

Outline

- Introduction
- Background
- Distributed Database Design
- Database Integration
- Semantic Data Control
 - View Management
 - Data Security
 - Semantic Integrity Control
- Distributed Query Processing
- Multidatabase Query Processing
- Distributed Transaction Management
- Data Replication
- Parallel Database Systems
- Distributed Object DBMS
- Peer-to-Peer Data Management
- Web Data Management
- 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 :

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

EMP

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, PNO, RESP  
FROM SYSAN, ASG  
WHERE SYSAN.ENO = ASG.ENO
```


Query Modification

Queries expressed on views



Queries expressed on base relations

Example :

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



```
SELECT ENAME, PNO, RESP  
FROM EMP, ASG  
WHERE EMP.ENO = ASG.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 EMP E1, EMP 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 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 **materialized views**

Materialized View

- Origin: snapshot in the 1980's
 - Static copy of the view, avoid view derivation for each query
 - But periodic recomputing of the view may be expensive
- Actual version of a view
 - Stored as a database relation, possibly with indices
- Used much in practice
 - DDBMS: No need to access remote, base relations
 - Data warehouse: to speed up OLAP
 - Use aggregate (SUM, COUNT, etc.) and GROUP BY

Materialized View Maintenance

- Process of updating (refreshing) the view to reflect changes to base data
 - Resembles data replication but there are differences
 - View expressions typically more complex
 - Replication configurations more general
- View maintenance policy to specify:
 - When to refresh
 - How to refresh

When to Refresh a View

- Immediate mode
 - As part of the updating transaction, e.g. through 2PC
 - View always consistent with base data and fast queries
 - But increased transaction time to update base data
- Deferred mode (preferred in practice)
 - Through separate refresh transactions
 - No penalty on the updating transactions
 - Triggered at different times with different trade-offs
 - Lazily: just before evaluating a query on the view
 - Periodically: every hour, every day, etc.
 - Forcedly: after a number of predefined updates

How to Refresh a View

- Full computing from base data
 - Efficient if there has been many changes
- Incremental computing by applying only the changes to the view
 - Better if a small subset has been changed
 - Uses differential relations which reflect updated data only

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

Refreshing a view V is then done by computing

$$V^+ \cup (V - V^-)$$

computing V^+ and V^- may require accessing base data

Example

```
EG = SELECT DISTINCT ENAME, RESP  
FROM EMP, ASG  
WHERE EMP.ENO=ASG.ENO
```

```
EG+ = (SELECT DISTINCT ENAME, RESP  
FROM EMP, ASG+  
WHERE EMP.ENO=ASG+.ENO) UNION  
(SELECT DISTINCT ENAME, RESP  
FROM EMP+, ASG  
WHERE EMP+.ENO=ASG.ENO) UNION  
(SELECT DISTINCT ENAME, RESP  
FROM EMP+, ASG+  
WHERE EMP+.ENO=ASG+.ENO)
```


Techniques for Incremental View Maintenance

- Different techniques depending on:
 - View expressiveness
 - Non recursive views: SPJ with duplicate elimination, union and aggregation
 - Views with outerjoin
 - Recursive views
- Most frequent case is non recursive views
 - Problem: an individual tuple in the view may be derived from several base tuples
 - Example: tuple $\langle \text{M. Smith, Analyst} \rangle$ in EG corresponding to
 - ✓ $\langle \text{E2, M. Smith, ...} \rangle$ in EMP
 - ✓ $\langle \text{E2,P1,Analyst,24} \rangle$ and $\langle \text{E2,P2,Analyst,6} \rangle$ in ASG
 - Makes deletion difficult
 - Solution: Counting

Counting Algorithm

- Basic idea
 - Maintain a count of the number of derivations for each tuple in the view
 - Increment (resp. decrement) tuple counts based on insertions (resp. deletions)
 - A tuple in the view whose count is zero can be deleted
- Algorithm
 1. Compute V^+ and V^- using V , base relations and diff. relations
 2. Compute positive in V^+ and negative counts in V^-
 3. Compute $V^+ \cup (V - V^-)$, deleting each tuple in V with count=0
- Optimal: computes exactly the view tuples that are inserted or deleted

View Self-maintainability

- A view is self-maintainable if the base relations need not be accessed
 - Not the case for the Counting algorithm
- Self-maintainability depends on views' expressiveness
 - Most SPJ views are often self-maintainable wrt. deletion and modification, but not wrt. Insertion
 - Example: a view V is self-maintainable wrt to deletion in R if the key of R is included in V

Data Security

- Data protection
 - Prevents the physical content of data to be understood by unauthorized users
 - Uses encryption/decryption techniques (Public key)
- Access control
 - Only authorized users perform operations they are allowed to on database objects
 - Discretionary access control (DAC)
 - Long been provided by DBMS with authorization rules
 - Multilevel access control (MAC)
 - Increases security with security levels

Discretionary Access Control

- Main actors
 - Subjects (users, groups of users) who execute operations
 - Operations (in queries or application programs)
 - Objects, on which operations are performed
- Checking whether a subject may perform an op. on an object
 - Authorization= (subject, op. type, object def.)
 - Defined using GRANT OR REVOKE
 - Centralized: one single user class (admin.) may grant or revoke
 - Decentralized, with op. type GRANT
 - More flexible but recursive revoking process which needs the hierarchy of grants

Problem with DAC

- A malicious user can access unauthorized data through an authorized user
- Example
 - User A has authorized access to R and S
 - User B has authorized access to S only
 - B somehow manages to modify an application program used by A so it writes R data in S
 - Then B can read unauthorized data (in S) without violating authorization rules
- Solution: multilevel security based on the famous Bell and Lapuda model for OS security

Multilevel Access Control

- Different security levels (*clearances*)
 - *Top Secret > Secret > Confidential > Unclassified*
- Access controlled by 2 rules:
 - No read up
 - subject S is allowed to read an object of level L only if $level(S) \geq L$
 - Protect data from unauthorized disclosure, e.g. a subject with secret clearance cannot read top secret data
 - No write down:
 - subject S is allowed to write an object of level L only if $level(S) \leq L$
 - Protect data from unauthorized change, e.g. a subject with top secret clearance can only write top secret data but not secret data (which could then contain top secret data)

MAC in Relational DB

- A relation can be classified at different levels:
 - Relation: all tuples have the same clearance
 - Tuple: every tuple has a clearance
 - Attribute: every attribute has a clearance
- A classified relation is thus multilevel
 - Appears differently (with different data) to subjects with different clearances

Example

PROJ*: classified at attribute level

PNO	SL1	PNAME	SL2	BUDGET	SL3	LOC	SL4
P1	C	Instrumentation	C	150000	C	Montreal	C
P2	C	DB Develop.	C	135000	S	New York	S
P3	S	CAD/CAM	S	250000	S	New York	S

PROJ* as seen by a subject with confidential clearance

PNO	SL1	PNAME	SL2	BUDGET	SL3	LOC	SL4
P1	C	Instrumentation	C	150000	C	Montreal	C
P2	C	DB Develop.	C	Null	C	Null	C

Distributed Access Control

- Additional problems in a distributed environment
 - Remote user authentication
 - Typically using a directory service
 - ✓ Should be replicated at some sites for availability
 - Management of DAC rules
 - Problem if users' group can span multiple sites
 - ✓ Rules stored at some directory based on user groups location
 - ✓ Accessing rules may incur remote queries
 - Covert channels in MAC

Covert Channels

- Indirect means to access unauthorized data
- Example
 - Consider a simple DDB with 2 sites: C (confidential) and S (secret)
 - Following the “no write down” rule, an update from a subject with secret clearance can only be sent to S
 - Following the “no read up” rule, a read query from the same subject can be sent to both C and S
 - But the query may contain secret information (e.g. in a select predicate), so is a potential covert channel
- Solution: replicate part of the DB
 - So that a site at security level L contains all data that a subject at level L can access (e.g. S above would replicate the confidential data so it can entirely process secret queries)

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

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 PROJ (BUDGET ≥ 500000 AND BUDGET ≤ 1000000)

- Domain constraint on deletion

CHECK ON PROJ WHEN DELETE (BUDGET = 0)

- Transition constraint

**CHECK ON PROJ (NEW.BUDGET > OLD.BUDGET AND
NEW.PNO = OLD.PNO)**

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:EMP, e2:EMP

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

- Constraint with aggregate function

CHECK ON g:ASG, j:PROJ

(SUM(g.DUR WHERE g.PNO = j.PNO) < 100 IF

j.PNAME = "CAD/CAM")

Integrity Enforcement

Two methods

- Detection

Execute update $u: D \rightarrow D_u$

If D_u is inconsistent then

if possible: 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
```



```
UPDATE PROJ
SET BUDGET = BUDGET*1.1
WHERE PNAME = "CAD/CAM"
AND NEW.BUDGET ≥ 500000
```

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

$$C1: \forall \text{NEW} \in \text{ASG}^+ \quad \exists j \in \text{PROJ} : \text{NEW.PNO} = j.\text{PNO}$$

$$C2: \forall g \in \text{ASG}, \forall \text{OLD} \in \text{PROJ}^- : g.\text{PNO} \neq \text{OLD.PNO}$$

$$C3: \forall g \in \text{ASG}, \forall \text{OLD} \in \text{PROJ}^- \quad \exists \text{NEW} \in \text{PROJ}^+ : \\ g.\text{PNO} \neq \text{OLD.PNO} \text{ OR } \text{OLD.PNO} = \text{NEW.PNO}$$

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

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