Semantic Integrity Control

- View management
- Security control
- Integrity control

View Management

- At the core, a view is a query. It can involve selection, projection, aggregate functions, etc.

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

Semantic Integrity Control

- Views enable full logical data independence
- Views define virtual relationships (views are queries over the base relationships, however typically the views are not materialized)

View Management (Centralized Databases)

- Queries expressed on views are translated into queries expressed on base relationships

```
SELECT ENAME, PNO, RESP
FROM SYSAN, ASG
WHERE SYSN.ENO = ASG.ENO
AND TITLE = "Syst. Anal."
```

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

View Management (Centralized Databases, Query Modification)

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

```
EMP
<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>M. Smith</td>
<td>Syst. Anal</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>Mech. Eng</td>
</tr>
<tr>
<td>E4</td>
<td>J. Miller</td>
<td>Programmer</td>
</tr>
<tr>
<td>E5</td>
<td>B. Casey</td>
<td>Syst. Anal</td>
</tr>
<tr>
<td>E6</td>
<td>L. Chu</td>
<td>Elect Eng</td>
</tr>
<tr>
<td>E7</td>
<td>R. Davis</td>
<td>Mech. Eng</td>
</tr>
<tr>
<td>E8</td>
<td>J. Jones</td>
<td>Syst. Anal</td>
</tr>
</tbody>
</table>

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

```
<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 (Centralized Databases, Update Queries)

 eş Views can be updatable and not-updatable

 eş Updatable

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

 eş Non-updatable

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

### View Management (Distributed Databases)

 eş Views are conceptually the same as the base relations, therefore we store them in the directory

 eş Two issues:
  - How do we decompose select queries
  - How do we decompose update queries

 eş Queries are translated into distributed queries due to possible fragmentation of data

 eş Snapshots can be used if base relations are fragmented. Snapshots
  - are static views: do not reflect the updates to the base relations
  - managed as temporary relations: the only access path is sequential scan
  - typically used when selectivity is small (no indices can be used efficiently)
  - is subject to periodic recalculation

### Data Security

**Data Security Aspects:**

 eş Data Protection

 eş Authorization Control

### Data Security (Data Protection)

**Well established standards:**

 eş Data encryption standard

 eş Public-key encryption schemes
Three aspects are involved in authorization: (user, operation, data object)

Introduction to the system is done by a pair (username, password)

Data objects are files in a file system; views, relations, tuples, attributes in DBMS

Privileges are stored in the directory, conceptually as a matrix

<table>
<thead>
<tr>
<th>EMP</th>
<th>ENAME</th>
<th>ASG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casey</td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
<tr>
<td>Jones</td>
<td>SELECT</td>
<td>SELECT WHERE RESP ≠ “Manager”</td>
</tr>
<tr>
<td>Smith</td>
<td>NONE</td>
<td>SELECT</td>
</tr>
</tbody>
</table>

Different materialization of the table is possible (by row, by columns, and by element, depending on the optimization strategy)

Data Security (Authorization Control, Centralized 2/3)

Grand and revoke statements are used to authorize triplets (user, operation, data object)

- GRANT <operations> ON <object> TO <users>
- REVOKE <operations> ON <object> TO <users>

Typically, the creator of objects gets all permissions

Authorization schemes:

- (username, password) is communicated between the sites whenever the relationships are accessed (beneficial if the users move from a site to a site)
- (username, password) is authorized by the GUI/application at the start of the session. No authorization is required for accessing relations further (beneficial if the users are static)
- Distribution of the directory must be designed carefully to support efficient authorization
Semantic Integrity Constraints

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 Constraints (Specification Language)

SQL like language:
- Not-null attribute: ENO NOT NULL IN EMP
- Unique key: (ENO, PNO) UNIQUE IN ASG
- Foreign key: PNO IN ASG REFERENCES PNO IN PROJ
- Functional dependency: ENO IN EMP DETERMINES ENAME

Semantic Integrity Constraints (Definition)

Procedural control embedded in each application program (inefficient, therefore is rarely used)

Declarative
- assertions in predicate calculus
- easy to define constraints
- definition of database consistency clear

Semantic Integrity Constraints (Specification Language, Precompiled Constraints)

Pre-compiled constraints are constraints that must be satisfied by all tuples in the relation

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)
Semantic Integrity Constraints (Specification Language, General Constraints)

- General Form:
  \[ \text{CHECK ON } \langle \text{variable} \rangle : \langle \text{relation} \rangle \ (\langle \text{qualification} \rangle ) \]

- Functional dependency:
  \[ \text{CHECK ON } e_1: \text{EMP}, e_2: \text{EMP} \\
  (e_1.\text{ENAME} = e_2.\text{ENAME} \ \text{IF} \ e_1.\text{ENO} = e_2.\text{ENO}) \]

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

 Semantic Integrity Constraints (Integrity Enforcement)

- Two methods to enforce integrity:
  - Detection:
    1. Execute update \( u : D \rightarrow D_u \)
    2. 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

 Semantic Integrity Constraints (Types of Distributed Constraints)

- Individual assertions (constraints)
  - Single relation
  - Domain constraint

- Set oriented assertions
  - Single relation, multi variable
  - Multi-relation, multi-variable

- Assertion involving aggregates

 Semantic Integrity Constraints (Distributed Constraints)

- Difficulty: data is fragmented and replicated
  - Where to store
  - How to enforce
Conclusion

Views enable full logical data independence
- Queries expressed on views are translated into queries expressed on base relationships
- Views can be updatable and non-updatable

Three aspects are involved in authorization: (user, operation, data object)
- There are two authorization schemes: (i) authorization is performed whenever a relation is accessed, or (ii) at the beginning of the session

Semantic integrity constraints maintain database consistency
- Individual assertions are checked at each fragment site, check for compatibility
- Involves joins between fragments and optimal enforcement of the constraints is similar to distributed query optimization

Semantic Integrity Constraints (Storage of Assertions)

Individual assertions
- At each fragment site, check for compatibility
- If compatible, store; otherwise reject
- If any of 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

Semantic Integrity Constraints (Enforcement of Assertions)

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
  - 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