CSC 405
Introduction to Computer Security

Topic 6.2 Multi-Level Databases

MAC in DBMS

- Attribute values and tuples are considered as objects
  - Each attribute $A$ is associated with a classification attribute $C$ (the label)
  - In some models, a tuple classification attribute $TC$ is added to the relation
  - Example:
    - Employee ($SSN$, $Name$, $Salary$, $Performance$)
    - Employee ($SSN$, $C_{SSN}$, $Name$, $C_{Name}$, $Salary$, $C_{Salary}$, $Performance$, $C_{Performance}$, $TC$)
- Such a relation is called a multi-level relation
MAC in DBMS (Cont’d)

- Employee (SSN, C_{SSN}, Name, C_{Name}, BDate, C_{BDate}, Salary, C_{Salary}, TC)

- Primary key:
  - The set of attributes that can uniquely identify each tuple.

- Apparent key:
  - The set of attributes that would have formed the primary key in a regular (single-level) relation.
Polyinstantiation

- Several tuples can have the same apparent key value but have different attribute values for users at different classification levels.

<table>
<thead>
<tr>
<th>ShipID</th>
<th>C_S</th>
<th>Mission</th>
<th>C_M</th>
<th>Target</th>
<th>C_T</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Attack</td>
<td>S</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Explore</td>
<td>U</td>
<td>Moon</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>Enterprise</td>
<td>C</td>
<td>Explore</td>
<td>C</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

Is this possible?

<table>
<thead>
<tr>
<th>ShipID</th>
<th>C_S</th>
<th>Mission</th>
<th>C_M</th>
<th>Target</th>
<th>C_T</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Attack</td>
<td>S</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Explore</td>
<td>U</td>
<td>Moon</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>Enterprise</td>
<td>C</td>
<td>Explore</td>
<td>C</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

What could be the real key?
### What if?

#### Mission

<table>
<thead>
<tr>
<th>ShipID</th>
<th>Cₙ</th>
<th>Mission</th>
<th>Cₘ</th>
<th>Target</th>
<th>Cₜ</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Attack</td>
<td>S</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Voyager</td>
<td>U</td>
<td>explore</td>
<td>C</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Explore</td>
<td>U</td>
<td>Moon</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>Enterprise</td>
<td>C</td>
<td>Explore</td>
<td>C</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

What could be the real key?

#### Class C user sees

<table>
<thead>
<tr>
<th>ShipID</th>
<th>Cₙ</th>
<th>Mission</th>
<th>Cₘ</th>
<th>Target</th>
<th>Cₜ</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Null</td>
<td>C</td>
<td>Null</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>Enterprise</td>
<td>C</td>
<td>Explore</td>
<td>C</td>
<td>Null</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

Class C user:

```
UPDATE Mission
SET Mission = ‘Explore’, Target = ‘Moon’
WHERE ShipID = ‘Voyager’
```
After Update

Mission

<table>
<thead>
<tr>
<th>ShipID</th>
<th>CS</th>
<th>Mission</th>
<th>CM</th>
<th>Target</th>
<th>CT</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voyager</td>
<td>U</td>
<td>Attack</td>
<td>S</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Enterprise</td>
<td>C</td>
<td>Explore</td>
<td>C</td>
<td>Mars</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

What should be returned to a class C user?
How about a class S user?
What is the general method?
Mission

<table>
<thead>
<tr>
<th>ShipID</th>
<th>Mission</th>
<th>Target</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voyager</td>
<td>Attack</td>
<td>Mars</td>
<td>S</td>
</tr>
<tr>
<td>Voyager</td>
<td>Attack</td>
<td>Mars</td>
<td>S</td>
</tr>
<tr>
<td>Voyager</td>
<td>Explore</td>
<td>Moon</td>
<td>S</td>
</tr>
<tr>
<td>Enterprise</td>
<td>Explore</td>
<td>Mars</td>
<td>S</td>
</tr>
</tbody>
</table>

What to return to Class C user?

Integrity Constraints for Multi-level relations

- **Entity integrity**
  - All attributes that are members of the apparent key must not be null and must have the same security class.
  - All other attribute values in the tuple must have a security class greater than or equal to that of the apparent key.
  - **Purpose**: make the retrieved information meaningful.

- **Null integrity**
  - If a tuple value at some security level can be derived from a higher-level tuple, then it’s sufficient to store the higher-level tuple.
  - **Purpose**: Reduce redundancy
Approaches to Multi-level Databases

- Partitioning
- Encryption
- Integrity lock
- Trusted Front-End
- Distributed Databases

Partitioning

- Separate data in different levels into different partitions.
  - Redundancy
    - Example: the primary key of a logical relation must be duplicated in all partitions in which the relation are stored.
  - Usability
    - Example: a high-level user needs to combine both high-level and low-level data.
Encryption

- Encrypt the sensitive data at each level with a key unique to that level.
  - Known plaintext attack
    - Example:
      - Party attribute is encrypted.
      - Alice knows party="Democrat" for Bob; she can compare the ciphertext of Bob’s party attribute with other tuples
    - Reason: Limited set of plaintexts.
  - Authentication
    - Example:
      - Replace one ciphertext with another
  - Above problems can be partially avoided with multiple keys.
  - Unable to use DBMS functionalities for encrypted data.
    - Query optimization, indexes, etc.

Integrity Lock

- Provide integrity and limited access for a database.

Data \rightarrow Security class \rightarrow Crypto checksum

Secret Agent \rightarrow TS \rightarrow 10FB

Key \rightarrow Encryption Function

- Any unauthorized changes to data items can be detected.
- Access to data items is based on the security labels.


**Integrity Lock DBMS**

- **Problems**
  - Efficiency
    - Data expansion
    - Processing time required for generating, modifying, and verifying integrity locks
  - Security
    - Untrusted DBMS sees all data passing through it.

**Trusted Front End**

- **Trusted Front End**
  - User authentication
  - Access control
  - Verification
  - Essentially a reference monitor
Trusted Front End (Cont’d)

- **Commutative Filters**
  - Processes that interfaces to both the user and the DBMS.
  - Reformat the query by putting in more conditions to filter out unnecessary records.
  - Example:
    - Retrieve NAME where ((Occup= Physicist) \(^\land\) (City =WashDC))
      From all records R
    - After reformatting
    - Retrieve NAME where ((Occup= Physicist) \(^\land\) (City =WashDC))
      From all records R where
        - (Name-level (R) <= User-level) \(^\land\)
        - (Occup-level (R) <= User-level) \(^\land\)
        - (City-level (R) <= User-level)

Distributed Databases

- Store data items at different level in different physical databases
- Trusted front-end translates each query into single-level queries and send to different databases
- Trusted front-end combines results and returns to the user.