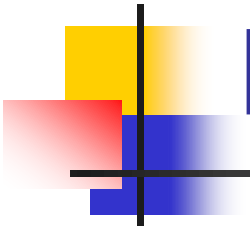
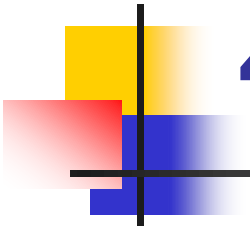


Distributed Data Processing

- Introduction
- Distributed DBMS Architecture
- Distributed DB Design
- **Semantic Data Control**
- Query Processing
- Transaction Management

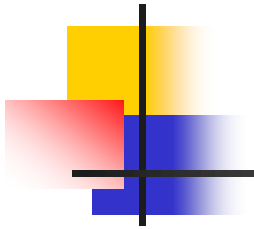


4. Semantic Data Control

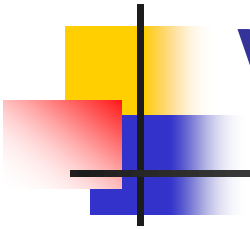
- View Management
- Data Security
- Semantic Integrity Control

Semantic Data Control

- Objective
  - Insure that **authorized** users perform **correct** operations on the database, contributing to the ***maintenance of the database integrity***.
- Involves
  - View management
  - Security control
  - Integrity control
- In the relational framework
  - They can be achieved in a uniform fashion – defined as **rules** that the system automatically enforces.
  - Rules are defined by DBA, and stored in the directory.



4.1 View Management



View Management

- View -- virtual relation
  - generated from base relation(s) by a query
  - not stored as base relations
- Used in
  - External schema
    - Particular view for different users
  - Data security
    - Hiding some data by selecting a subset of the database

Example of View

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

EMP

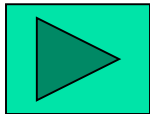
ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

SYSAN

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

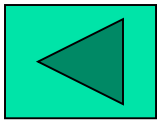
4.1.1 Views in Centralized DBMSs

- Definition
  - A view is a relation derived from base relations as **the result of a relational query**
- Design
  - The view definition is stored in the directory (the result is not produced in fact)
  - The view can be manipulated as a base relation
- Materialization
  - **Query modification** – mapping a query *expressed on views* into a query *expressed on base relations*.
  - It can be done at compile time
- Access control
  - Refining the access controls to include subsets of objects.



Example of
Views manipulated as base relations

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



Example of Query modification

Query expressed on view

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

Query expressed on base relation

SELECT **ENAME,PNO,RESP**
FROM ***EMP*, ASG**
WHERE ***EMP*.ENO = ASG.ENO**
AND TITLE = "Syst. Anal."

Definition of view

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

Result of Query

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.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

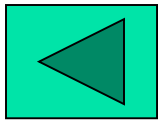
PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

SYSAN

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

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



Example of Access Control

To restrict access

e.g. restricts the access by any user to those employees having the same title

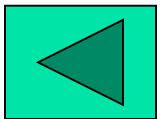
```
CREATE VIEW     ESAME
AS      SELECT *
        FROM EMP E1, EMP E2
        WHERE E1.TITLE = E2.TITLE
            AND E1.ENO = USER
```

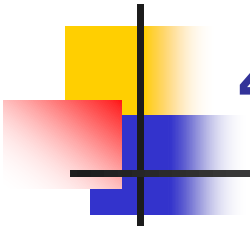
Query

e.g. issued by the user J. Doe

```
SELECT *
FROM ESAME
```

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





4.1.2 Updates through Views

- Updates through views can be handled automatically only if they can be propagated correctly to the base relations
  - For views derived from a single relation by selection and projection
  - For views derived by join if they include the keys of the base relations
- Updatable vs. Non-updatable

Updatable vs. Non-updatable

- Updatable

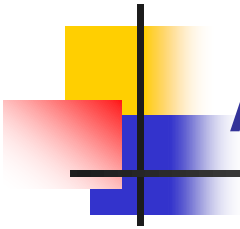
```
CREATE VIEW     SYSAN(ENO,ENAME)
AS    SELECT ENO,ENAME
      FROM EMP
      WHERE TITLE="Syst. Anal."
```

- Non-updatable

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

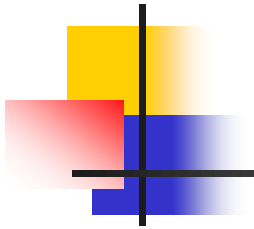
4.1.3 Views in Distributed DBMSs

- Definition
  - Similar to that in a centralized system
  - Might be derived from fragments
- Design
  - View definition storage should be treated as database storage
  - In any case, the information associating a view name to its definition site should be duplicated
- Materialization
  - Query modification
  - Results in a distributed query
- Fragmentation & view
  - A unified mechanism for managing views and fragments is feasible
    - Views can be defined with rules similar to fragment definition rules
    - Replicated data can be handled in the same way
  - DBA may increase locality of reference by making views in one-to-one correspondence with fragments

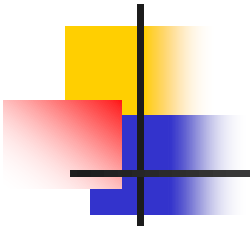


Alternatives of materialization

- Query modification
  - By view derivation - merging the view qualification with the query qualification
  - View derivation might be costly to evaluate if base relations are distributed
- Another solution – snapshot
  - To avoid view derivation by maintaining actual versions of the views
    - A snapshot represents a particular state of the database
    - **Static** views - do not reflect the updates to the base relations
    - managed as temporary relations - only access method is sequential **scan**
    - More adequate for queries that have bad selectivity and scan the entire snapshot - snapshots behave as **pre-calculated answers**
    - periodic recalculation



4.2 Data Security

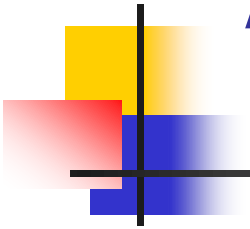


Data Security

To protect data against unauthorized access

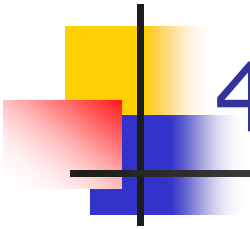
- Data protection
  - prevent the physical content of data to be understood by unauthorized users
  - encryption/decryption
    - Data Encryption Standard
    - Public-key encryption
- Authorization control
  - only authorized users perform operations they are allowed to on the database
    - identification of subjects and objects
    - authentication of subjects
    - granting of rights (authorization matrix)

Authorization control in DB vs. in FS



- Authorization must be refined so that different users have different rights on the same database objects
  - To specify subset of objects more precisely than by name and to distinguish between groups of users
  - Finer and more general protection granularity

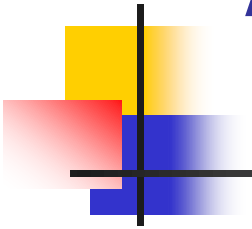
    i.e. (view,relation,tuple,attribute,content) vs. file
- Decentralized control
- In relational systems, authorizations can be uniformly controlled by database administrators using high-level constructs
  - By predicts in the same way as is a query qualification



4.2.1 Centralized Authorization Control

- Authorization control consists of checking whether a given triple **(user,operation,object)** can be allowed to proceed
- Three main actors
  - The users
    - Who trigger the execution of application programs
    - **(user name, password)**
  - The operations
    - Which are embedded in application programs
    - Such as **Select, Insert, Update, Delete**
  - The database objects
    - On which the operations are performed
    - Such as **view, relation, tuple, attribute, content**

Authorization control – Discretionary Access Control

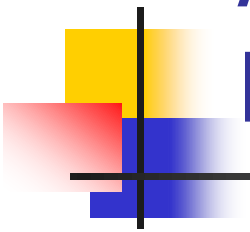


- A right expresses a relationship between a user and an object for a particular set of operations
  - **GRANT** <operation types> ON <object> TO <users>
  - **REVOKE** <operation types> FROM <object> To <users>
- The privileges of the subjects over objects are recorded in the directory as authorization rules.
- The most convenient approach is to consider all the privileges as an **authorization matrix**

Example of Authorization Matrix

	EMP	ENAME	ASG
Casey	UPDATE	UPDATE	UPDATE
Jones	SELECT	SELECT	SELECT Where RESP≠"Manager"
Smith	NONE	SELECT	NONE

Authorization control – Multilevel Access Control

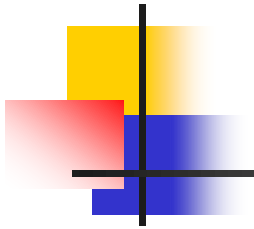


- Discretionary access control has some limitation: a malicious user can access unauthorized data through an authorized user.
- Multilevel access control defines security levels for both subjects and data objects.
- The security levels
  - Top Secret(TS) > Secret(S) > Confidential(C) > Unclassified(U)
- Two rules
  - No read up
    - A subject S is allowed to read an object of security level l only if level(S)>=l
  - No write down
    - A subject S is allowed to write an object of security level l only if level(S)<l

4.2.2 Distributed Authorization Control

- Remote user authorization
  - By user
  - By site
- Management of distributed authorization rules
  - directory
- Views
  - Composite objects – composed of other underlying objects
- User groups
  - Define GROUP <group_id> AS <list of subject ids>

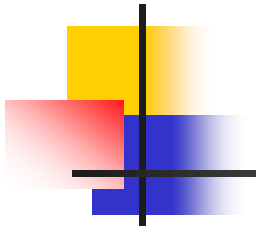
In conclusion, **full replication** of authorization information has two strong advantages: authorization control is much *simpler* and can be done at *compile time*. However, the *overhead cost* incurred for managing the distributed directory can be significant if there are many sites in the system



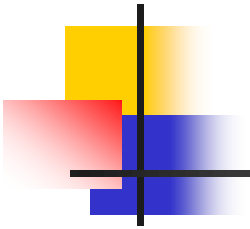
4.3 Semantic Integrity Control

Semantic Integrity Control

- How to guarantee database consistency
  - A database state is said to be *consistent* if the database satisfies a set of constraints, called *semantic integrity constraints*. In general,those are rules that represent the knowledge about the properties of an application.
  - Concurrency control, reliability, protection, and **semantic integrity control**
  - Semantic integrity control ensures database consistency by **rejecting** update programs which lead to inconsistent database states, or by **activating** specific actions on the database state, which compensate for the effects of the update programs



- 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
    - Language for expressing and manipulating integrity assertions
  - Integrity constraint enforcement
    - Performing specific actions to enforce database integrity at updates

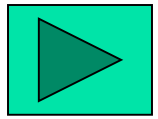


Procedural vs Declarative method

- Procedural
  - control embedded in each application program
- Declarative
  - expressing integrity constraints using 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

4.3.1 Centralized Integrity Control

- Specification of Integrity Constrains
  - Predefined constraints
  - Precompiled constraints
  - General constraints
- Integrity Enforcement
  - Detection
  - Prevention



Predefined constraints

- based on simple keywords
- specify the more common constraints of the relational model
  - **Not-null attribute**

    ```
    ENO NOT NULL IN EMP
    ```

  - **Unique key**

    ```
    (ENO, PNO) UNIQUE IN ASG
    ```

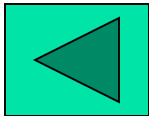
  - **Foreign key**

    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

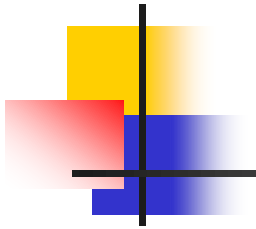
  (INSERT, DELETE, MODIFY)

  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>
```



- **Domain constraint**

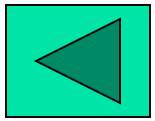
```
CHECK ON PROJ(BUDGET ≥500000 AND BUDGET
≤1000000)
```

- **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)
```



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>)
```

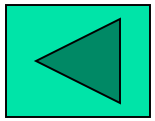
  - **Functional dependency**

```
CHECK ON e1:EMP, e2:EMP
        (e1.ENAME = e2.ENAME IF e1.ENO = e2.ENO)
```

  - **Constraint with aggregate function**

e.g. The total duration for all employees in the CAD project is less than 100

```
CHECK ON g:ASG, j:PROJ
        (SUM(g.DUR WHERE g.PNO = j.PNO) < 100 IF
                j.PNAME = "CAD/CAM")
```



Integrity Enforcement

- Detection - posttests

  Execute update u: D → Du

  If Du is inconsistent

  Then compensate Du → Du'

  Else undo Du → D

- Preventive - pretests

  Execute u: D → Du only if Du will be consistent

  - **Determine valid programs**
  - **Determine valid states**

Example of preventive method – Query Modification

- **Efficient esp for** domain constraints
- **Add the assertion qualification to the update query**
- **Only applicable to tuple calculus formulae with** universally quantified variables
- **Example: increasing the budget of CAD/CAM project by 10%**

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

UPDATE   PROJ
SET B UDGET = BUDGET*1.1
WHERE   PNAME ="CAD/CAM"
        AND NEW.BUDGET ≥500000
        AND NEW.BUDGET ≤1000000
```

General preventive method – Compiled Assertions

Triple *(R,T,*C) where

R	**relation**
T	**update type (insert, delete, modify)**
C	**assertion on differential relations**

Example: Foreign key assertion

∀ g∈ASG, ∃ j∈PROJ : g.PNO = j.PNO

Compiled assertions:

(ASG, INSERT**, C1), (PROJ,** DELETE**, C2), (PROJ,** MODIFY**, C3)**

where

C1: ∀ NEW∈**ASG+,** ∃ j∈**PROJ:** ***NEW.PNO = j.PNO***

C2: ∀ g∈**ASG,** ∃ OLD∈**PROJ- :** ***g.PNO ≠OLD.PNO***

C3: ∀ g∈**ASG,** ∀ OLD∈ **PROJ-,** ∃ NEW∈**PROJ+:**

g.PNO ≠OLD.PNO OR OLD.PNO = NEW.PNO

Differential Relations

Given relation *R* and update *u*

R^+ contains tuples inserted **by *u***

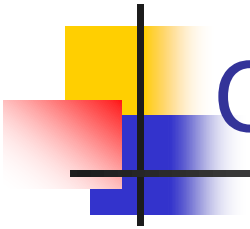
R^- contains tuples deleted **by *u***

Type of *u*

insert R^- **empty**

delete R^+ **empty**

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



Compiled assertion

- **When an integrity constraint I is defined, a set of compiled assertions may be produced for the relations used by I**
  - That permit the substitution of differential relations for base relation – simplification
- **Whenever a relation involved in I is updated by a program u, the compiled assertions that must be checked to enforce I are only those defined on I for the update type of u.**
- **Advantage:**
  - **The number of assertions to enforce is minimized**
    - **Since only the compiled assertions of type u need be checked**
  - **The cost of enforcing a compiled assertion is less than that of enforcing I**
    - **Since differential relations are, in general, much smaller than the base relations**

Enforcement Algorithm

Input: Relation R, update u, compiled assertion *Ci*

Step 1: Generate differential relations R^+ and R^-

Step 2: Retrieve the tuples of R^+ and R^-which do not satisfy *Ci*

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

Example :

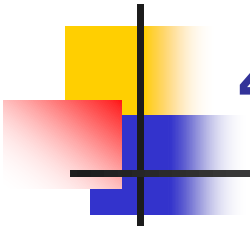
***u* is delete on J. Enforcing (J, DELETE, C2) :**

retrieve all tuples of J -

into RESULT

where not(C2)

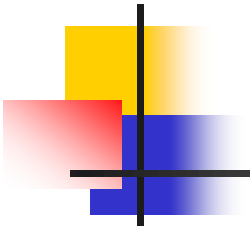
If RESULT = null , the assertion is verified.



4.3.2 Distributed Integrity Control

- Problems:
  - **Definition of constraints**
    - similar to the centralized techniques
    - consideration for fragments
    - **Step1**: transform the assertions to compiled assertions
      - Using the techniques in centralized system

      Step2: store compiled assertions according to the class of assertion
      - Since assertions can involve data stored at different sites, their storage must be decided so as to minimize the cost of integrity checking
  - **Enforcement**
    - minimize costs

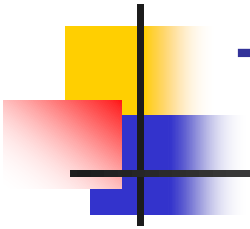


Assertion Enforcement

- **The main problem - Where do you enforce assertions?**

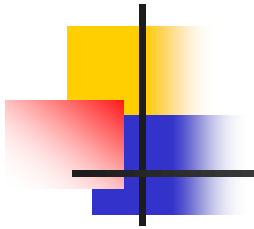
  Depends on:

  - type of assertion
  - type of update
  - where update is issued



Types of Distributed Assertions

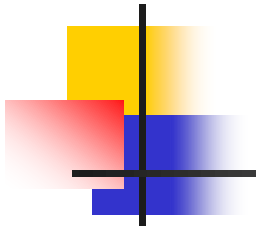
- **Individual assertions**
  - single relation, single variable
    - domain constraint
- **Set oriented assertions**
  - single relation, multi-variable
    - functional dependency
  - multi-relation, multi-variable
    - foreign key
- **Assertions involving aggregates**
  - Require special processing because of the cost of evaluating the aggregates
  - Treated like the above, depending on whether they are individual or set-oriented



- **Individual Assertions**

  e.g. domain constraint

  - update = insert
    - enforce at the site where the update is issued
  - update = qualified update (delete or modify)
    - send to all the sites involved
    - execute the qualification to obtain R+ and R-
    - each site enforce its own assertion



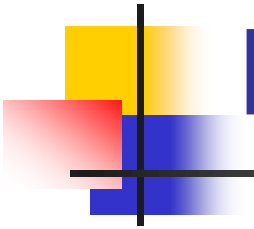
- **Set-oriented Assertions**
  - single relation

    e.g. functional dependency
    - similar to individual assertions with qualified updates
  - multi-relation

    e.g. foreign key
    - move data between sites to perform joins; then send the result to the query master site

Summary of Distributed Integrity Control

- The two main issues are the **definition** of the the distributed assertions and of the **enforcement** algorithm
- The main problem is that the communication and processing **cost** of enforcing distributed assertion can be prohibitive
- Distributed integrity control can be completely achieved by **extending** a general preventive method based on the compilation of semantic integrity assertions
- A better performance of distributed integrity enforcement can be obtained if fragments are defined carefully – i.e. the specification of distributed integrity constraints is an important aspect of the distributed database design process.



References

- M. Adiba, Derived Relations: A Unified Mechanism for Views, Snapshots and Distributed Data, In: Proc 7th Int. Conf. on Very Large Data Bases, Sept. 1981, 293-305
- P. F. Wilms and B. G. Lindsay, A Database Authorization Mechanism Supporting Individual and Group Authorization, Research Report RJ 3137, San Jose, Calif.: IBM Research Lab., May 1981.
- E. Simon and P. Valduriez, Integrity Control in Distributed Database Systems, In: Proc. 19th Hawaii Int. Conf. on System Science, Jan. 1986, 622-632