E **DETERMINES** ENAME

Constraint Specification Language

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

  CHECK ON J(BUDGET≥500000 **AND** BUDGET≤1000000)

- Domain constraint on deletion

  CHECK ON J **WHEN DELETE** (BUDGET = 0)

- Transition constraint

  CHECK ON J (**NEW**.BUDGET > **OLD**.BUDGET **AND**
  NEW.JNO = **OLD**.JNO)

Constraint Specification Language

General constraints

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

General Form

CHECK ON <variable>:<relation>,(<qualification>)

- Functional dependency

  CHECK ON e1:E, e2:E

  (e1.ENAME = e2.ENAME **IF** e1.ENO = e2.ENO)

- Constraint with aggregate function

  CHECK ON g:G, j:J

  (**SUM**(g.DUR **WHERE** g.JNO = j.JNO) < 100 **IF** j.JNAME = "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 J
SET    BUDGET = BUDGET*1.1
WHERE  JNAME ="CAD/CAM"
```

$\Downarrow$

```
UPDATE J
SET    BUDGET = BUDGET*1.1
WHERE  JNAME ="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

$\forall g \in G, \exists j \in J:\ g.JNO = j.JNO$

Compiled assertions:

(G, **INSERT**, C1), (J, **DELETE**, C2), (J, **MODIFY**, C3)

where

C1:$\forall$**NEW** $\in$ G+, $\exists$j $\in$ J: NEW.JNO = j.JNO

C2:$\forall$g $\in$ G, $\forall$**OLD** $\in$ J- : g.JNO $\neq$ **OLD**.JNO

C3:$\forall$g $\in$ G, $\forall$**OLD** $\in$ J-, $\exists$**NEW** $\in$ J+:g.JNO $\neq$ **OLD**.JNO
OR OLD.JNO = **NEW**.JNO

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 $R+ \cup (R - R-)$

Differential Relations

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