Database Security

Course 4

The inference problem (part 1)

Processing of security constraints for inference control

Plan

1. Introduction
2. Security Constraints
3. An Approach to the Processing of Security Constraints
4. Constraints' consistency and completeness
5. Design of the query processing module
6. Design of the update processing module

1. Introduction (1)

- Security constraints will be approached from the perspective of the inference problem.
- Security constraints are **rules** that assign **security levels** to data.
- These constraints include those that classify data according to:
  - content
  - context
  - aggregation
  - time.

1. Introduction (2)

- Ways of dealing with security constraints **during the query processing** have been studied so that certain non-compliance with security, because of inference, does not occur.
- Another study was focused on the processing of constraints **during the design of the database**; the goal of the security constraints' processing is to avoid the non-compliance with security.
  - We will describe the design techniques for the processing of security constraints.

1. Introduction (3)

- Adequate processing of security constraints is essential for the development of a useful and secure MLS/DBMS.
- Following the analysis of the different types of security constraints, they can be seen as **a form of integrity constraints** applied in MLS/DBMS.
  - This is because in a multilevel database, the security level of an entity can be viewed as part of the value of that entity.
  - Therefore, security constraints specify allowed values that an entity can take.

1. Introduction (4)

- Because a security constraint can be viewed as a form of integrity constraint, many of the techniques developed by logical programming researchers to address integrity constraints in non-MLS relational database systems can be used to address security constraints in an MLS/DBMS.
- Within these techniques:
  - some integrity constraints (called derivation rules) are addressed during the query processing;
  - other integrity constraints (known as integrity rules) are addressed during the database updates;
  - certain constraints (known as schema rules) are addressed during the database design.

1. Introduction (5)

- A **query processing module** must also have the ability to process **the security constraints**.
- This is due to the fact that users usually build the "reservoir" of knowledge from the answers they receive by querying the database.
- Subsequently, users deduce the secret information from this reservoir.
- Moreover, no matter how well the security database has been designed or whether the data in the database is properly labeled with security labels, users will eventually be able to overcome security due to inference because they update continuously their reservoir of knowledge as things evolve.
- It is not possible to redesign the database or reclassify the data continuously.

1. Introduction (6)

- It should be noted that the **processing of a large number of security constraints** could have an **impact on the performance of query processing algorithms**.
- Therefore, it is desirable to process as many constraints as possible **during database updates** and **database design**.
  - This is because, in general, the database design and the update operation are performed **less frequently** than the query operation.
- Therefore, when the database is initially designed, the security constraints should be examined, and the schema should be generated.
- When data is updated, constraints are reviewed and security levels are assigned or reassigned to the affected data.

1. Introduction (7)

- The System Security Officer (SSO) should periodically review security constraints, redesign the database, and reclassify the data.
- If the application is **static** and the data in the database is consistent with the security constraints, the query processing module **should not examine the constraints** handled by the update processing module and the database designer.
- If there have been **changes** in the real world that cause the database or schema to become inconsistent, then the update processing module should be triggered to **process the relevant constraints during its operations**.
- In this way, much of the tasks assigned to the query processor are made easier.

2. Security Constraints (1)

- The types of security constraints that have been identified include the following:

1. ***Simple constraints*** – classify a database, relationship or attribute.
2. ***Content-based constraints*** – classify any part of the database according to the **value** of certain data.
3. ***Event-based constraints*** – classify any part of the database based on the **occurrence of a real-world event**.
4. ***Association-based constraints*** – classify the relationships between data.
5. ***Delivery-based constraints*** – classify any part of the database according to the **information that was previously provided**. 2 types of such constraints have been identified:
    - **general constraints**, which classify an entire attribute according to the condition that a value of another attribute has been provided;
    - **individual constraints**, which classify the value of an attribute according to the condition that a value of another attribute has been provided.

2. Security Constraints (2)

6. Aggregate constraints – useful for classifying data **collections**.

7. Logical constraints – specify **implications**.

8. Constraints with conditions – they have **conditions** associated with them.

9. Level-based constraints – classify any part of the database according to **the security level of certain data**.

10. Fuzzy constraints – assign **fuzzy values** to the classification.

2. Security Constraints (3)

- We will give examples of security constraints of each type.
- The examples refer to a database in which there are 2 relations:
  - SHIP(S#, SNAME, CAPTAIN, M#)
  - MISSION(M#, MNAME, LOCATION)
- S#, M# are keys, and the domain of M# from the 2 relations is the same.

2. Security Constraints (4)

- Constraints can be expressed **in the form of logical rules**.
- We will use **the Horn clauses** to represent the constraints.
- In this way we can benefit from the techniques that have been developed for logical programming.

2. Security Constraints (5)

1. **Simple constraints**

- *R(A1, A2,…, An) -> Level(Ai1, Ai2,…, Ait) = Secret*
- Each attribute Ai1, Ai2,…, Ait of the relation R is Secret.
- *Example*: SHIP(S#, SNAME, CAPTAIN, M#) -> Level (CAPTAIN) = Secret

2. **Content-based constraints**

- *R(A1, A2,…, An) AND COND(Value(B1, B2, …, Bm)) -> Level(Ai1, Ai2,…, Ait) = Secret*
- Each attribute Ai1, Ai2,…, Ait of the relation R is Secret if a condition is applied over the values of certain data, specified by B1, B2, …, Bm.
- *Example*: SHIP(S#, SNAME, CAPTAIN, M#) AND (Value(SNAME) = Washington) -> Level (CAPTAIN) = Secret

2. Security Constraints (6)

3. Association-based constraints (context constraints)

- *R(A1, A2,…, An) -> Level(Together(Ai1, Ai2,…, Ait)) = Secret*
- The attributes Ai1, Ai2,…, Ait of the relation R, taken together, are Secret.
- *Example*: SHIP(S#, SNAME, CAPTAIN, M#) -> Level (Together(SNAME, CAPTAIN)) = Secret

4. Event-based constraints

- *R(A1, A2,…, An) AND Event(E) -> Level(Ai1, Ai2,…, Ait) = Secret*
- Each attribute Ai1, Ai2,…, Ait of the relation R is Secret if the event E has appeared.
- *Example*: SHIP(S#, SNAME, CAPTAIN, M#) AND Event(Change of President) -> Level(CAPTAIN, M#) = Secret

2. Security Constraints (7)

5. General delivery-based constraints

- *R(A1, A2,…, An) AND Release(Ai, Unclassified) -> Level(Aj) = Secret*
- The attribute Aj of the relation R is Secret if the attribute Ai has been delivered at the level Unclassified.
- *Example*: SHIP(S#, SNAME, CAPTAIN, M#) AND Release(SNAME, Unclassified) -> Level(CAPTAIN) = Secret

6. Individual delivery-based constraints

- *R(A1, A2,…, An) AND Individual-Release(Ai, Unclassified) -> Level(Aj) = Secret*
- The individual delivery-based constraints classify the elements of an attribute at a particular level, after the elements of another attribute had been delivered. These constraints are more difficult to implement than the general ones.

2. Security Constraints (8)

7. Aggregate constraints

- These constraints classify the collections of tuples taken together at a higher level than the individual levels of the tuples in the collection. There could be a semantic association between tuples. These tuples can be specified as follows:

  R(A1, A2,…, An) AND Set(S, R) AND Satisfy(S, P) -> Level(S) = Secret

- If R is a relation and S is a set containing the tuples of R, and S satisfies a property P, then S is classified at the Secret level.
- P can be any property (for example: the number of elements is greater than 10).

2. Security Constraints (9)

8. Logical constraints

- They are rules that are used to derive new data from existing data in the database. The data obtained can be classified using one of the other constraints. The form of a logical constraint is the following:
  - *Ai => Aj*

  where Ai, Aj are attributes either of a relation in the database, or of an external one.
- Logical constraints are not really security constraints, in the sense that they do not associate security levels to data.
- In fact, they are integrity constraints.
- In particular, they can be seen as integrity constraints that are treated as derivation rules.

2. Security Constraints (10)

9. Constraints with conditions

- The other types of constraints can be specified with conditions.
- Consider the example: *Ai => Aj if condition C is true*.
- *Example*: The attribute LOCATION of the relation MISSION implies MNAME if LOCATION = Atlantic Ocean

10. Level-based constraints

- *R(A1, A2,…, An) AND Level(Ai) = Unclassified -> Level(Aj) = Secret*
- The attribute Aj of the relation R is Secret if the attribute Ai is Unclassified.
- *Example*: SHIP(S#, SNAME, CAPTAIN, M#) AND Level(SNAME) = Unclassified -> Level(CAPTAIN) = Secret.

2. Security Constraints (11)

11. Fuzzy constraints

- These constraints use fuzzy values and can be associated with any other type of constraint. An example of a fuzzy constraint associated with a content-based constraint is:
- *R(A1, A2,…, An) AND COND(Value(B1, B2, …, Bm)) -> Level(Ai1, Ai2,…, Ait) = Secret and Fuzzyvalue = r*
- Each attribute Ai1, Ai2,…, Ait of the relation R is Secret with a *fuzzy* value r if a condition is applied over the values B1, B2, …, Bm.

2. Security Constraints (12)

12. Complex constraints

- The previous examples refer to a single relationship. Constraints can, however, be applied **to several relationships**. Such constraints are called **complex**. An example of a complex constraint is the following:
- *R1(A1, A2, … An) & R2(B1, B2,…, Bm) & R1.Ai = R2.Bj -> Level(Together(Ak, Bp)) = Secret*
- The previous constraint states that a pair of values that includes the *k* attribute of the R1 relation and the *p* attribute of the R2 relation has the Secret level, provided that the corresponding values (from the same row) of the *i* attribute of the R1 relation and the *j* attribute of the R2 relation are equal.
- *Example*: SHIP(S#, SNAME, CAPTAIN, M#) & MISSION(M#, MNAME, LOCATION) & SHIP.M# = MISSION.M# -> Level(Together (SNAME, MNAME) = Secret

3. An Approach to the Processing of Security Constraints (1)

- Security constraints lead to the **application of a security policy**.
- Therefore, it is essential that these constraints be processed only by an authorized person (SSO).
- We assume that the **constraints** themselves **can be classified on several levels**. They are stored at the system level.
- **The constraint manager**, which is a secure component, will ensure that a user can only read constraints classified at a lower or equal level to the user.

3. An Approach to the Processing of Security Constraints (2)

- The presented approach regarding the processing of security constraints addresses:
  - certain constraints during the processing of queries,
  - others during database updates, and
  - certain constraints during database design.
- We will briefly illustrate the architectures for processing constraints during database design and the query and update operations.

3. An Approach to the Processing of Security Constraints (3)

- **The architecture for query processing** – is presented in figure 1.
- This architecture can be seen as a (weak) **link between an MLS / DBMS and a deductive manager**, which is what we called query processing module.
- It must operate online.

3. An Approach to the Processing of Security Constraints (4)

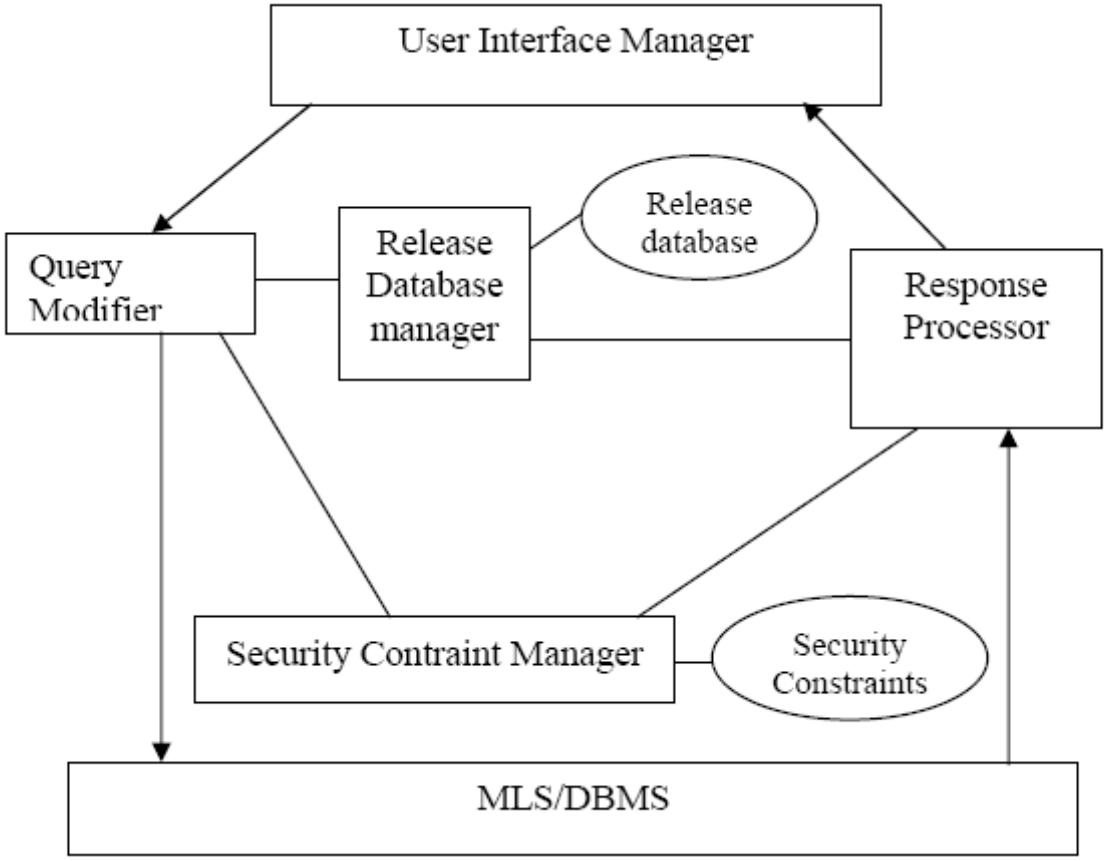

Figure 1. Query processing module (design details)

3. An Approach to the Processing of Security Constraints (5)

- **The architecture for updates processing** is shown in Figure 2. It can also be seen as a link between an MLS / DBMS and a deductive manager (update processing module).
- This module could be used:
  - online where data security levels are determined during insertions and updates in the database;
  - offline as a tool that ensures that data entered or updated through bulk data mechanisms are properly labeled.

3. An Approach to the Processing of Security Constraints (6)

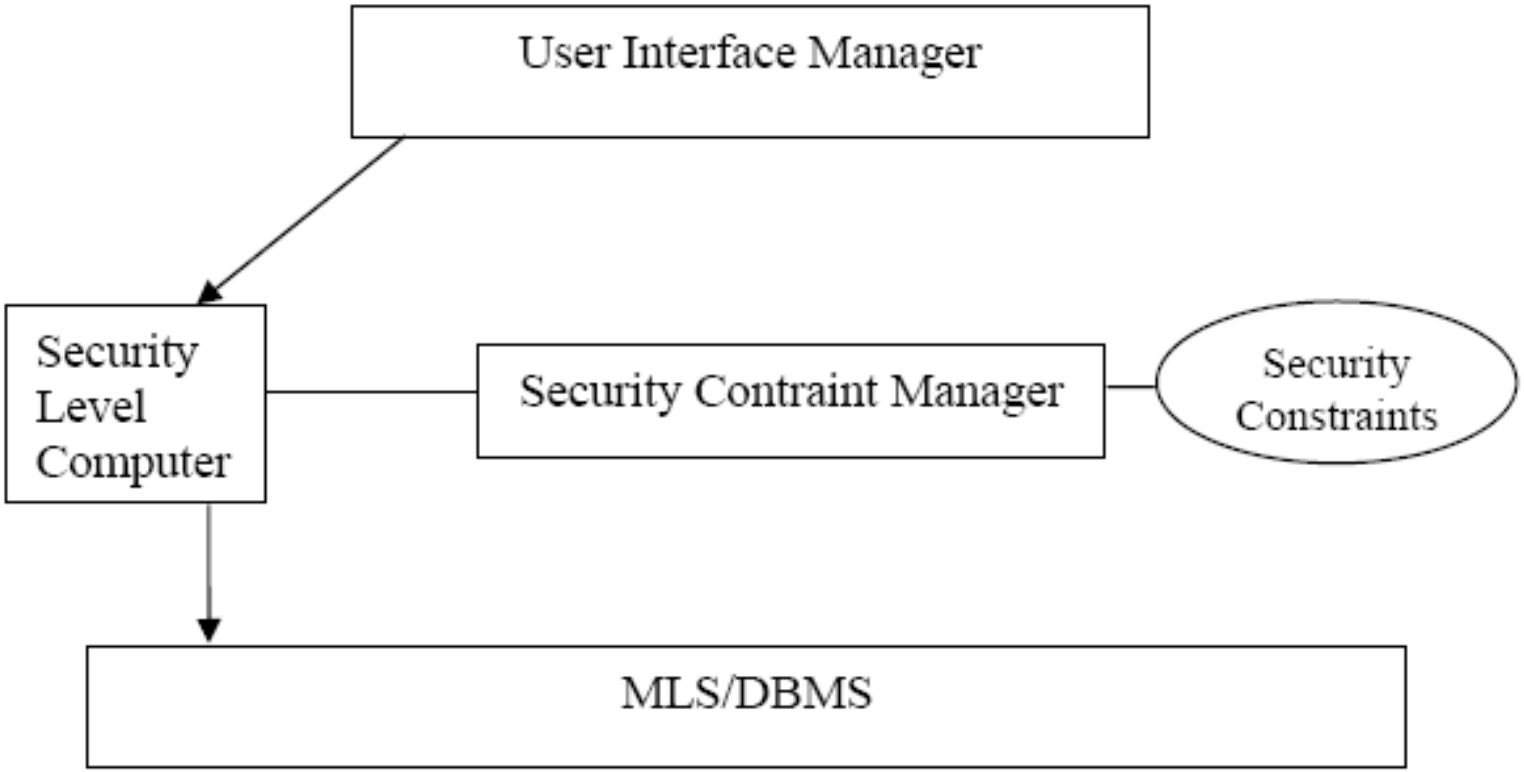

Figure 2. Update processing module

3. An Approach to the Processing of Security Constraints (7)

- If this tool is used offline, it may be difficult to recalculate levels for data that are already in the database if these levels are affected by new data being inserted.
- The tool that handles **security constraints during database design**, illustrated in Figure 3, can be used by the SSO to design the schema.
  - The input is the set of security constraints that should be processed during the database and schema design.
  - The output consists of the modified schema and constraints.

3. An Approach to the Processing of Security Constraints (8)

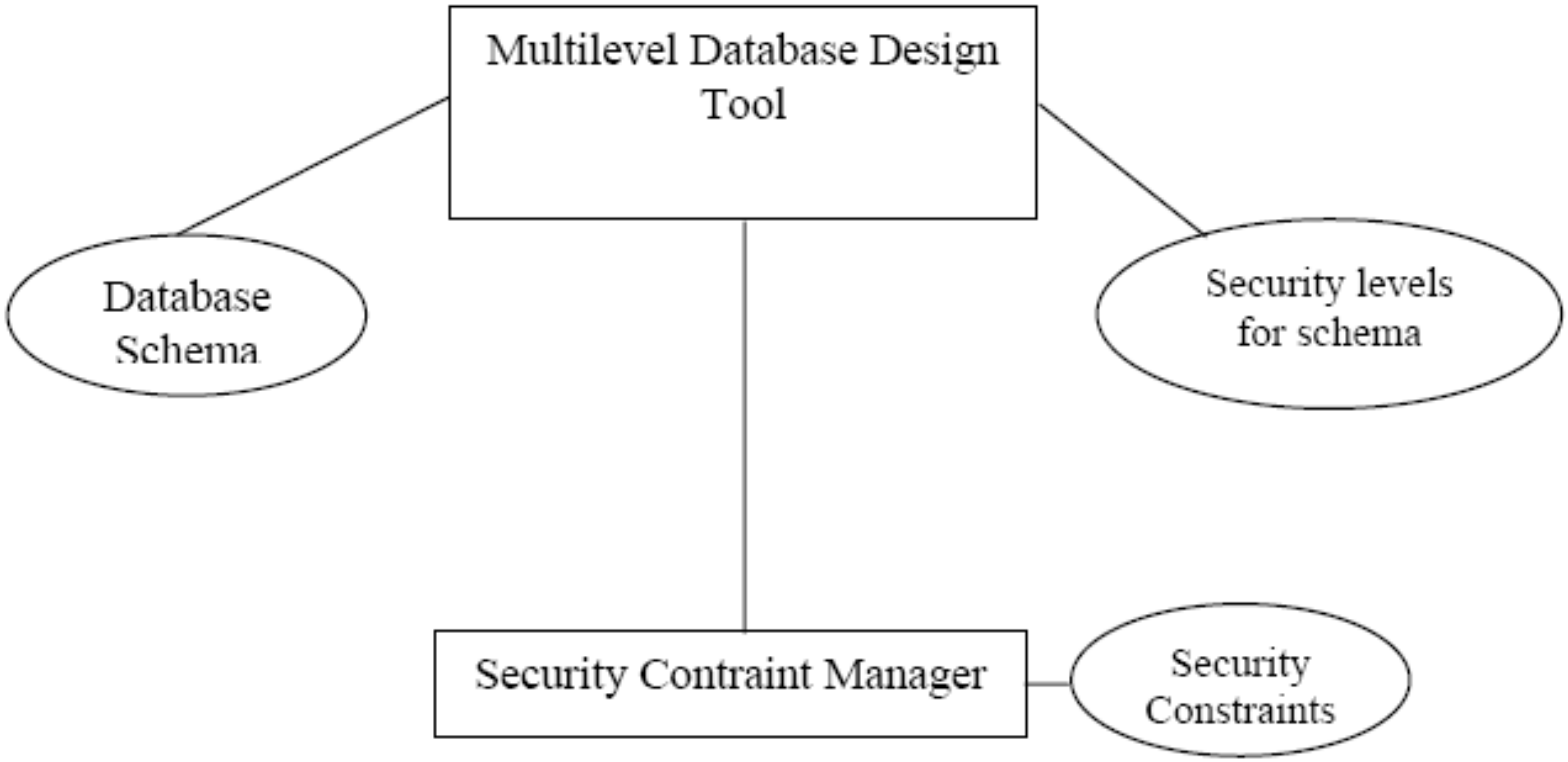

Figure 3. Database design tool

3. An Approach to the Processing of Security Constraints (9)

- Although the query processing, update processing and database design modules are separate, they all together provide the solution to processing constraints in multilevel relational databases.
- This means that they provide an **integrated solution to the processing of security constraints in a multilevel environment**.
- Figure 4 illustrates this integrated architecture.

3. An Approach to the Processing of Security Constraints (10)

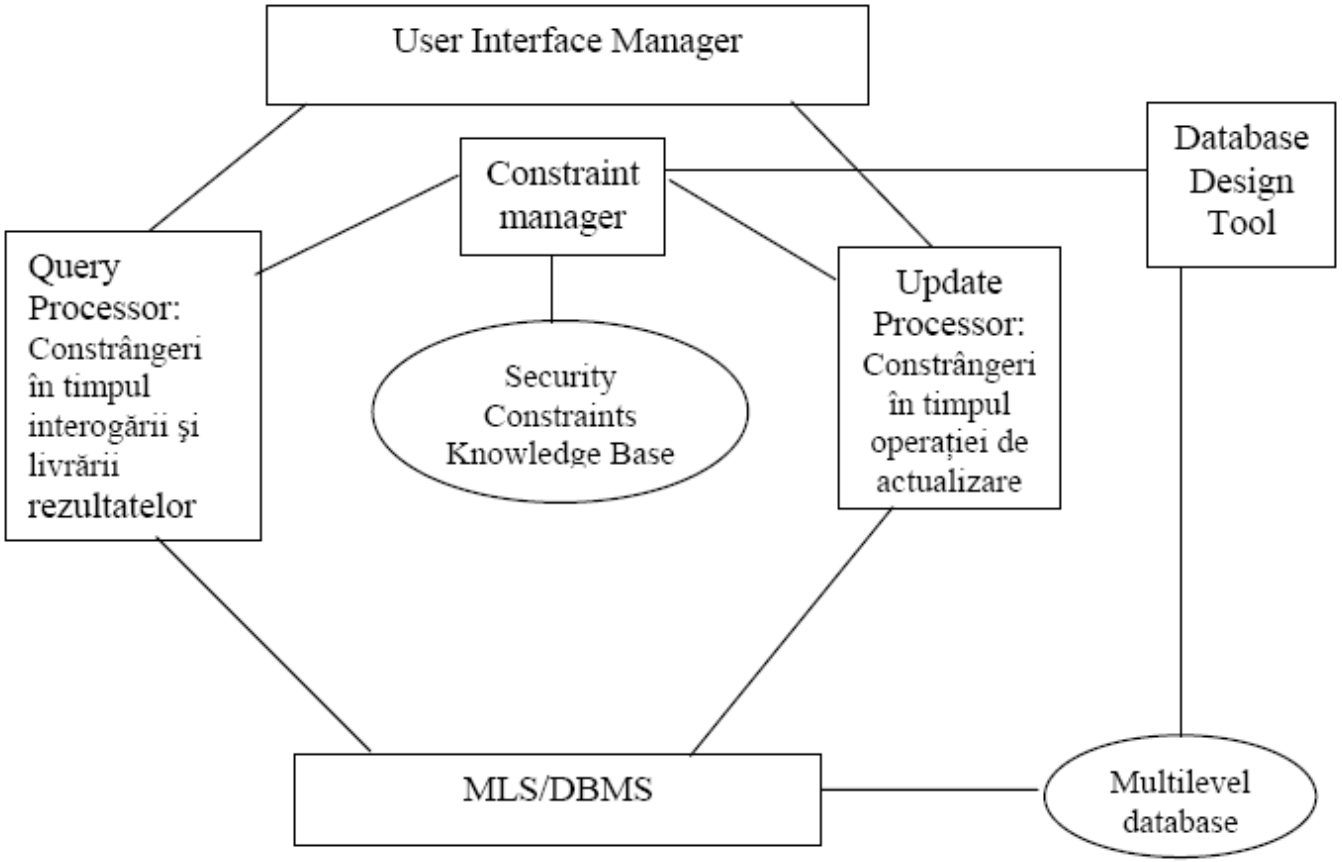

Figure 4. The integrated architecture

3. An Approach to the Processing of Security Constraints (11)

- The constraints and the schema produced by the **constraint generator** are further processed by the **database design tool**.
- Modified constraints are given to ***Constraint Updater*** to update the constraints database.
- The schema is given to ***Constraint Generator***.
- The constraints in the constraint database are used by the query and update processing modules.
- We assume that there is a **secure constraint manager** that handles these constraints. In a dynamic environment where data and constraints change, the query processing module will examine all **relevant constraints** and ensure that users do not obtain unauthorized data.

4. Constraints' consistency and completeness (1)

- The two tasks imposed by dealing with constraints are the **generation of constraints** and **their application**. The relationship between these is illustrated in Figure 5.
  - The constraint generator retrieves the multilevel application specification and determines the initial schema and the constraints to be applied.
  - The database design tool takes this result and designs the database.
  - The update and query processing modules use the constraints and schemas produced by the database design tool.
- Generating **the initial set of constraints** remains a challenge. It is necessary to analyze **the use of conceptual structures and semantic data models** to see if we can specify the application and, consequently, generate security constraints.

4. Constraints' consistency and completeness (2)

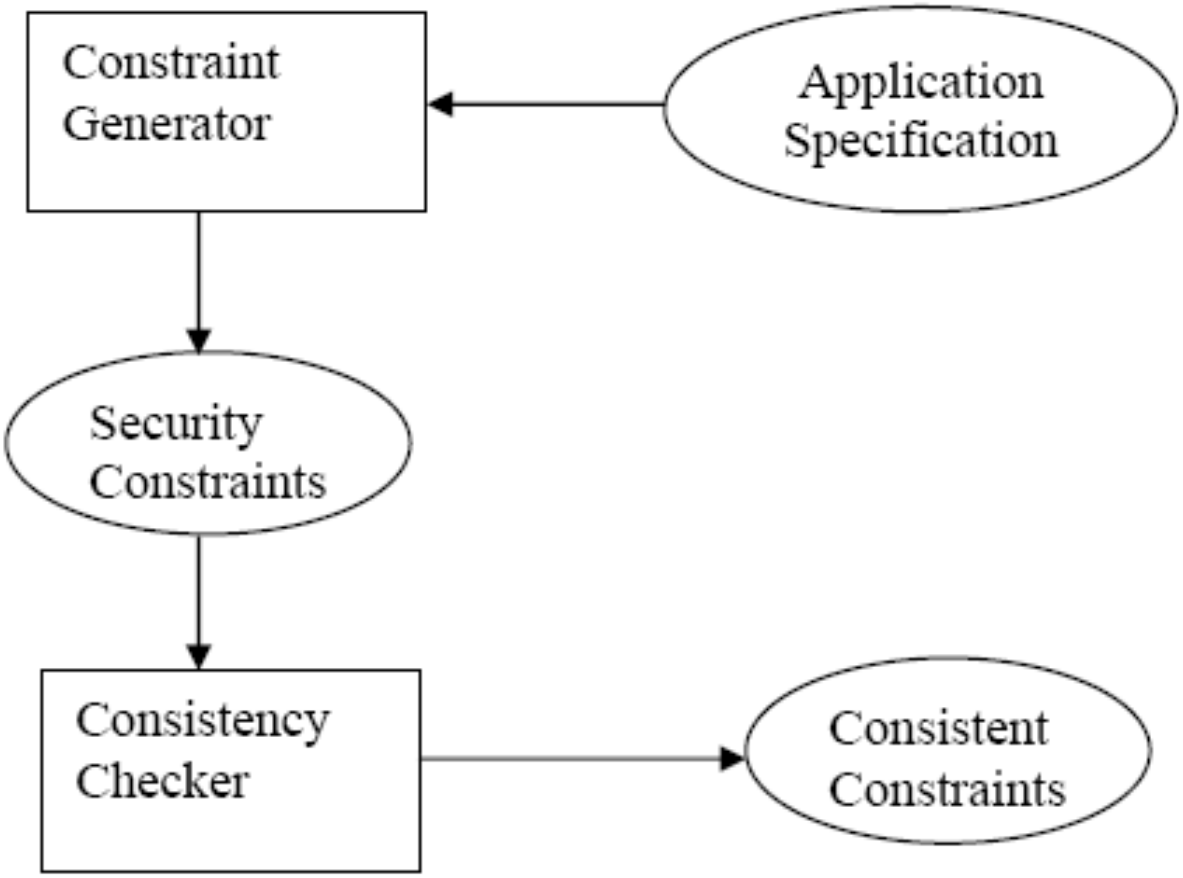

Figure 5. Constraint generation

4. Constraints' consistency and completeness (3)

- The following algorithm determines the consistency and completeness of security constraints.
- **Algorithm**

Input : **The set of security constraints**

Output : **The set of consistent and complete security constraints**

Step 1. For each relation and attribute, compute the associated security level.

Step 2. If there are constraints that assign multiple security levels to a relation or attribute, then delete or update the constraint that assigns the lowest security level; the constraint is changed if it assigns security levels to other attributes or relations.

Step 3. If there is a relation or attribute that is not associated with a security level by the constraints, then create a constraint that will assign the lowest security level to the relation or attribute.

Step 4. The resulting set of constraints is the output of the algorithm.

5. Design of the query processing module (1)

- We present a **security policy for dealing with inference while processing the queries**, and then we introduce an approach to module design.

The security policy

- The security policy presented for query processing extends the simple security property in order to address the inference problem.
- *Given a security level L, we denote by E(L) the knowledge base associated with L. Therefore, E(L) will consist of all the answers that were delivered at the security level L in a certain period of time and the information from the real world at the security level L.*

5. Design of the query processing module (2)

- *We consider that a **user U** with **security level L** launches a query. Then the **answer R** to the query will be delivered to the user if the following **condition** is met:*
- *For all security levels L*, where L* dominates L, if (E (L*) UNION R) ==> X, for any X, then L* dominates Level (X).*
- *A ==> B means that B can be deduced from A using any of the inference strategies*
- *Level (X) is the security level of X.*

5. Design of the query processing module (3)

- We assume that any response that is delivered in a knowledge base at the level *L* is also delivered in the knowledge base at the level *L* > = L*.
- The policy states *that when a response is delivered to a user at the level L, we must ensure that any user at the L* >= L cannot deduce information classified at a level L+ >= L from the response together with the knowledge that has been already obtained.*
- We consider only hierarchical levels in the policy specification, but things can be extended to non-hierarchical levels.

5. Design of the query processing module (4)

Functionality of the query processing module

- The power of the query processing module depends on **the type of inference strategies it can handle**. In the design we present, only a limited set of inference strategies, such as **inference by association and deduction**, is considered.
- We will discuss the techniques that were used to implement the security policy: **query modification and response processing**.
- It should be noted that most of the code for query modification and response processing modules must be **secure** because it performs security-critical functions.

5. Design of the query processing module (5)

Query rewrite

- The query modification technique has been used in the past in the context of discretionary security and views.
- Subsequently, this technique was extended to include mandatory security.
- In the query processor's design, this technique is used by the **inference engine** to modify the query based on **security constraints, previously delivered responses, and real-world information**.
- When the modified query is launched, the generated response will not overcome security.

5. Design of the query processing module (6)

- We consider the architecture for query processing presented in figure 1.
- The inference engine has access to the knowledge base which includes:
  - security constraints
  - the answers provided previously and
  - real world information.
- Conceptually, we can think of the database as part of the knowledge base.
- We illustrate the technique of query modification by examples.

5. Design of the query processing module (7)

- Consider the database containing the SHIP and MISSION relations described before. We assume that the knowledge base consists of the following rules :

1. SHIP(X, Y, Z, A) and Z = Smith -> Level(Y, Secret)
2. SHIP(X, Y, Z, A) and A = 10 -> Level(Y, TopSecret)
3. SHIP(X, Y, Z, A) -> Level(Together(Y, Z), Secret)
4. SHIP(X, Y, Z, A) and Release(Z, Unclassified) -> Level(Y, Secret)
5. SHIP(X, Y, Z, A) and Release(Y, Unclassified) -> Level(Z, Secret)
6. NOT(Level(X, Secret) or Level(X, TopSecret)) -> Level(X, Unclassified)

5. Design of the query processing module (8)

- The first rule is a content-based constraint, which classifies the name of a ship whose captain is Smith at the Secret level.
- Similarly, the second rule is a content-based constraint that classifies the name of a ship whose M# value is 10 at the TopSecret level.
- The third rule is an association-based constraint, which classifies the names of ships and captains, taken together, at the Secret level.
- Rules 4 and 5 are additional restrictions that are applied as a result of the context-based constraint specified in rule 3.
- Rule 6 states that the default classification level of a data is Unclassified.

5. Design of the query processing module (9)

- Suppose an Unclassified user queries the names of ships in the SHIP relation. This query is represented as follows:

  SHIP(X, Y, Z, D)

- A ship's name is classified at the Secret level if its captain is Smith or if its name has already been provided at the Unclassified level, and is classified at the TopSecret level if the value of M# is 10. We assume that the captains' names were not still provided to an Unclassified user, and the query is modified as follows:

- SHIP(X, Y, Z, D) and NOT(Z = Smith and D = 10)

- We note that because the query modification is made in real time, it will have an impact on the performance of the query processing algorithm.

5. Design of the query processing module (10)

Response processing

- Consider the following delivery constraints:
  - All names of the ships whose captain names have already been delivered to Unclassified users are Secret.
  - All names of captains whose ship names have already been delivered to Unclassified users are Secret.
- Suppose an Unclassified user asks for the names of ships for the first time.
  - Based on the other enforced constraints, we assume that only certain names are provided to the user.
- Then, the delivered ship names must be registered in the knowledge base.
- Subsequently, we assume that an Unclassified user (not necessarily the same) asks for the captains' names. These values (some or all) are then assembled in the response. Before the response is provided, the ships' names that have already been provided to the Unclassified user must be analyzed. Then the captains' names which correspond to ships' names that have already been delivered will be deleted from the response.

5. Design of the query processing module (11)

Example: Consider the following aggregate constraint:

- A collection of 10 or more tuples in the SHIP relationship is Secret.
- Suppose an Unclassified user queries tuples from SHIP. The answer is assembled and then analyzed to check if it has more than 10 tuples. If so, it is deleted.

- There are a number of issues associated with maintaining the delivered information.
- As more relevant information is inserted, the knowledge base can grow at a very fast pace.
- Therefore, efficient techniques for knowledge base processing need to be developed.
- This would also have an impact on the performance of query processing algorithms. A solution would be to include only certain information provided in the knowledge base. The rest of the information can be stored with audit data, which can then be used by the SSO for analysis.

6. Design of the update processing module (1)

Security policy

- MLS/DBMSs ensure that a level of security is assigned to data as it is inserted or modified.
- The level of security assigned to the data is generally assumed to be **the level of security of the user inserting the data**.
- A stronger and more dynamic approach in assigning security levels to data is by means of using the security constraints during update operations.
- The update processor's security policy is formulated based on the simple security property and a security policy provided by the base MLS/DBMS.

6. Design of the update processing module (2)

- This **policy** is the following:

1. All users are granted an authorization level relative to the security levels. A user can log in at any level that is dominated by his maximum authorization level. Subjects act on behalf of users at the log-in security level.
2. Objects are rows, tables, and databases, and each object is assigned a security level at creation.
3. A subject has read access to an object if the security level of the subject dominates the security level of the object.
4. A subject has write access to an object if the security level of the object dominates the subject's level.

- Claims 3 and 4 of the above policy are, in fact, the simple and the * properties of the Bell-La Padula policy.

6. Design of the update processing module (3)

Functionality of the update processing module

- The update processing module uses simple and content-dependent security constraints as a guide for determining the security level of the data that is being updated.
- The use of security constraints may protect against:
  - incorrect labeling of data as a result of connecting users to the wrong levels or importing from systems with different operating modes
  - database inconsistencies caused by data security labels in the database into which data is inserted.
- The security level of an update is determined by the update processing module.
- Simple and content-dependent security constraints associated with the relation being updated and with a security label higher than the user's login security level are found and analyzed for applicability.

6. Design of the update processing module (4)

- If multiple constraints apply, the security level is determined by the **constraint that specifies the highest classification level**.
- If no constraint applies, the update level is **the user's login level**.
- Therefore, the update processor does not determine the security level of data only from the security constraints, but uses these constraints to determine the level of input data. The following example illustrates the functionality of the update processor.
- We consider a database that consists of the relationship SHIP (ship_number, sname, captain, mission) and MISSION (mission_number). The content-based constraint that classifies all SHIP values named Josephine as Secret is expressed by:
- SHIP.sname = “Josephine” -> Secret

6. Design of the update processing module (5)

- A user with the **Confidential** security (log-in) level enters the following data:
  - INSERT INTO SHIP VALUES(“S123”, “James”, “Thomsen”, “MR2000”)
- The processor receives the insert command and retrieves the constraints associated with the SHIP relation that specify a higher level than the user’s
- The previous content-based constraint is found. Because the name entered is not "Josephine", the security constraint associated with the SHIP relation will not affect the classification level of the insert, and the update processor will determine the **insertion level to be that of the user, Confidential**.

6. Design of the update processing module (6)

- We assume that a user with the **Confidential** login security level enters the following command:
  - INSERT INTO SHIP VALUES(“S124”, “Josephine”, “Jane”, “MR3000”)
- The processor will find the constraint associated with the SHIP relationship. Since the name is Josephine, the processor will cause **the insertion level to be Secret**. If the user connects to the **TopSecret** security level, the processor will **insert at this level** (because it is higher than the one specified in the security constraint).
- The update operation works similarly to the insert operation. Suppose a user with the Confidential level launches the command:

  UPDATE SHIP

  SET name=”Josephine”

  WHERE captain = “Thomsen”;

- The processor will find the constraint and determine the update level to be Secret.

6. Design of the update processing module (7)

- In addition to describing the functionality of the update processor, the previous examples illustrate the potential signaling channels that exist when operating with the update processor.
- A signaling channel occurs when the actions of a high security user interfere with a low security user in a visible manner.
- Signaling channels occur when data is entered at a higher level than the user's and the user is trying to retrieve the data he entered, or when the update processor tries to insert data at a higher level, but it cannot because a tuple with that primary key already exists at this level.