

Chapter 6. Security in Database Systems

Databases are collections of structured permanent information. They contain the long-term data of an enterprise: customer lists, orders, manufacturing information, shipping data, etc. They represent a valuable asset of a company and are clearly an obvious target for attack. Two books dedicated to database security are [Fer81] and [Cas94].

In this chapter we survey the security problems of database systems and the methods for solving them. We start with a brief introduction discussing the main concepts and architecture of a DBMS, and then we discuss security in Relational systems in Section 6.4, and then security in Object-oriented and XML databases in Section 6.5. This discussion assumed a discretionary (DAC) access control model. We then discuss Multi-level databases which try to enforce the Bell-Lapadula model (see Chapter 2) in Section 6.6. In Section 6.7 we discuss the problem of security in statistical databases and the Inference problem. Next we discuss three relatively new issues: security and privacy for data-warehousing and data mining in Section 6.8, security problems for web databases in Section 6.9, especially the SQL injection problem, and database encryption in Section 6.10. We conclude with a discussion of security features of some of the leading DBMS products in Section 6.11.

6.1. Introduction: Architecture and major concepts of a DBMS

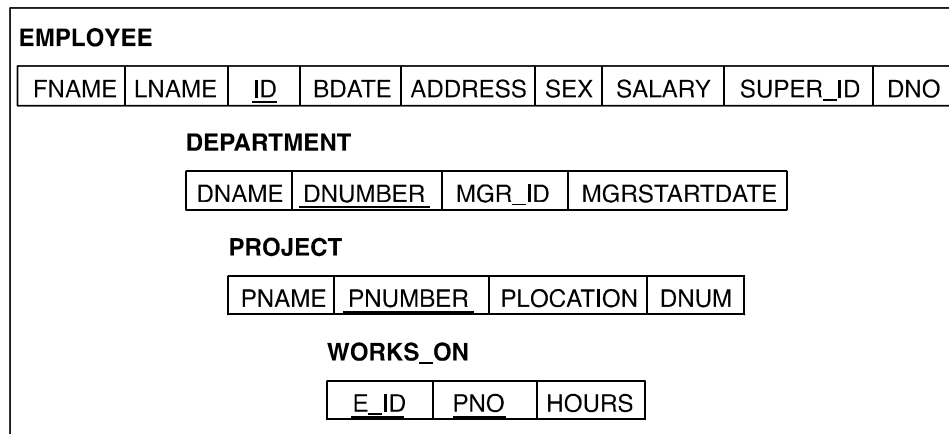
This section discusses the major concepts of a database and presents the general architecture of a typical DBMS. It then introduces the relational model and SQL briefly, and discusses the concept of views.

A *database management system* (DBMS) includes facilities for querying or retrieving information, for efficient search, for concurrency control, for transaction control, for security, and for recovery. A data description language (DDL) is used by administrators to create database entities and define aspects such as security or semantic integrity. A data manipulation language (DML) is used for retrieving and modifying information.

Relational databases store their data in the form of tables, where each entry (tuple) represents a record. The relational model uses several basic (relational algebra) operations to manipulate data: selection, projection, union, minus, Cartesian product, and join. There are also rules for integrity. The basic operations can be used to define views that present the users a subset of the database. Almost all relational databases use SQL as data manipulation language (SQL is an ANSI and ISO standard).

Object-oriented databases (OO DBMSs) use classes and objects as their units. Object-oriented programs can be mapped easily to this type of databases, while relational databases require rather complex mappings. Another possibility is the object-relational database that presents object-oriented user interfaces but stores its data in tables. Currently the most popular model for databases is the relational model; object-oriented databases have not become popular except for specialized applications.

Figures 6.1 describe an example database of a company, including entities such as Employees, Departments and Projects, and the relationship Works_On which describes the projects each employee is working on. Figure 6.2 presents its definition in SQL.



Figures 6.1: **Example - a company database**

CREATE TABLE EMPLOYEE

(FNAME	VARCHAR(15)	NOT NULL,
LNAME	VARCHAR(15)	NOT NULL,
ID	CHAR(9)	NOT NULL,
BDATE	DATE	
ADDRESS	VARCHAR(30),	
SEX	CHAR,	
SALARY	DECIMAL(10, 2),	
SUPER_ID	CHAR(9),	
DNO	INT	

PRIMARY KEY (ID),

FOREIGN KEY (SUPER_ID) REFERENCES EMPLOYEE (ID),

FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNUMBER));

CREATE TABLE DEPARTMENT

(DNAME	VARCHAR(15)	NOT NULL,
DNUMBER	INT	NOT NULL,
MGR_ID	CHAR(9)	NOT NULL,
MGRSTARTDATE	DATE,	

PRIMARY KEY (DNUMBER),

UNIQUE (DNAME),

FOREIGN KEY (MGR_ID) REFERENCES DEPARTMENT (ID));

CREATE TABLE PROJECT

(PNAME	VARCHAR(15)	NOT NULL,
PNUMBER	INT	NOT NULL,
PLOCATION	VARCHAR(15),	

PRIMARY KEY (PNUMBER),

UNIQUE (PNAME),

CREATE TABLE WORKS_ON

(E_ID	CHAR(9)	NOT NULL,
PNO	INT	NOT NULL,
HOURS	DECIMAL(3, 1)	NOT NULL,

PRIMARY KEY (E_ID, PNO),

FOREIGN KEY (E_ID) REFERENCES EMPLOYEE (ID)

FOREIGN KEY (PNO) REFERENCES PROJECT (PNUMBER));

Figure 6.2: **Example definition in SQL**

A standard architecture for databases uses three levels:

- Subschema's or user views present a subset of the data in different formats.
- A schema or conceptual model describes the relevant model for the institution.
- A storage schema describes access paths to search and retrieve data efficiently.

Mappings define the correspondence between schemas. Usually the DBMS runs on top of the OS, utilizing its services, in particular authentication and data storage. (Typically, the OS files contain the DBMS data.)

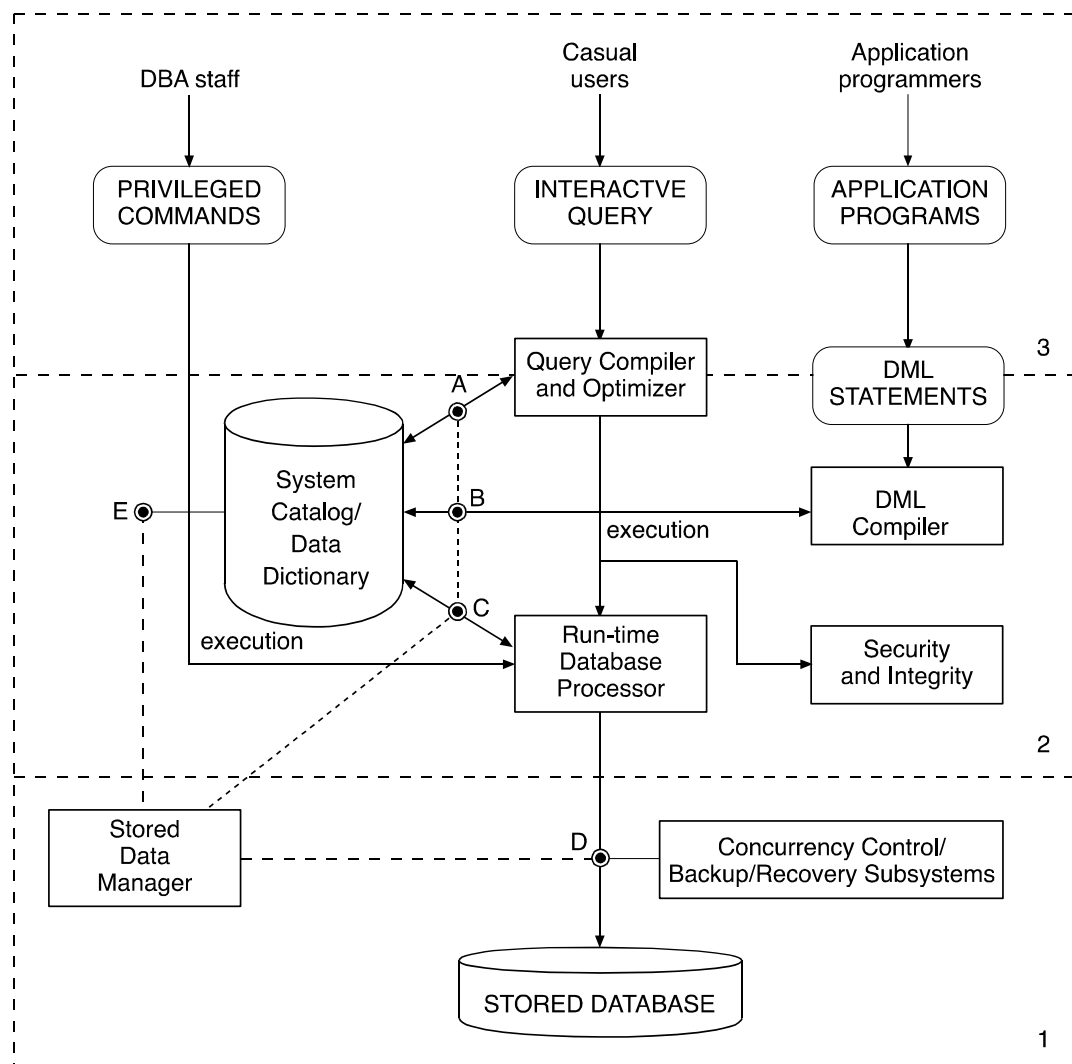


Figure 6.3: A typical DBMS architecture.

Let us present some more details on the various modules of the architecture in Figure 6.3

The Storage module.

This module (the lowest layer in the figure), is responsible for all the aspects which relate to the physical storage of the data on disk or in the operating system (OS) files. It maps the logical relations into physical files. It maintains the various indexes and access methods. It is responsible for interface with the OS and the buffer (cache) management. It contains the concurrency control component which enforces locking protocols and detects deadlocks during data access. It contains the Log & Recovery components which assure the atomicity and durability of transactions and provides tools for recovery in case of system failures.

The Query and DML module.

This module (layer 2 in the figure) is responsible for compiling, optimizing and executing queries and embedded DML (data manipulation language) programs. This module is responsible for storing and maintaining all the database (schema) definitions in the data dictionary. It is also responsible for parsing the SQL queries, and optimizing them using the information in the data dictionary (such as existence of indexes, selectivity of attributes, etc.).

This module also contains most of the security and integrity features which are our main interest here. Security is usually checked by the query compiler when a view is used, and the integrity is checked partly at compile time and partly at run-time (during Update), since some values which are being updated are known only at run-time.

The User-interface module.

This module (layer 3 in the figure) is responsible for the actual interface of the system to users to programs or to a host language. Standard protocols such as ODBC or JDBC may be used here.

For more details on DBMSs architecture and implementation, see any standard database book, e.g. [Ram2003].

As mentioned above, most of the security issues are at the higher layers of the system. In most cases, a DBMS will be implemented on top of an operating system and therefore will rely on its security features. Therefore, low level security issues such as: protection of memory buffers, or separation between various threads of control are delegated to the OS, and the DBMS is not concerned with them. In addition, most DBMSs rely on the OS for user authentication, although some products (e.g. Oracle) provide their own authentication methods.

Views and security

A view is a very important concept in Database systems. The view enables users (or groups of users) to access only a portion of the database, thus providing both convenience and security. A view also enables conversion of logical data types into language-dependent data types depending on the host language in which the view is embedded. In all SQL based systems, a view can be defined using the standard SQL syntax. Figure 6.4 presents two view definitions, the first provides access to names and hours worked for employees who are working on project named: "Program-X", and the second view provides access to count and total salary of employees by the department they work in.

Since all SQL syntax is available to the definer of a view, a view may be very complex. It's important to note though that a view is only a "window" on the database, and does not exist as a physical table. Thus the view has to be retrieved with every access, and therefore is always up-to-date.

SQL queries may use a view as any other table, which makes the definition of queries much more convenient and simpler. However, it is more efficient to perform a query on a view, then to phrase the query directly, since usually the view is stored after compilation and optimization, and therefore the query compilation phase is shortened. One important point, which should be mentioned here, is the issue of view update. Not all views can be updated using standard update queries, since they may lead to ambiguous values in the database. This is called the "view update problem" and it is outside the scope of this book. Most commercial (SQL based) systems put heavy restrictions on the type of views, which can be updated.

Views are very important to security. They allow the definition of access to only a portion of the database. Also, because of their flexible and powerful definition language, they allow content dependent authorization and avoid the use of security rules like in Ingress (see below). The authorization mechanism using views will be discussed in Section 6.4.

V1: CREATE VIEW	WORKS_ON1
AS SELECT	FNAME, LNAME, PNAME, HOURS
FROM	EMPLOYEE, PROJECT, WORKS_ON
WHERE	ID=EID AND PNO = PNUMBER AND PNAME = 'PROGRAM-X';
V2: CREATE VIEW	DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
AS SELECT	DNAME, COUNT (*), SUM (SALARY)
FROM	DEPARTMENT, EMPLOYEE
WHERE	DNUMBER = DNO
GROUP BY	DNAME;

Figure 6.4 : **Definition of views in SQL.**

Next we discuss the main security threats in a database system.

6.2 Security problems in Databases

In this section we discuss general security problems in database systems and the common policies and mechanisms to deal with them. We also give an overview of database authorization models

Database security risks

The risks to a database system are similar to those of other software systems, such as the network or the operating systems, except that since a database usually contains sensitive and private information, special requirements may be derived from this.

The first risk is the risk to the Integrity and Consistency of the database, which may not necessarily be the result of malicious access. We discussed Integrity problems in Chapter 1; let us repeat the main problems here:

1. Violation of integrity constraints. A database usually contains a set of integrity constraints such as: uniqueness constraints, foreign-key constraints and value constraints. If a transaction violates any of these constraints, the DBMS usually aborts the transaction, and rolls it back so that the database will remain consistent. Note that such an integrity violation may be caused by a single naive user with no harmful intentions.
2. Concurrency problems. A database may become inconsistent because of the concurrent execution of transactions. Usually such situation is protected by the DBMS by implementing some form of a Locking protocol (e.g. Two-phase locking). We will not detail this topic more here, since it is covered in great detail in all Database system books.
3. Recovery problems. A database may become inconsistent as a result of a system failure (hardware or software). Such a failure may leave the database inconsistent. The recovery procedure usually uses a *Log* which records all updates to restore the database into a consistent state. This can be done by either *rolling back* uncommitted transactions, or by *rolling forwards* committed but not yet flashed out transactions. Again, the details of this mechanism are outside the scope of this book.

In contrast with the integrity problems mentioned above, security problems are caused by acts of malicious users. Also, while for integrity, Read access is not a problem, it may be a problem for security! Following are some of the common security problems in databases.

1. Illegal access to information. This problem can occur either by wrong authorization rules or by a user bypassing the authentication process (see next item). A database usually contains large quantities of data shared by multiple users. It is important to regulate access to this data, authorize it correctly, and allow access only to users that are authorized to access it. The access control must be defined and enforced in various granularities, and administered correctly. The risks may be due to invalid or incomplete specification of authorization, or due to lack of control on the administration of authorization.
2. Bypassing authentication. Database access control relies heavily on the authentication of users, since authorization is usually specified (and enforced) by using users' unique identifiers (or roles). In Client/Server systems, the client machine must also be authenticated. However, generally the authentication problem is delegated to the Operating system and is usually not dealt with within the DBMS. Note that some systems when operating in a client/server architecture may do the authentication themselves (e.g. user/password scheme in ODBC/JDBC)

3. **Attacking data in transition.** Data, which is extracted from the database, and transferred to/from users, may pass through several layers. Its protection is done by the Operating and Network layers and is usually not the job of the DBMS. Some DBMS provide Encryption to minimize the risk of data in transition. Recently, organizations are starting to use outsourcing services to manage their database. Encryption in those cases becomes more important! [Hac02] (see also Section 6.10)
4. **Using Inference to find sensitive information.** Even valid access (e.g. to aggregate data), to a database may reveal undesirable or secret information by using Inference algorithms the inference problem refers to the ability to combine statistical queries to deduce individual private information. Such inference may cause serious privacy problems. A database system must be able to protect against such inferences, and we will discuss this in Section 6.7.
5. **SQL injection and other web-based attacks.** With the development and popularity of the web, many databases are now accessed by users via Browsers and web servers. This architecture lead to many vulnerabilities which derived from the fact that users are authenticated by the web server and may be unknown to the database. This is discussed further in Section 6.9.

As the second and third problems are usually solved by the Operating system, we will concentrate in this chapter on the first and fourth and fifth problems. We next review some of the security policies discussed in Chapter 2 of this book, and see how they apply to databases.

Security policies for Database systems

In chapter 2, we discussed the various security policies which are applicable to software systems. In this section, we discuss the policies which are commonly used in Database systems. The models derived from these policies and their implementation in various database systems will be detailed in the rest of this chapter. The various policies which are common in database systems can be classified according to the following categories:

- **Discretionary or mandatory,** most commercial system use the discretionary i.e. need to know policies and provide access to users to only the portion of the database they need to access. However, some vendors have developed Multi-level databases which support the military mandatory policies, where tuples or attributes may be classified at different levels. This type of databases will be discussed in section 6.6.
- **Open or Closed system.** In a closed system access is allowed only if explicitly authorized. In an open system access is allowed unless explicitly forbidden Most DBMSs use the closed system approach, where when a new object (relation, view) is created, by default only the creator has access to this object and must explicitly grant access to it to other users.
- **Centralized or Distributed,** A fundamental security choice is between centralized or decentralized control. With centralized control, a single authorizer (or group) controls all security aspects of the system. INGRES (see Section 6.3) is an example of a system that has adopted this type of policy. In a decentralized system different administrators control different portions of the database, normally following guidelines that apply to the whole database. This is achieved in SQL-based systems using the Grant/Revoke commands (see Section 6.4).
- **Ownership vs. Administration,** A related policy concerns the concepts of ownership and administration of databases in the enterprise. In operating systems, usually, the owner of

an object (e.g. a file) administers its security. This may not be always the case with databases. While the owner is allowed every possible type of access, the administrator possesses only the rights that control the data. For example, commands for tuning a database, can usually be issued by the administrator only. In a sensitive database it may even be that the administrator controlling access to it is not able to read the data because it is encrypted. This is typical if the database is “out-sourced” to a third (not fully trusted) party [Hac02].

- Granularity of access control, Databases can support access control in various granularity levels, such as: the full database, a full relation, part of a relation, including only specific attributes or rows. The last feature is usually achieved by using views. In most cases, access to a large granule implies access to its components; however, exceptions may be defined using Negative authorization, as is the case for example in Object-oriented databases (see Section 6.5).
- Name or Content or Context Dependent, This tie to the granularity discussion above. Access may be defined based on the Name of the object (e.g. Relation), which include its complete content, or based on some specific content predicates (using Views), or based on the context or history. The last type is usually not implemented in commercial DBMSs. It is used mainly to protect against invalid information flows. Therefore it is usually included in the Multi-level DBMSs, or in statistical databases (see Section 6.6 and 6.7).
- Individuals or Role-based. The notion of Groups or Roles and their advantages over defining access based on individual users, was discussed in Chapter 2. It has recently become popular in DBMSs; this is especially convenient if there are thousand of users accessing the same database! Managing access by individual users becomes difficult. As a result, Roles have become part of the new standard SQL-99. They will be discussed in Section 6.4.

The richness of security policies discussed above and the different security risks possible make the handling of security in databases a complex issue. Further complexity arises because often the security functionality is not clearly separated in the system. For example in many organizations which use a relational database, there is an external layer (e.g. an "enterprise" software system) that handles most of the security definitions, and the database security is completely redundant! Another example may be a web-server that checks authorizations itself and when it accesses the database it basically access it as a "super-user" or administrator. This type of architecture clearly "defeats the purpose" of database security! (see also Section 6.9).

In the rest of this chapter we will assume that security is indeed implemented in the database component and is not treated as a "degenerated" feature.

6.3 Overview of Database authorization models

The models for commercial database systems are usually based on the access matrix model. There have been some database systems using the multilevel model but have not become popular outside of specific application environments (see Section 6.6). An authorization rule has the form: (s,o,a,p), where s is a user or user group, o is a named data item, a is an access type (retrieve, modify, add a record, delete a record), and p is a content-dependent predicate [Fer75]. In relational databases rules define access to tables or views, rows, or columns. In OO DBMSs rules define access to classes and class methods. This rules model describes the models used by most commercial database systems, including Oracle, MS-SQL DB2, and others. These systems usually have the concept of owner (which may violate the principle of separation of duty), and/or the concept of security administrator; a poor administration system can compromise security [Fer81].

In most relational systems users can grant their privileges to other users. Authorization rules are defined using SQL statements and these rules can be enforced in two basic ways (see Section 6.4 for details):

- Performing the AND of the request and the rules, as was done in INGRES [Ston76].
- Predefining views incorporating access constraints and giving users access to specific views, as is done in System R, DB2, and others [Grif76].

One can also assign the database rights to roles; that is, we can build an RBAC model using authorization rules. This is easier to do in a DBMS because the rules are normally written by administrators and not by users as in OS file systems.

A useful concept to define authorization rules is *implied authorization* [Fer75a]. [Fer75a] introduced the concept of types of rules depending on their implementation: stored, effective (used to validate a request, maybe implied by other rules), and defined (specified by the authorizer). Implied authorization was used in Orion [Rab91], in several OO DBMS authorization models [Fer93, Fer94], and in Windows 2000 object access. The idea is that authorization in components or in subclasses can be deduced from higher-level authorizations and need not be explicitly written. When there are predicates the implications become more complex [Lar90]. A similar idea is used in XML based systems [Dam02].

There have been proposals to give the DBMS a larger system importance by making it a peer of the OS and sharing with the OS a common I/O subsystem [Spo84]. In this proposal security functions are concentrated in a separate Security Subsystem common to the OS and the DBMS. The I/O and the Security subsystems would execute in a separate processor. This approach could be useful for database servers and could enhance security for the DBMS. Another attempt in this direction was database kernels [Downs77].

6.4 Security in Relational and SQL-based systems

The concern for security in relational databases was very much part of its early developers. Historically, both of the earliest research oriented relational systems, Ingress in U. of California Berkeley [Ston76], and System R in IBM Research San Jose [Grif76] had components, which addressed the security issue, especially in terms of access control. Since both of these research type systems have influenced the development of commercial systems considerably, and especially the development of the SQL standards (SQL92,SQL99), we will discuss their approach in some detail. Then we'll discuss the security aspects of SQL92 and SQL99.

6.4.1 Security in the INGRES system – Query modification

The INGRES relational DBMS introduced a way to implement fine-grained access control, using a technique called *query modification*. INGRES provides two modes of interaction with the database:

1. a high-level query language, QUEL, and
2. QUEL statements embedded in programs written in the "C" language.

A QUEL *interaction* includes a RANGE statement and one or more statements of the form:

Command (target list) [WHERE qualification]

For example, consider the relation EMPLOYEE, with attributes NAME, SAL, MGR, and DEPT. A query to retrieve the names and salaries of all employees whose manager is JONES would be:

*(Q1) RANGE OF E IS EMPLOYEE
RETRIEVE (E. NAME, E. SAL) WHERE E. MGR = 'JONES'*

Where E is a tuple variable that ranges over the EMPLOYEE relation. Query Q1 retrieves all tuples of EMPLOYEE that satisfy the qualification (predicate).

In INGRES a centralized DBA is responsible for the creation and destruction of all shared relations and also for all authorization. No mechanism for the delegation of this responsibility exists. This centralized approach was chosen in order to simplify database administration and the system design. The specification of authorization is done by **Access Rules**. Access rules are written by the DBA in QUEL. For example,

(Rule 1)

**RANGE OF E IS EMPLOYEE
PERMIT E TO JONES FOR
RETRIEVE (E. NAME, E. SAL, E. MGR)
WHERE E. DEPT = 'D1'**

Meaning: permit to Jones Read access to names, salaries and managers for employees working in department 'D1'.

Access rules may also be specified for maintenance operations: APPEND, DELETE, and REPLACE. The rules are stored in a directory relation named PROTECTION. The predicates are not stored in their original form, but are converted into an internal tree-structured form.

Let us see how query Q1, issued by user Jones, is affected by Access Rule 1. The rule predicate (E. DEPT = 'D1') is conjoined (ANDed) with the predicate of the original query (E. MGR = 'JONES'). This is done by appending the stored tree representing the access-rule predicate to the tree representing the request predicate. Figure 6.5 shows the tree for query Q1, and Figure 6.6 the tree for the modified query. The left-hand side of each tree specifies the attributes to be retrieved and the right-hand side the qualifications.

Now consider the following set of rules about Jones's access to EMPLOYEE.

(Rule 2)

RANGE OF E IS EMPLOYEE

PERMIT E TO JONES FOR

RETRIEVE (E. NAME, E. DEPT, E. MGR)

WHERE E. DEPT = 'D1'

(Rule 3)

RANGE OF E IS EMPLOYEE

PERMIT E TO JONES FOR

RETRIEVE (E. NAME, E. SAL)

WHERE E. MGR = 'JONES'

(Rule 4)

PERMIT E TO JONES FOR RETRIEVE (E. SAL)

WHERE E. SAL < 100000

Which rules would be used on the following requests?

(Q2) RETRIEVE (E. NAME, E. SAL)

Before answering this question, we describe a simplified version of the INGRES *access-control algorithm* for any RETRIEVE request R.

1. Find all attributes in the target list or in the request predicate. This set is called S.
2. Find the set T of access rules with RETRIEVE as the access type and with a target list containing all attributes in S. If there is no such rule, deny the request.
3. Ignore any access rules in T whose target lists contain the target lists of other rules in T. Call the resulting set T'. That is, find the smallest target lists that cover the request R. Call the predicates of the remaining rules P1,..., Pn.
4. Replace the request predicate, Pr, by Pr AND (P1 OR P2 OR ...OR Pn).

This algorithm is depicted in Figures 6.5 and 6.6

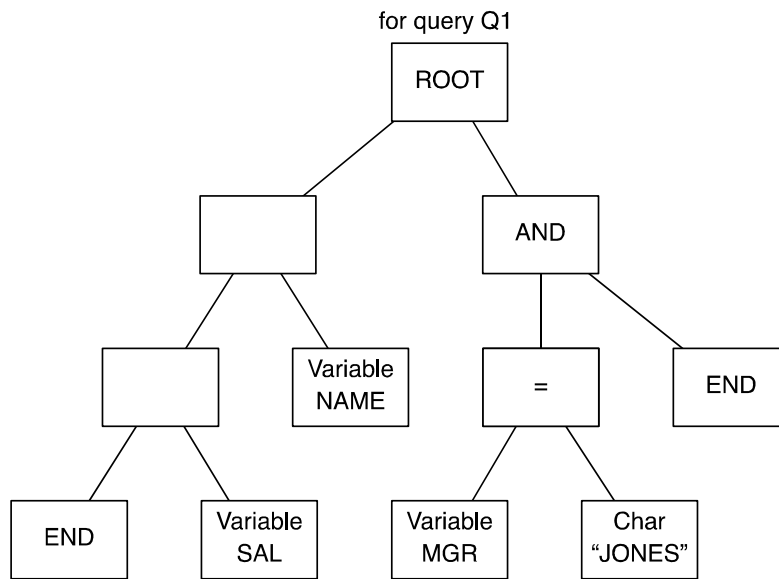


Figure 6.5: **Query tree**

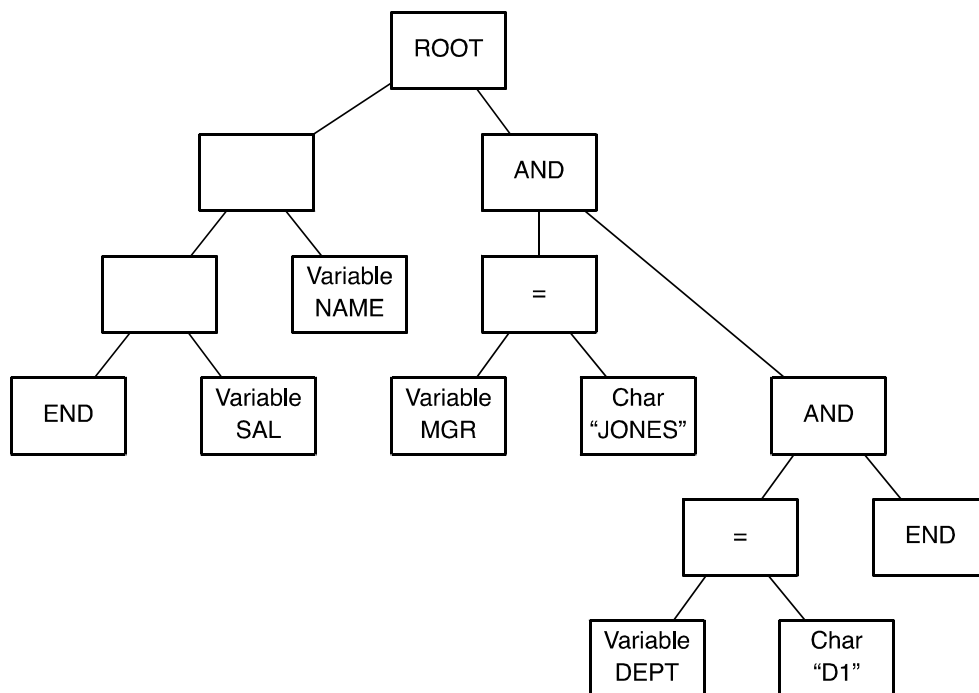


Figure 6.6: **Modified query tree**

The resulting query can now be executed normally.

What this algorithm means is that all rules relevant to the request are first located. It is then assumed that rules are well nested; that is, Rule A is more restrictive than Rule B if Rule B author-

izes access to a subset of the attributes in Rule A. Only the most restrictive rules are retrieved for the query modification.

For request Q2 the set S is {NAME, SAL}, so only Rule 3 contains all attributes of S. Jones is allowed to see NAME and SAL only for her employees, even though she can see NAME for all employees in department D1. The modified query is:

(Q2') *RETRIEVE (E. NAME, E. SAL) WHERE E. MGR = 'JONES'*

Now consider another request:

(Q3) *RETRIEVE (E. NAME).*

Now both rules 2 and 3 apply, but neither rule contains the other. Therefore the modified query is:

(Q3') *RETRIEVE (E. NAME) WHERE E. DEPT = 'D1'*
OR E. MGR = 'JONES'

Finally, consider the request

(Q4) *RETRIEVE (E. SAL)*

Here {T} = {Rule 3, Rule 4}, but {T'} = {Rule 4}, since Rule 3 includes more attributes than Rule 4. So Rule 4 is applied and the modified request is:

RETRIEVE (E. SAL) WHERE E. SAL < 100000.

Note, that although INGRES protection mechanism was not adopted by developers of the SQL standard, but rather System R's one (see next section), recently some commercial DBMS like Oracle 9i adopted a similar scheme called "virtual private databases" (see Section 6.11), and thus INGRES's scheme has more than just historical importance.

6.4.2 Security in System R - Views

As mentioned above, System R security approach formed the basis for the current SQL authorization approach. System R designers [Grif76] have addressed two major disadvantages of the Ingress approach:

1. **Centralization.** Ingress policy is completely centralized. All authorization rules are stored in a single location and maintained by a single administrator. This creates a bottleneck in administration.
2. **Rule semantics.** Rules were quite complex, and the interaction between rules as evident by the algorithm described in the previous section, is not easily understood by both administrators and users.

The new ideas of System R addressed both issues. The first issue was addressed by the concept of *Delegation*. An administrative policy was defined where users can delegate access administration rights to other users and can also revoke this delegation at a later time.

The second issue was addressed by the concept of a View which we discussed in Section 6.1. Instead of specifying complex rules, one can specify any complex view she likes, and authorize users to access such views. The queries applied to such views are "modified" automatically by

the definition of the view, and thus Ingress "query modification" mechanism is automatically applied by the query compiler. As we'll see in the next section, these two ideas become the cornerstone of the later SQL authorization systems.

Specifying authorization

In System R, the authorization was defined over two types of objects: Tables (Relations), and Views. In both cases, the creator of the object received all rights for the object, and she could grant these rights to other users, and also revoke them at a later time. The right could be delegated further if it was received with the GRANT option. The rights supported by System R were:

READ: the ability to use this relation in a query. This ability permits a user to read tuples from the relation, to define views based on the relation, etc.

INSERT: the ability to insert new rows (tuples) into the table.

DELETE: the ability to delete rows from the table.

UPDATE: the ability to modify existing data in the table. This ability may optionally be restricted to a subset of the columns of the table.

DROP: the ability to delete the entire table from the system.

The GRANT command of System R had the following form:

GRANT (privilege-list or ALL RIGHTS) ON (table) TO (user-list) [WITH GRANT OPTION]

Where the GRANT option provides the delegation capability.

The format of the REVOKE command was:

REVOKE (privilege-list or ALL RIGHTS) ON (table) FROM (user-list) where privileges may represent rights such as Read or Update or Delete.

The Grant command was pretty intuitive. The user receiving the grant is now able to access the objects specified in the grant, with the privileges specified in the Grant. The users which receive their rights with the GRANT option may grant these rights to other users, but these rights may be revoked later by users along the grant chain (see below).

The interesting problem is the problem of *Revocation* of rights. We next discuss the problem of revocation with respect to a given table. Let the sequence of grants of a specific privilege on a given table by any user before any REVOKE command be represented by:

G1, G2,...Gn

Where each Gi is the grant of a single privilege. (We represent grants or revocations of several privileges as a sequence of individual grants or revocations.) If $i < j$, then grant Gi occurred at an earlier time than did Gj. Now suppose that grant Gi is revoked. The sequence becomes

G1, G2,...Gi - 1, Gi + 1,... Gn

We formally define the semantics of the revocation of G_i to be as if G_i had never occurred. This implies that the state of the authorization tables after the above sequence of grants should be identical to their state as if the grant of the revoked privilege was never made!

We shall explain this concept in more detail using an example. Suppose we have a table of employees called: EMP, Figure 6.7 shows a sequence of GRANTS on this table, while Figure 6.8a shows the same sequence in a form of a graph, where an arrow from A to B, means that A granted some rights on EMP to B. Figure 6.8b shows the graph without the GRANT from B.

A GRANT sequence
A: GRANT READ ON EMP TO B WITH GRANT OPTION
A: GRANT READ ON EMP TO C WITH GRANT OPTION
B: GRANT READ ON EMP TO X
C: GRANT READ ON EMP TO X

Figure 6.7: **Grant sequences**

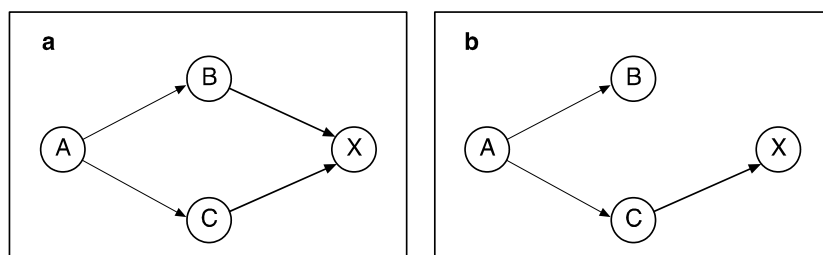


Figure 6.8: **Grant sequences and the Grants graph**

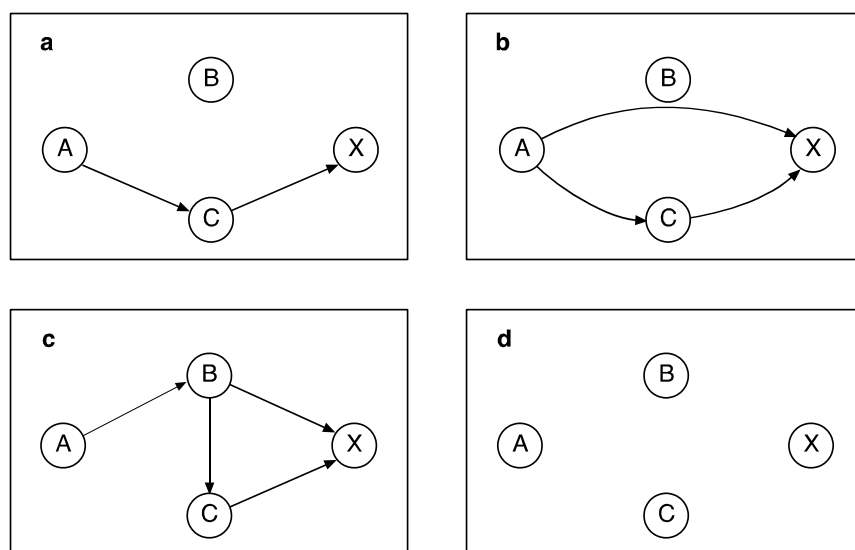


Figure 6.9: **Recursive revocation options**

Now, given the graph in Figure 6.8a, and a revoke of all rights of B by A, there are two options:

1. A full recursive Revoke, i.e. A revokes from B, and B revokes from X. This is shown in Figure 6.9a. Note that X still maintains the right it received from C.
2. After the revoke, X is denoted as it has received the right from A and not from B. This is shown in Figure 6.9b. This is called a non-recursive revoke (not implemented in SQL).

Now, assume that the sequence of commands is slightly modified as shown in Figure 6.9c, and then we expect a different result. Now, if A revokes its rights from B, we expect X to lose his rights entirely, as shown in Figure 6.9d, since he received them from B directly or via C. In fact, system R approach is that a revoke should result with a graph, which is valid with all commands without the command that was revoked! In order to implement that, System R developed a time-stamp based algorithm shown in Figure 6.10 (see [Grif76]). The algorithm is based on a table called SYSAUTH, an example of it is shown in figure 6.11a. This table represents for each grant command the following information: who received the grant - USERID, who granted the grant - GRANTOR, and for each right at what time it was granted. (in this example all rights are granted with the GRANT OPTION)

```

REVOKE: procedure (grantee, privilege, table, grantor);
      comment turn off the grantee's authorization for (privilege)
      obtained from (grantor);
      set (privilege) = 0 in the (grantee, table, grantor) tuple in SYSAUTH;
      comment find the minimum timestamp for the grantee's remaining
      grantable (privilege) on (table);
      m ← current timestamp;
      for each grantor u such that (grantee, privilege, table, u, grantor)
      is in SYSAUTH do
        if privilege ≠ 0 and privilege < m then m ← privilege
        comment revoke grantee's grants of (privilege) on (table) which were
        made before time m;
        for each user u such that (u, privilege, table, grantee)
        is in SYSAUTH do
          if privilege < m then REVOKE (u, privilege, table, grantee);
      return
end REVOKE

```

Figure 6.10: The Revocation algorithm

USERID	TABLE	GRANTOR	READ	INSERT	DELETE
X	EMPLOYEE	A	15	15	0
X	EMPLOYEE	B	20	0	20
Y	EMPLOYEE	Y	25	25	25
X	EMPLOYEE	C	30	0	30

Figure 6.11a: an example for table SYSAUTH

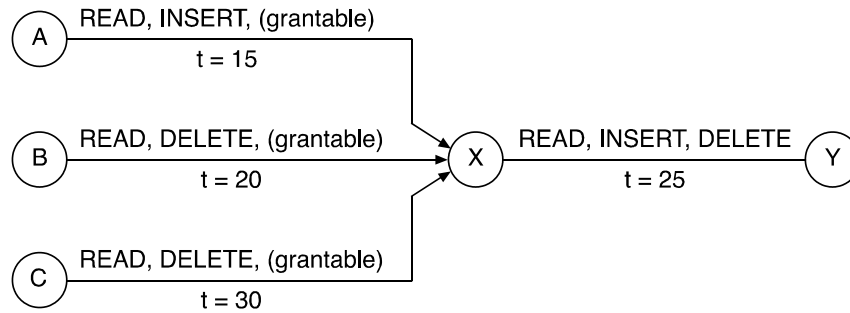


Figure 6.11b: **SYSAUTH** as a graph

USERID	TABLE	GRANTOR	READ	INSERT	DELETE
X	EMPLOYEE	A	15	15	0
X	EMPLOYEE	B	—	—	—
Y	EMPLOYEE	X	25	25	—
X	EMPLOYEE	C	30	0	30

Figure 6.11c: **SYSAUTH** after revocation

The algorithm first deletes the row for the revoked grant. It then searches in the table for the same right, what is the minimal time it was granted. If that time is *later* then any of the grants for this right, given by the USERID to other users, then those rights must be revoked using the same recursive procedure.

Consider the graph of grants shown in Figure 6.11b. Assume that at time $t=35$, B issues the command:

REVOKE ALL RIGHTS ON EMPLOYEE FROM X

As a result, the second row in SYSAUTH is deleted. Next we look at all rights received by X.

The minimal for READ is 15. The minimal for INSERT is 15. The minimal for DELETE is 30.

Therefore the DELETE right granted to Y by X at time = 25, should be revoked! The end result is the table shown in Figure 6.11c.

System R researchers also considered the effects of Grants and Revokes on views. When a user creates a base relation, he is fully and solely authorized to perform actions on that relation. Similarly, when a user defines a view he is solely authorized to perform actions upon it, except for two cases: (1) View semantics: certain operations may not be performed on any view; for instance, statistical views are not updateable. (2) The definer's authorization on the underlying tables: clearly a user with read-only access to the EMPLOYEE table should have read-only access to any view which he defines on top of it. System R restricted a user's authorization on a view to be only those privileges held on the underlying table(s). If the view involves more than one underlying table, the user's privileges are constrained by the intersection of the privileges,

which he holds on the underlying tables. Furthermore a privilege P on the defined view is grantable if and only if the definer holds a grantable privilege P on all underlying tables.

To introduce the view mechanism into the formal model of the semantics of revocation, consider a sequence of grants and view definitions, followed by a revocation or a drop. The semantics of this sequence is defined to be the same as if the corresponding grant or view definition had never been made. A revocation or a drop affects both grants and view definitions. All grants of dropped views are automatically revoked. All view definitions based on dropped views, or on tables, for which the definer has no remaining privileges, are deleted. The exact algorithm is given in [Grif76].

6.4.3 Authorization in SQL

As commercial Relational DBMSs started to come out in the beginning of the 80s, they started to adopt System R approach to many features, including Query languages and Security. This therefore influenced heavily the development of the SQL standard. In fact, SQL92, which is still the major standard for SQL, is very similar both in spirit and in syntax to System R. Next we explain the main features for security in SQL, and discuss the differences with system R.

As in system R, the basic units for authorization are either a table or a view. If one wants to grant access to part of a table (some of the columns, or some of the rows), or if one like to support *content-based* protection, one must do it by definition of an appropriate *view*. Also, as in System R, the administration of authorization is not central, and the creator of a table (or a view) can grant access with a Grant option to other users, thus delegating the right to define authorization distributively.

The syntax of the GRANT and REVOKE commands is very similar to that of System R and is the following:

GRANT privileges on Object to users [with GRANT option]

REVOKE [GRANT option for] Privileges ON Object from Users {RESTRICT | CASCADE}

Next we explain some of the differences with System R:

Privileges

The first four privileges are similar to those of System R:

Select - the right to read all columns of the object, including columns added later by an SQL Alter Table command.

Insert - the right to insert new values (tuples) into all columns of the object. If the object is a View, restrictions apply on what can be inserted, but this is dependent on the view definition (see the discussion of the "view update problem" earlier...) and not on the authorization. Insert may be specified with a "column-name" in which it applies only to the specified column (s). Otherwise, it applies to all columns, including columns added later by the Alter command.

Update - the right to update all columns of the object with options similar to those of Insert.

Delete - the right to delete rows from an Object. In addition, there are two new privileges:

Usage - the right to use the current definition in other definitions.

Reference - the right to use the object in specifying Integrity constraints of other objects.

Again, a specific column may be specified. We'll discuss this specific privilege in more details later.

The following example demonstrates the use of various privileges in a GRANT command. In order to perform this SQL Insert, one needs the privilege Insert on Dept (at least on Name), and at least the SELECT privilege on DeptName in StudDept, and on Name in Dept.

```
INSERT INTO Dept (name)
SELECT DISTINCT DeptName
FROM StudDept
WHERE DeptName NOT IN
  (SELECT name
   FROM Dept);
```

Users and Authorization Ids

While in System R, the basic subject entity is a User, in SQL there is the concept of an Authorization-ID, which may be a user, a group of users (or a Role), the Id Public or a user executing a specific module. This is especially oriented towards a Client/Server architecture, where the Client is identified to the DBMS server as an Authorization-Id.

One useful statement provided by SQL is the command:

AUTHORIZATION user-id

this command attached to a module will cause the module to be executed with the rights of User-id (similar to SETUID in UNIX). The Authorization-Id resulted will be: User-Id...

The implementation of Revoke

SQL developers accepted in principle the concept of *Recursive Revocation* of System R. However, they made several minor modifications. First there is the optional RESTRICT/CASCADE. If the Revoke causes recursive revocation then if RESTRICT is specified (default), then the Revoke is rejected. On the other hand, if CASCADE is specified, a recursive Revoke is performed.

Second, when a recursive Revoke is performed, it does not behave exactly according to timestamps. Take example 6.9c, if before the command Revoke, A would execute the following command: GRANT READ to C with GRANT OPTION,

in System R it would not make any difference, since the timestamp of this command is *after* the Grant from C to X. In SQL92, on the other hand, it makes a lot of difference! In that case, the

right Read will not be revoked from X, since the fact that C received it (even after X received it) makes the grant of C to X legitimate. This is a major difference between the two implementations!

The distinction can be defined precisely using the concept of authorization graph.

In SQL92, if a path exists from a "system" node to the node in question, after the revoked edges were removed, then the corresponding right should not be removed. The Revoke algorithm itself is similar to the one in Figure 6.10, but instead of using timestamps, it checks to see if a path exists to the "system" node, and only if such a path does not exist, it deletes the granted right (recursively).

Authorization of Views

The privileges held by a creator of a view may change over time if the creator's rights on the underlying tables change. First remember that in order to create a view a user must have SELECT right on all the underlying tables and if he wants to define foreign keys he must have also a REFERENCE right (see below). A user may grant others access to the view only if he received the access with GRANT option on all tables on which the view is defined. If the creator loses a privilege held with the GRANT option, users who were given that privilege on the view lose it as well. On the other hand, if at a later time, a user that created a view is granted additional rights on the Base tables, these rights are automatically granted to the user on the view, however, they are not automatically granted to other views that are defined on this particular view. Note that not all commercial DBMSs support authorization on views this way.

Integrity constraints and Reference privilege

In SQL it is possible to define integrity constraints between tables. The most common one is the "foreign Key constraint". The following example demonstrates the definition of an integrity constraint on table X which is based on table Y:

```
CREATE TABLE SNEAKY (Maxgrade INTEGER, CHECK (Maxgrade <=
(SELECT MAX (Grade FROM STUD-GRADE)))
```

First we should realize that such a definition requires that the user given access to SNEAKY should have a Select right on STUD-GRADE, since if not, he would be able to guess values of Grade, by just inserting values into SNEAKY. In the example above, by repeatedly entering increasing values, one can guess the maximum value in STUD-GRADE.

However, the SELECT right may not be sufficient. If the integrity constraint is a *foreign key* constraint, then by inserting values into Y, one is restricting the ability of the creator of X to do manipulations on X, e.g. deleting rows. This may not be desirable and that's the reason why the right REFERENCE was created! The creator of X grants the reference right to B purposely to allow him to define restricting constraints on X.

As another example assume Bob, creator of STUDENT, gives Bill the SELECT right and allows him to define a new table with a foreign key constraint.

```

CREATE TABLE STUD-GRADE1 (sname CHAR (10) NOTNULL,
sid    INTEGER,
day    DATE,
PRIMARY KEY (sid, day),
UNIQUE (sname),
FOREIGN KEY (Sid) REFERENCES STUDENT)

```

Now Bill can restrict the creator of STUDENT by entering values into STUD-GRADE1! To avoid this undesirable situation, Bob gives BILL an explicit right to define constraints on STUDENT called: REFERENCE. In this case, Bob will grant the right REFERENCE to Bill in addition to the SELECT privilege and therefore, Bob is aware and agrees that Bill actions may restrict him...

To summarize this section we should mention that several DBMS products have additional security features. For example, ORACLE has also the Execute privilege, which enables limiting executing some SQL procedures on the database. It also has a set of special DBA rights. (see also Section 9).

Role-based Model in Databases and in SQL

The Role-based model was discussed in detail in chapter 2. Its main advantages are that it's much easier to specify the security policies of the organization, without specifying hundreds or thousands of authorizations for individual users. In particular the four Role based models suggested by [San96] were mentioned:

Basic RBAC

RBAC with hierarchies

RBAC with constraints (e.g. Separation of duties)

RBAC with both hierarchies and constraints

In SQL99, there are several features which relate to the Role-based model. There are commands for defining new roles, e.g.

```

CREATE ROLE EDITOR
CREATE ROLE AUDITOR_GENERAL

```

Hierarchies may be defined by using a command such as:

```

GRANT AUDITOR TO AUDITOR_GENERAL

```

Permissions may be assigned to Roles by a command such as:

```

GRANT INSERT ON TABLE BUDGET TO AUDITOR

```

And roles may be assigned to users or to programs by a GRANT command and a specific role may be assigned to a user at run-time by a command such as:

SET ROLE AUDITOR TO 'JOHN'

All the above are very useful additions to SQL99. However, the full power of the RBAC model including various constraints is not yet part of the standard.

Generalizations of SQL

Recently, there appear several papers suggesting generalization of the SQL92 and SQL99 authorization schemes, see e.g. [Bert99, Bark01]. In [Bert99], the authors propose several generalizations to the current SQL99 standard:

1. They define precisely the notion of non-recursive revocation. All descendant of recursive revocations are not deleted, and instead are registered as if the revoker has granted them. Note, that this raises the issue of Responsibility (who is responsible for a Grant!), but the authors insist that such option should be defined.
2. They add a new privilege called NEGATIVE right. By using negative rights, the authors can define exceptions to groups or Roles (see next section). They also show the difference between a negative right and a revoke, since the first one, when deleted restores the past positive rights, where this is not true for a revoke!
3. They also generalize concepts of authorization on views and provide precise algorithms to deal with views authorization and revocation.

[Bark01] generalization to SQL99 suggests to add constraints to a GRANT, e.g. Grant only if some constraint is satisfied. This reminds of the Role-based constraints model discussed in Chapter 2.

6.5 Security in Object-oriented and XML Databases

The Object-oriented approach has become the most important approach in programming languages and software design today. In the late 80s and early 90s, the object-oriented approach was advocated by database researchers and developers as a way to overcome the weaknesses of the Relational model. Several object-oriented DBMS systems were developed and at the time the thought was that they soon will replace the Relational model. This, however, did not happen (mainly because of market reasons and not scientific or technical reasons), and the Relational systems remain in control in the 90ths and later. Nevertheless, it is very important and relevant to study security in object-oriented databases for two reasons:

1. Instead of developing pure OO databases, most DBMS vendors as well as the SQL standard developers have developed the *Object-Relational* systems, which are basically relational systems with object-oriented features; therefore, security mechanisms developed for OO databases are relevant also to object-relational systems.
2. There is recently tremendous interest in XML which is basically hierarchical representation of data for storing and exchanging data for the Internet. Several studies

were recently made on XML security, and not surprisingly they follow quite closely the ideas of OO database security. We will discuss them briefly in Section 6.5.2.

We start this section with a brief overview of the OODB ideas, and follow it with a description of the two major models of security in OO databases, the one by Rabiti, Bettino, Kim and Whelk [Rab91] and the one by Fernandez, Gudes and Song [Fer94]. We then discuss briefly the XML database model and its security aspects

6.5.1 Security in Object-oriented Databases

As mentioned, the relational model has several disadvantages, especially in its modeling power. It's very difficult to model in a relational system the complex objects and relationships required in many applications especially in CAD and Engineering applications. The main new ideas in OO databases are:

- The main entity is an object and not a relation. Objects may be composed of attributes or methods. In contrast to the relational model, attributes may not be atomic but may contain sets (or lists) of other attributes, including references to other objects. Thus an object may have quite a complex structure. In addition, also in contrast to relational, an object may have a variable structure and not all attributes need to be present.
- Objects can be grouped into classes. Classes may be related by *Semantic relationships*. Two of the most important relationships are: Generalization/specialization - basically the Inheritance relationship, both attributes and methods may be inherited, but also can be re-defined, using the well-known techniques of Overriding and Overloading. The other relationship is Aggregation, which allows the construction of complex objects (objects composed of other objects). Figure 6.12 and 6.13 show examples of the two types of relationships (also called hierarchies).
- Each object has a unique system-wide identity which is independent of any user-oriented key (not like primary key in relational).
- Objects may have multiple versions or multiple representations. This is particularly useful in engineering applications.
- The *Interface* to an OODBMS may be a special query language such as OQL, or an object-oriented programming language such as: C++. Such an interface enables a program to manipulate objects within the program directly from the database without the need to convert them first by a language such as SQL. This is a major advantage of OO databases overcoming the well known "impedance gap", between languages and databases.

As we shall see below most security models deal with the impact of complex objects and semantic relationships on security.

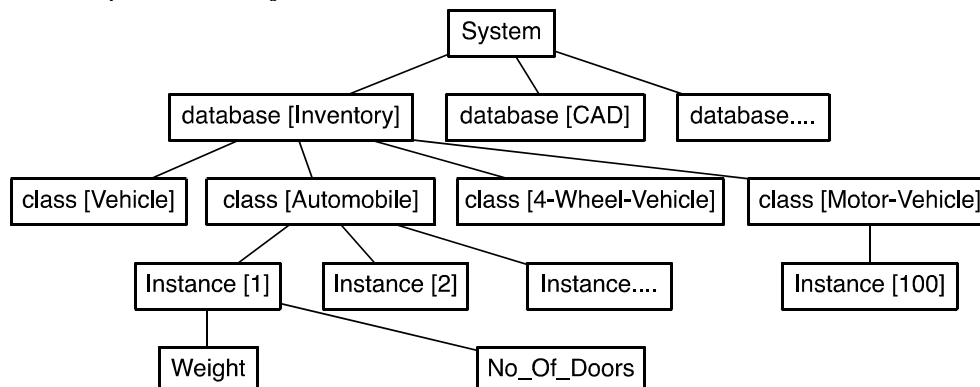


Figure 6.12: **Example for an Aggregation hierarchy (relationship)**

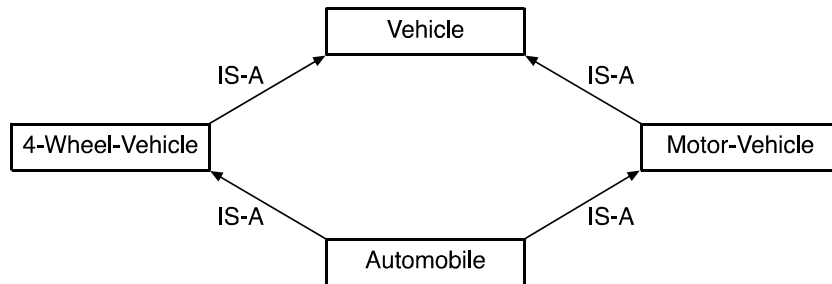


Figure 6.13: **Example for a Generalization hierarchy (relationship)**

Rabitti et. al. Model [Rab91]

We will use Figure 6.12, which demonstrates a typical aggregation hierarchy. The first concept presented by [Rab91] is that of **Explicit vs. Implicit** authorization which was introduced earlier by Fernandez [Fern81]. The idea is that authorization may be specified for an entity explicitly, and as a result other entities lower in the hierarchy inherit the authorization. For example, in Figure 6.12, the explicit authorization specified for database "inventory" is inherited by all classes which are descendants of "inventory", and by all instances of each class and by each attribute/value pair of each instance. Similar explicit/implicit authorizations may be defined on other hierarchies.

The second concept defines **Strong vs. Weak** authorization. A strong authorization implies that it and all implicit authorizations inherited from it, cannot be overridden, while a weak authorization (explicit or implicit) may be overridden by a strong authorization. This is very useful for defining *Exceptions*, as we'll see below.

The third concept is that of **Positive vs. Negative** authorization. A positive authorization allows access, while a negative authorization denies access. Since we usually deal with "closed" systems, the negative authorization is mostly useful for defining exceptions using strong negative to override weak positive authorization.

Figure 6.14 shows all the above combinations: the weak positive authorization on database "inventory" is overridden by the weak negative on class automobile, which is overridden by a strong positive authorization at instance[1], which is inherited as implicit positive authorization by attribute weight.

In Figure 6.13 we see an example of a Generalization hierarchy. In this type of hierarchy it's possible to define a weak positive authorization on the class *Motor-vehicle* and a strong negative authorization on the class *automobile*. One also has to deal with the issue of multiple inheritance, see [Rab91] for details.

The algorithm for computing the correct authorization is therefore as follows:

1. If the object has an explicit strong authorization, this is the actual authorization.
2. else, if exists a strong authorization to an object of a higher level in the hierarchy, then that the authorization
3. Else if there is a weak authorization to the object than that the authorization.
4. Else if exists a weak authorization at a higher level, then that's the authorization otherwise no authorization.

Note that the assumption is that there are no conflicting strong authorizations, since they are rejected at definition time. In [Rab91], Rabitti et al. show how to generalize the above model for the case of a database with multiple versions and a database with groups instead of individual users.

A slightly different model was proposed by Fernandez et al [Fern94]. The differences are that all authorizations have the same strength, that partial results for a query are allowed, and the location of the object in the hierarchy on which the authorization rule is defined influences the end result. We next describe the model in more detail

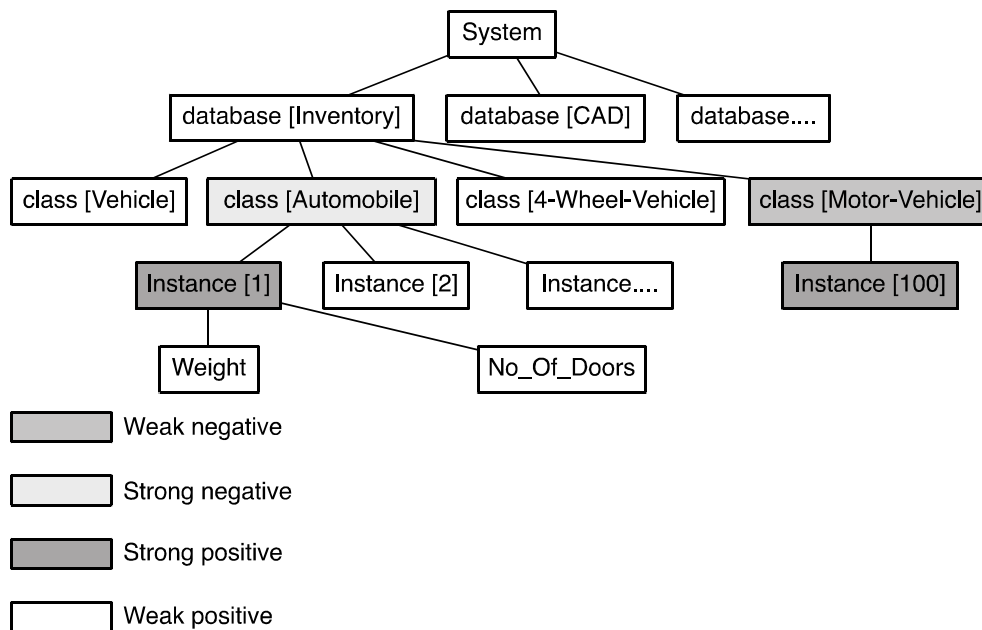


Figure 6.14: **Weak and Strong authorizations**

The model by Fernandez, Gudes and Song

The model uses an example of a university database shown in Figures 6.15a and 6.15b.

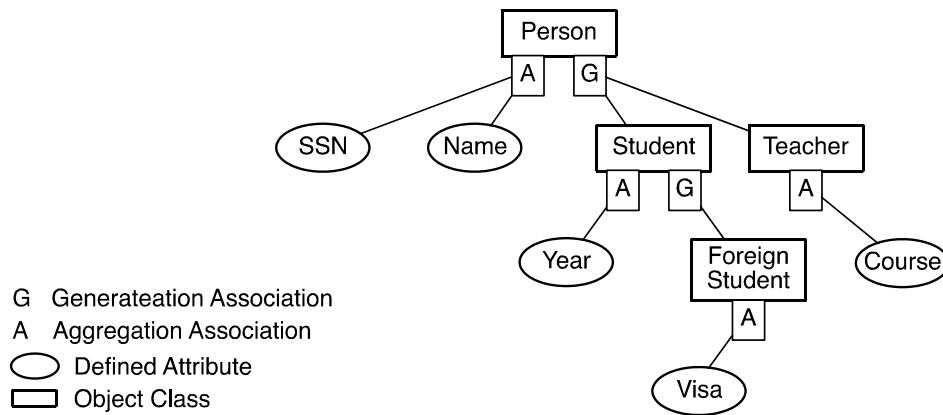


Figure 6.15a: The University database example

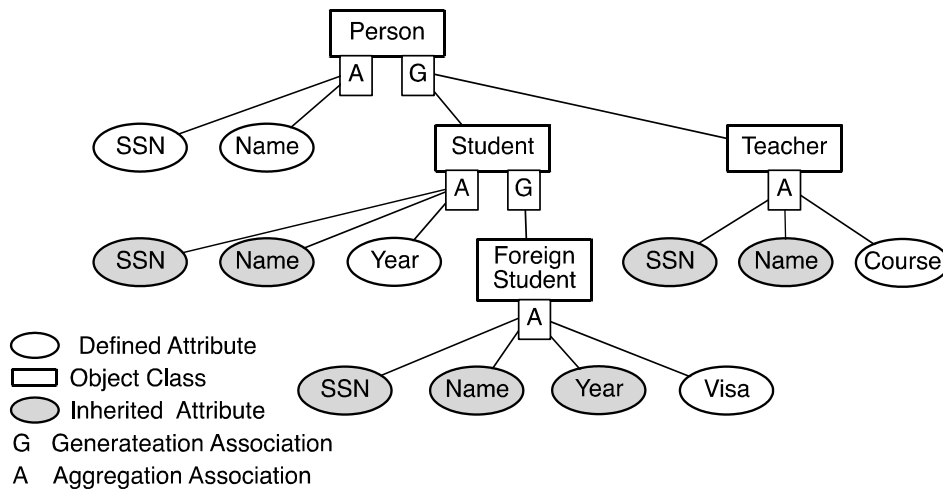


Figure 6.15b: Effective structure of the database of Fig. 6.15a

First a set of policies for such a database is defined, the policies are:

- *P1 (inherited authorization)*: a user who has access to a class is allowed to have similar type of access in the corresponding subclasses to the attributes inherited from that class.
- *P2 (class access)*: access to a complete class implies access to the attributes defined in that class (but only to the class-relevant values of these attributes). If there is more than one ancestor (multiple inheritance) access is granted to the union of the inherited attributes.
- *P3 (visibility)*: an attribute defined only for a subclass is not accessible by accessing any of its super classes.

For example, consider the graph of Fig. 6.15a, and assume the following authorization rules are defined:

R1: (S A, Read, S.SSN) – The Student Advisor can read SSN of students.

R2: (FSA, Read, (FS.SSN, FS.Visa)) – The Foreign Student Advisor can Read SSN and Visa of foreign students.

As a result of the above policies, a Student Advisor (SA) could have access to SSNs of all students (P1), but no access to their visas (P3); a Foreign Students Advisor (FSA) could have access to visas but only to SSNs of Foreign Students (P2).

For this example we define the following two queries, each of which is issued by SA and FSA.

Q1: read SSN for all students

Q2: read SSN and visa for all foreign students.

According to these policies, we expect the following behavior as a result of the evaluation of the indicated requests:

(SA, Q1) = (SA, Read, S.SSN) – All SSNs can be read (Policy P1).

(SA, Q2) = (SA, Read, {FS.Visa, FS.SSN}) – Only SSNs of foreign students are to be read and not their visas (Policy P3).

(FSA, Q1) = (FSA, Read, S.SSN) – Only foreign students SSNs are to be read (Policy P2).

(FSA, Q2) = (FSA, Read, {FS.SSN, FS.Visa}) – Both foreign students SSNs and visas are to be read (policy P2).

In order to enforce the above policies one needs an appropriate access validation algorithm. The architecture of the algorithm is shown in Figure 6.16. As can be seen in the figure, it is somewhat similar to the Ingres approach (see Section 6.4 above). The query compiler generates a query graph, that's query graph is combined with the given authorization rules and produces a new structure called authorization tree. An example of an authorization tree is shown in Figure 6.17. The authorization tree (called the AT_yes tree) contains basically only the nodes and attributes which are allowed access by the rules and the three policies above. [Fer94] shows the exact evaluation algorithms which uses the above defined policies

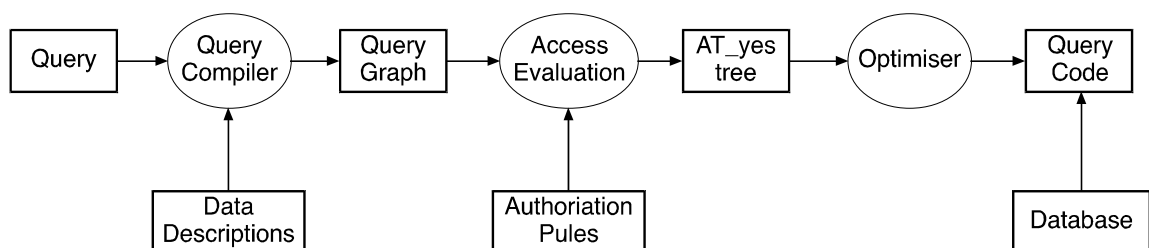


Figure 6.16: The access evaluation architecture

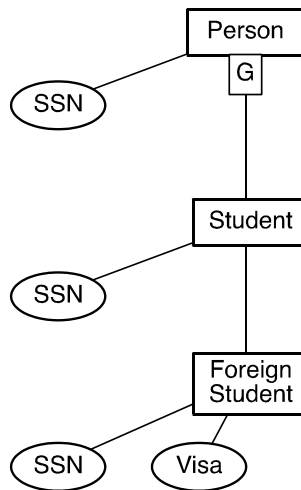


Figure 6.17: **An example for an *AT* yes tree**

Finally, the ideas from authorization models of object-oriented databases greatly influenced the ideas currently proposed for XML databases as discussed further in the next section.

6.5.2 Security in XML Databases

XML is a standard for sharing and exchanging data on the Web. Therefore it is the basis for many web applications and web services, not necessarily DBMS oriented. But since there is an increasing tendency to store data in XML files, it leads to the view of XML as a Database model. As a result there is much research and commercial products associated with XML databases, including query languages, indexing mechanisms, and security mechanisms.

In this chapter we first briefly discuss the XML standard and then elaborate on some security issues. Basic material on XML is available in any modern database book, e.g. [Ram2003].

XML is a language for representing information on the web. Its main components are: Tags, Elements, Attributes, Values and References. An XML file (instance) may be described by a Schema. Such schema can describe many instances which conform to the schema. An XML schema may be specified by a DTD or an XML schema. Figures 6.18 and 6.19 depict part of a DTD for a student-class database, and part of an XML instance conforming to this DTD.

```

<!DOCTYPE Report [
    <!--ELEMENT Report (Students, Classes, Courses)-->
    <!--ELEMENT Students (Student*)-->
    <!--ELEMENT Classes (Class*)-->
    <!--ELEMENT Courses (Course*)-->
    <!--ELEMENT Student (Name, Status, CrsTaken*)-->
    <!--ELEMENT Name (First, Last)-->
    <!--ELEMENT First (#PCDATA)-->
    .
    .
    <!--ELEMENT CrsTaken EMPTY-->
    <!--ELEMENT Class (CrsCode, Semester, ClassRoster)-->
    <!--ELEMENT Course (CrsName)-->
    .
    .
    <!--ELEMENT ClassRoster EMPTY-->
    <!--ATTLEST Report Data CDATA #IMPLIED-->
    <!--ATTLEST Student StudId ID #REQUIRED-->
    <!--ATTLEST Course CrsCode ID #REQUIRED-->
    <!--ATTLEST CrsTaken CrsCode IDREF #REQUIRED
        Semester CDATA #REQUIRED-->
    <!--ATTLEST ClassRoster Members IDREFS #IMPLIED-->
]>

```

Fig6.18 Student/ClassDTD

D

```

<?xml version="1.0" ?>
<Report Data="2000-12-12">
  <Students>
    <Student StudId="s111111111">
      <Name><First>John</First><Last>Doe</Last></Name>
      <Status>U2</Status>
      <CrsTaken CrsCode="CS308" Semester="F1997"/>
      <CrsTaken CrsCode="MAT123" Semester="F1997"/>
    </Student>
    <Student StudId="s666666666">
      <Name><First>Joe</First><Last>Public</Last></Name>
      <Status>U3</Status>
      <CrsTaken CrsCode="CS308" Semester="F1994"/>
      <CrsTaken CrsCode="MAT123" Semester="F1997"/>
    </Student>
  </Students>
  <Classes>
    <Class>
      <CrsCode>CS308</Semester>F1994</Semester>
      <ClassRoster Member="s666666666 s987654321"/>
    </Class>
    <Class>
      <CrsCode>MAT123</CrsCode><Semester>F1997</Semester>
      <ClassRoster Member="s111111111 s666666666"/>
    </Class>
  </Classes>
  <Courses>
    <Course CrsCode="CS308">
      <CrsName>Software Engineering</CrsName>
    </Course>
    <Course CrsCode="MAT123">
      <CrsName>Algebra</CrsName>
    </Course>
  </Courses>
</Report>

```

Figure 6.19 An XML instance file

The most common language to query XML databases is XQUERY. XQUERY has similarity with SQL and uses XPATH to navigate the tree structure XML file and to express qualifying

predicates. XPATH is basically a regular expression on one or more paths of the XML tree structure. The basic format of an XPATH expression is (label.label)* where label is the name of an XML element, e.g.: *student.name.first*. An attribute can also appear in an XPATH expression, below is an example of an XPATH expression retrieving students who have taken a course in 1994, using the attribute: Semester.

```
//Student [CrsTaken/@Semester="F1994"]
```

As we'll see next, path expressions are used to express security policies for XML.

Security in XML databases

When discussing security in XML, we must realize that there are several aspects and several standards that relate to this subject (usually published by W3C). The overall security issues include also Authentication and Encryption and they are described by standards such as: XML Encryption and SAML, and these standards are discussed in Chapter 8.

Here we concentrate on *Authorization*. Several papers have appeared on XML authorization, among them [Dam02] and [Bert00]. These early work describe ways to represent XML authorization rules and provide some policies for conflict resolution. The rules and policies are very similar to what we have seen in Section 6.5.1 on authorization in OO databases, since here we also deal with hierarchical structures, and the policies for resolving positive and negative authorization are similar. Later the standard XCAML was developed based on the above early works, and we will discuss it briefly afterwards.

[DAM02] defines two types of objects on which authorization rules are defined.

Authorization subjects – may be users, IP addresses or any set of identifiers with hierarchy between them

Authorization objects – may be a URI representing an XML document or part of such document represented by an XPATH expression.

The authorization rules may be defined on the DTD in which case it applies to all conforming XML files, or on a specific XML instance file.

An authorization rule is defined as a combination of five components as appeared in [DAM02]

- *subject* \in AS is the subject to whom the authorization is granted;
- *object* is either a URI in Obj or is of the form *URI:PE*, where *URI* \in Obj and *PE* is a path expression on the tree of document *URI*;
- *action* = read is the action being authorized or forbidden;⁴
- *sign* \in {+, -} is the sign of the authorization, which can be positive (allow access) or negative (forbid access);
- *type* \in {LDH, RDH, L, R, LD, RD, LS, RS} is the type of the authorization (Local DTD Hard, Recursive DTD Hard, Local, Recursive, Local DTD, Recursive DTD, Local Soft, and Recursive Soft, respectively).

As an example, the following rules define authorization to parts of a hospital database in the order of Subject, Object (path expression), Action, Sign, Type

- *Public*, *, * /department/@name read C L
- *Public*, *, * /department/division read C L
- *Administrative*, *, *.hospital.com /department//name read C LDH

- *Administrative,*,*.hospital.com /department//address read C RDH*

It is interesting to note that the type attribute is a generalization of the policies we have seen for object-oriented databases, i.e Negative overrides Positive or Strong override Weak authorization.

Using the above rules definitions and the XPATH expressions, [DAM02] presents a detailed algorithm to compute the part of the XML document which is authorized to the particular user. This algorithm called *ComputeView* and it scans both the original XML file and the rules database in parallel and based on the defined policies decides which nodes of the XML file are "visible" and which are not (see the [Dam02] for details).

Later the W3C organization developed the XACML standard which followed up and expanded the above ideas.

In XACML there are three different language components. A component to define the requests which require authorization, a component to describe the response, and a component to define the authorization rules. This is depicted in Figure 6.20

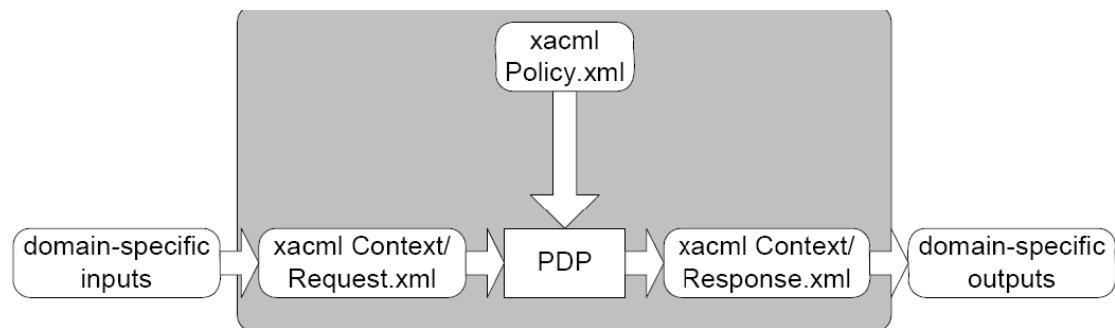


Figure 6.20 XACML components

The three components are defined by XML elements and they are input to a module called PDP (Policy Decision Point) which makes the decision based on the authorization rules with an algorithm similar to [DAM02].

The following example demonstrates the XML syntax of an authorization rule in **XACML**:

```

<Rule>
  RuleID = "urn: .... : SimpleRule1"    //urn = uniform resource name
  Effect = "Permit"
  <Target>
    <Subjects> <AnySubject/> </Subjects>
    <Resources> ..... medical record ..... </Resources>
    <Actions> .....Read.....</Actions>
  </Target>
  <Conditions>... value of subject name = value of medical record patient name .....
  </Conditions>
</Rule>

```

This rule specifies that patient may read its own personal file.

Another rule may define that a patient cannot reads its personal file if it is marked as "sensitive". If there are several rules, one can define a "policy" which specifies that deny override positive access. This is defined by the Policy component as follows:


```

</Policy>
  PolicyID = "urn: .... : SimplePolicy1"
  RuleCombiningAlgID = "deny-overrides"> //can be permit-overrides as well.
  <Target>
    Med record
    Read
  </Target>
  <Rule>
    ... SimpleRule1
    ...SimpleRule2
  </Rule>
</Policy>

```

One can also define an algorithm to combine policies called: *policyCombiningAlgorithm*.

As mentioned above, XACML is very general and the objects involved may not be just users or XML files. For example one can define an authorization rule based on email addresses or IP addresses to certain resources (this somewhat overlaps with an Application firewall, see Chapter 7).

Finally we like to mention some of the other parts of the XML security standard, these will be discussed more in Chapter 8:

XML Digital Signatures - Provides integrity and Authenticity of XML messages

XML Encryption – provides encryption of XML messages

XML Key Management – provides way to share public keys

SAML – provides way to associate objects with their Access rights

6.6 Security in Multi-level Databases

In chapter 2 we explained the advantages of the Mandatory policy and the multi-level security models. In chapter 4 we discussed the implication of developing a secure operating system, which supports a multi-level model and we explained the guidelines of the “Orange book” in this area. In this section we extend the ideas to the database domain and discuss the characteristics of a DBMS, which supports a multi-level security policy. Since a database contains usually the most sensitive information, it is reasonable to require that such a database will provide a multi-level support, especially if this DBMS runs under a multi-level operating system (clearly, the latter is a must, otherwise security will be violated, once the information is extracted from the database...)

In this section we assume the Relational model (just little work was done on multi-level security in other models see e.g. [Jaj90]). We also like to remind the reader of the two principles of the BLP model:

No read up - a user of level i , cannot read information of level j where $j > i$.

No write down - a user of level i cannot write information into an object of level j where $j < i$.
The water- mark policy - which a program can write into level j provided it did not access an object with level $i > j$, is a refinement on the above.

The first main issue is: What is the basic unit on which labels are defined?

It turns out that as the case with discretionary policy, secure labeling is required at all levels. In particular, one can define labels of the entire database, of the entire relation, of a single attribute, of a single tuple or of an attribute/value pair. As an example, assume a database table contains budgets of various projects; it may be that the whole table may be classified at level confidential, but a particular project's budget will be classified at the top-secret level. Another example is a table of employees, which is unclassified except for the salary attribute, which is classified at the secret level.

The main problem addressed by most researchers is how to preserve the general semantics and integrity of a database and yet provide multi-level support at multiple granularities. We will base our solution mainly on ideas discussed by Jajodia and Sandhu [Jaj91] and the Seaview model [Lunt90].

First let us define the schema of a multi-level relation as: $R(A_1, C_1, A_2, C_2, \dots, A_n, C_n, C)$

Where A_i is the i 's attribute, C_i - is the label of the i 's attribute and C is the label of the total tuple. An example can be seen in Table 6.1.

According to the BLP model we saw in Chapter 2, a user may see the information in the table only if her label is larger than that of the table (tuple, column). Therefore two different users looking at the same table will see different tables as is demonstrated in Table 6.2. As can be seen from this table, a user at the S level, cannot see Sam's row and also cannot see Ann's salary and see 'Null' instead, also the level of Ann's row is decreased to S. Note that this view is consistent with views we have seen earlier in this chapter, i.e. when a user likes to access information, which *part* of it is denied to her, she still gets to view the allowed parts!

Name	CName	Department	CDepartment	Salary	CSalary	TC
John	S	Dept 1	S	20 K	S	S
Ann	S	Dept 2	S	20 K	TS	TS
Sam	TS	Dept 3	TS	40 K	TS	TS

Table 6.1

Name	CName	Department	CDepartment	Salary	CSalary	TC
John	S	Dept 1	S	20 K	S	S
Ann	S	Dept 2	S	—	S	S

S-Instance

Name	CName	Department	CDepartment	Salary	CSalary	TC
John	S	Dept 1	S	20 K	S	S
Ann	S	Dept 2	S	20 K	TS	TS
Sam	TS	Dept 3	TS	40 K	TS	TS

TS-Instance

Table 6.2

The main problem with databases of the above nature is the requirement to maintain their integrity under various update transactions. We shall review this using the model suggested in [Jaj91].

The model assumes the existence of a primary key, which is a set of attributes. The first integrity constraint is: **Entity integrity**. This constraint involves three parts: no attribute of a primary key may be null. All attributes of a primary key in a single tuple are classified at the same level and the primary key has the lowest level of all attributes classified in the tuple. The first constraint is similar to the one in standard relational systems. The second constraint assures that a primary key is either completely visible or completely isn't. The third constraint assures that no information can be seen with a null primary key.

The second constraint is: **Null Integrity** and it restricts null values to be classified at the level of the key. This constraint is used to solve the problem of *subsumption*. One tuple t subsumes another tuple s , if for every non-null attribute they have the same values, and if there is an attribute in s for which the value is null, and it is not null for the same attribute in t . We will see later the use of subsumption for the case of *Polyinstantiation*.

The third constraint is called **inter-instance integrity** and it allows different users with different classifications to view the information in the table differently, as was demonstrated above.

The fourth constraint is called **Polyinstantiation integrity** and it assures that the entity integrity constraint will be satisfied under various Update operations. Basically, it specifies that the primary key of the relation is the set $(A_k, C_k \text{ and } C)$, where A_k is the user primary key, C_k is the label of that key, and C is the label of the entire tuple.

The motivation for this constraint is the need to avoid leaking of high level information to low level users. Next, let us now see some examples of *Polyinstantiation*.

Name	CName	Department	CDepartment	Salary	CSalary	TC
John	S	Dept 1	S	20 K	S	S
Ann	S	Dept 2	S	20 K	TS	TS
Sam	TS	Dept 2	TS	30 K	TS	TS
Sam	S	Dept 1	S	10 K	S	S

Table 6.3

Name	CName	Department	CDepartment	Salary	CSalary	TC
John	S	Dept 1	S	20 K	S	S
Ann	S	Dept 2	S	20 K	TS	TS
Ann	S	Dept 2	S	30 K	S	S
Sam	TS	Dept 2	TS	30 K	TS	TS

Table 6.4

Let assume that in Table 6.1, a S user likes to add a new row with the primary key "Sam". If we reject this insert due to the entity integrity constraint (duplicate primary key), we will reveal to the S user that there is a record with key "sam" at the TS level, this is bad! So we need to accept this update, and maintain **two** instances with primary key "Sam". This is seen in Table 6.3, the S user will see only one entry, but the TS user will see both! Note that a similar situation arises if a TS user wants to add a row with key "Ann", but in this case, a rejection will not leak anything, just prevent some valid service.

A similar situation will result with Table 6.4 as a result of a user S updating the salary of Ann from "Null" to 30K in table 6.2. Now, let us compute the view of user S on table 6.4. User S will get two rows for Ann, where in one of these rows, Ann's salary is null (instead of 20K). This row is then subsumed by the other row with value 30K, which is exactly the example for *subsumption* we mentioned above.

Finally, if a TS user will issue the following command to Table 6.4:
 set Dept = Dept3 where Name = "Ann",
 two rows will be added to Table 6.4:

Ann, S,Dept3,TS,30K,TS,TS

and

Ann, S,Dept3,TS,20K,S

This assures that a TS user asking a query: Dept=Dept3 and Salary=20K will receive a positive answer by looking at a single row! To satisfy this requirement we therefore generalize the poly-instantiation rule to include in the primary key the classes of all the attributes in the relation, and the rule states: if two tuples of the original relation have the same primary key, but after the update there is an attribute which has different classes and therefore different values, then any com-

bination of these values gives a new tuple in the relation. (This is called: Poly-instantiation integrity).

One problem which polyinstantiation does not solve is the following. Suppose there is a constraint that the Salary attribute cannot be null, in that case having a null answer will leak information! (That there is a value at a higher level). The solution is to store a false value; such a false value is commonly called a "cover story". See [Cup01] for details.

Another problem has to do with **Concurrency control**. Assume that a certain record is being updated (and locked for Update), by a user at level S. Assume another user at level U is trying to read (and lock it for Read) this record. If the Update lock is maintained at level S (or by an S scheduler), then rejecting user U's lock will leak information! If on the other hand, the lock is maintained at level U (or by a U scheduler), then allowing the lock by a user S violates the "No write-down" principle! There are some solutions to this problem with Multi-level schedulers. See [Jaj90a] for details.

6.7 Inference and Statistical Database Security

In chapter 1 we discussed the general problem of *Privacy*. Since databases often contain private information of individuals, privacy is of great concern. The problem in general derives from the fact that access control and security restrictions of the DAC nature we studied in earlier chapters are usually not sufficient. Sometimes, secret and private information may be inferred by asking legal queries.

Sometimes, data is given to some users for purposes of statistical research only (this is especially important in areas such as Medicine). The problem is that even if individual identities are hidden, private information may still be inferred. Note that not only precise values are a problem (e.g. salary of an individual), but also range values are important, or even an existence of a value. For example, the fact that a person has non-zero number of felonies may be very sensitive, even without knowing the exact value.

In this section we focus on one type of attack on privacy, the statistical database problem. Other types of inference attacks are mentioned at the end.

Statistical databases

By a statistical database we mean a database whose data was collected over a period of time (or extracted from a data warehouse), and its use is for analysis and learning past trends in order to aid decisions and planning. Such databases are common in several domains such as: marketing, medical research and social sciences.

Usually, the databases contain information on individuals. This information may be sensitive, but the identities of the individuals are hidden. In addition, usually only statistical type of queries, such as: Sum or Average are allowed. Yet private information may be inferred as will be shown below.

Consider the database in Table 6.5. In this table we see information on employees, including their salaries which is sensitive information. Note that the "name" field is shown for purposes of

presentation, and is actually not stored in the database. Our goal is to protect the salary information of individuals, while the "enemy's" goal is to disclose it.

The legal queries are of the form: function (Predicate; Attribute), where the function may be SUM or COUNT and the attribute is the field on which the function operates. As an example, assume that we know that Smith is a woman and she lives in San Francisco (SF), since there is only one such individual, the following query which is a valid statistical query will disclose the salary of Smith:

SUM (LOC = 'SF' \wedge SEX = 'F'; SAL)

An obvious protection is not to allow answers to queries whose answer-set size is exactly 1!

However, this can be overcome by the "enemy" by using the following two queries:

$Q_1 = \text{COUNT}(\text{LOC} = \text{'SF'} \vee \text{LOC} \neq \text{'SF'})$

$Q_2 = \text{COUNT}(\text{LOC} = \text{'SF'} \vee \text{SEX} \neq \text{'F'})$

When we compute $Q_1 - Q_2$ we get 1, then we can issue the following two queries:

$Q_3 = \text{SUM}(\text{LOC} = \text{'SF'} \vee \text{LOC} \neq \text{'SF'}; \text{SAL})$

$Q_4 = \text{SUM}(\text{LOC} = \text{'SF'} \vee \text{SEX} \neq \text{'F'}; \text{SAL})$

Name (Not stored)	Sex	Lev (Job level)	Loc (Work location)	Sal (Salary)	Status
Diaz	M	60	SF	36	1
Smith	F	58	SF	24	3
Jones	M	56	LA	26	4
Katz	M	57	LA	30	3
Clark	F	58	LA	28	5
Wond	F	60	LA	34	1
Cruze	M	58	SF	35	5

Table 6.5: a statistical database

And the desired salary is obtained by performing: $Q_3 - Q_4$!

One may think that forbidding queries whose answer-set is 1 or $n-1$ will be sufficient, but this is not the case. Denning and others have shown methods by which we can infer the private information even if the answer-set sizes are restricted [Den79, Den80]. The method is called the *Tracker*.

In [Den79], Denning et al. define the notion of an *individual tracker*. Individual tracker can be used to find information on a specific individual. Let assume there is a predicate C which characterize uniquely the individual, therefore $Q(C)$ cannot be answered. But if we can decompose C into two parts: $C = A \ \& \ B$ and the following two queries can be answered:

$Q(A)$ and

$Q(\overline{AB})$.

then the following can be used to derive the secret information:

$$Q(C) = Q(A) - Q(A\bar{B})$$

and using a function on attribute S:

$$Q(C * S) = Q(A * S) - Q(A\bar{B} * S) !$$

Now, one may argue that finding individual trackers may be quite hard. In a follow-up paper [Den80] Denning et. al. show how to find a general tracker, which may be applied for finding private information on any individual!

Even if the database owner limits the answer-set size to sizes greater than K or smaller than N-K, such a tracker can be found, provided $K \leq N/4$! This is not a big limitation! Such a tracker, called T must satisfy the equation

$$2K \leq Q(T) \leq n - 2K,$$

Obviously if we denote T' as the complement of T, then it's easy to show that T' satisfies the same condition as T and in that case the private information may be computed as:

$$Q(C+T) + Q(C+T') - Q(T+T'),$$

And it's very easy to show that each of the above three queries is legal (Show it!). Unfortunately, the procedure required to find a tracker is quite simple as was shown in [Den80].

The Tracker is a special case of an attack method which uses multiple queries issued by the same user. Another way of using multiple queries investigated by Dobkin and Jones [Dob79] represents each query as a linear equation (this can be true e.g. for "sum" queries) and the set of queries as a system of linear equations. For instance given a database with 4 unknown values, a carefully built set of queries may result with the following set of equations:

$$Q1 = X1 + X2 + X3$$

$$Q2 = X1 + X2 + X4$$

$$Q3 = X1 + X3 + X4$$

$$Q4 = X2 + X3 + X4$$

It is easy to see that using the results for the above queries, one can solve for X4 as $(-2Q1 + Q2 + Q3 + Q4)/3$. In [Dob79] it is shown how to achieve a bound on the number of queries required given the input database.

The above research was done mainly in the late 70s and early 80s. A good survey of the above research can be found in [Adam89]. The subject has gained renewed interest since then and became a real worry of institutes such as National statistical institute in the US, or Euro stat in Europe. A more recent survey on this can be found in [Gus98].

- In general, there is no single method to protect inference from statistical databases usually, one of the following techniques may be used:
- Replacing exact values with range values, or combination of values

- Introducing noise (which is distributed according to the original distributions) either in the data or in the answer, so in average most statistical queries will not deviate considerably from the correct values.
- Use a sample of the database and not the entire database, where the "enemy" cannot know if the individual he is looking for is in the sample or not.
- Form a log of past queries and analyze this log to find out if the user is trying to find a tracker, or use other types of inference
- Partition the database such that queries will always be from multiple partitions.
- Use micro-aggregation. In this method, raw individual data is replaced by small aggregations, and the mean value of the aggregation is published instead of individual values.

Each of the above methods was discussed in the literature; see [Adam89] and [Gus98]. There are even some software packages which provide statistical database protection, such as Argus [Hun04]. Recently there is a new direction for protecting the privacy of individuals in databases. This is called K-anonymization and it is described in Section 6.8

Other types of inference

There is much interesting research on the Inference problem, which cannot be reviewed here for space reasons. We like to mention one model in which there are two adversaries: the User and the System.

The user is trying to infer secret information, while the system is trying to protect it. The user is allowed to ask a sequence of queries. The system follows closely the user queries and constructs a knowledge-base of his accumulated knowledge. If at some point, answering the query and using the already built knowledge-base, will reveal secret information, the system refuses to answer! However, if the user is familiar with the system logic, this may not help and the user may still be able to infer the information. The following two papers [Sic83] and [Bis00] are examples of the early research.

As a simple example, assume background knowledge of the statement: "all men are mortal", and a query asking if John is a "man", where the fact if John is mortal has to be kept secret. Clearly, the database system should refuse to answer (assuming its not lying).

More recently Biskup and Bonati (see [Bis01]), have developed the theory for logic based inference models. Assuming a confidentiality policy instance in terms of sentences to be kept confidential, or alternatively in terms of complementary pairs of sentences, and an assumption on the user awareness of the policy instance, appropriate controlled query evaluation methods have been designed and proved to be secure in the following strong sense: a user cannot infer any of the policy sentences to be true in the current instance of the information system, whatever sequence of queries he is issuing. The methods are based on uniform refusals, uniform lying, or a combined approach which, if necessary, attempts to lie as long as possible and refuses otherwise. All methods appropriately keep track of assumed knowledge of the user consisting of background knowledge like semantic constraints and previous answers.

6.8 Security and Privacy for Data-warehousing and Data mining

The amount of information stored in operational databases for many organizations is huge. In order to facilitate decision making processes, companies now tend to aggregate historical data and build large data-warehouses which provide aggregated query facilities and also conduct data mining on this data to gain valuable knowledge for future decision making (e.g. in marketing and sales).

We will start by reviewing briefly some special issues regarding security of Data-warehouse. The main problem with data mining is Privacy, therefore we discuss this problem in more detail, especially the k-anonymity problem.

6.8.1 Security for Data-warehouses

This section is based mainly on [Medina2009]. The process of constructing a Data-warehouse is depicted in Figure 6.21.

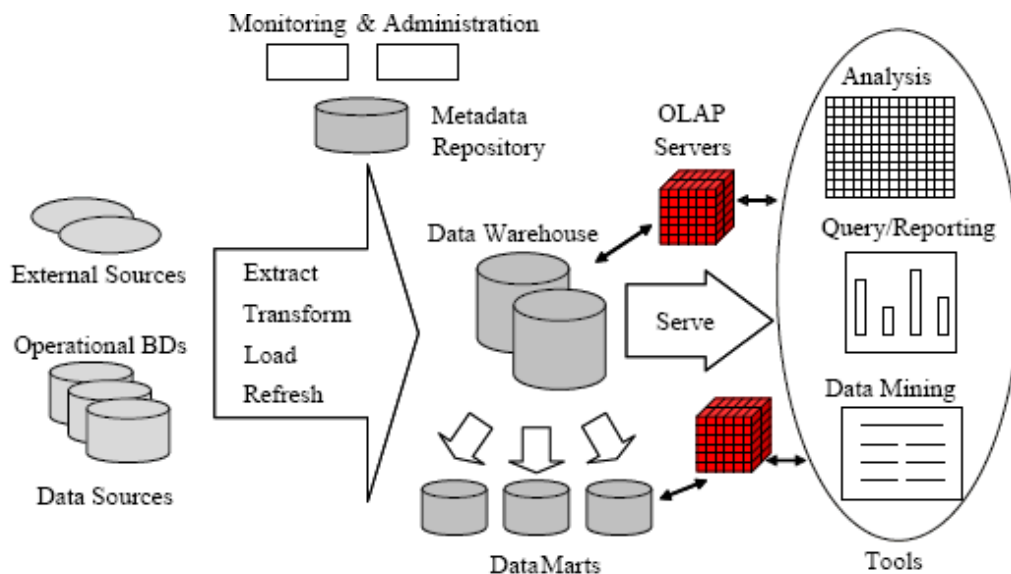


Figure 6.21: Constructing a Data-Warehouse

This process is composed of several parts, each has its own security problems.

1. Security in data sources

Here the issue is what parts of the source are accessible for the data-warehouse. This authorization can be defined by standard access control policies such as DAC and MAC. Another problem is that similar data may be stored in different sources with different security specification. Therefore there is a requirement for integrating these security policies from different sources. This has been dealt with in the context of the MAC model in [Saltor2001].

2. Security of the ETL process

Here the issue is how to protect the sensitive data during the Extract/Transform/Load process (see Fig. 6.21). It seems that encryption is the most useful tool for protecting the sensitive information before it is sanitized or anonymized.

3. Security of the Data-warehouse itself

Here the issue is different than protecting a relational database because the model is different – its multi-dimensional. There is almost no research on security of the data-warehouse modeling except for [Medina2009]. They propose a UML extension for modeling security in multi-dimensional databases and provide a tool to generate code for it for the Oracle DBMS system

4. Security of the OLAP and Mining tools

Here the issue is the definition of appropriate security constraints for the CUBE or data mining tools which are generally not part of the standard GRANT/REVOKE policies of SQL. This is discussed for example by [Pernul2001].

As mentioned earlier, the main issue with the data mining process is privacy, and this is discussed in the next section

6.8.2 Privacy for Data mining

Data mining is the process of extracting knowledge from large quantities of data. Such knowledge can be helpful in assisting decisions and predicting future behavior. There are many techniques for data mining like: Classification using Decision trees, naïve Bayes or other classifiers, extracting Association Rules, Clustering and more. Since many techniques deal with *Association rules*, we discuss them briefly here. (See [Han2000] for more details.)

Assume a database of n transactions T_i where each transaction contains m items X_i . A transaction supports an item-set $A = \{X_1, X_2, \dots, X_k\}$ if all the X_i appear in the transaction.

An association rule has the form of: $A \Rightarrow B$ where A and B are item-sets. The **support** of a rule is the number of transactions supporting the item-set $\{A \cup B\}$. The **confidence** of a rule is the ratio of $\text{support}(A \cup B) / \text{support}(A)$.

An example of such a rule may be:

70% of the times someone buys Diapers he also buys Beer with it.

The most common problem in association rules is finding frequent item-sets, i.e. item-sets which have support larger than a threshold S . The most popular algorithm to find frequent item-sets is the Ariori algorithm first proposed by Agrawal and Srikant in [Agraw94].

Since the data on which the mining is done may contain very sensitive information and the results of the mining process may also be very valuable, one is faced with the following privacy problems:

- a. Organizations like Hospitals or US Census Bureau like to publish data for the public for conducting statistical analysis and data mining. Such data may be financial data or medical data and may contain very sensitive information on individuals. The common method to protect the privacy of individuals in this case is *K-anonymity*.
- b. Often organizations are interested to perform specific data mining tasks and release the results to certain people. In that case the privacy problem is not the release of the source data but the release of the data mining results, e.g. the resulted Association rules. The problem in that case is how to protect sensitive association rules without sacrificing too much the utility of non-sensitive rules. This in general is called PPDM – *Privacy preserving data mining*
- c. There are cases where different organizations (or different departments within an organization) like to perform data mining on their global aggregated data and disclose the results, without each revealing to the other its own private local data. This is called *Privacy preserving distributed data mining*

Next we discuss briefly each of the above topics.

K-anonymity was first introduced by Sweeney in [Sweeney95]. The motivation of this approach was that even if one removes the individual identifiers from a database, one may be left with enough information to relate individuals to their sensitive attributes. Sweeney gave as an example a database from the US Census in which one can identify people by the combination of their age, sex and zip-code. To overcome this problem one usually modifies the database in such a way that at least k records will have the same quasi-identifier (e.g. the above set of attributes) and thus a sensitive attribute like a disease or salary could not be linked to a specific individual.

The formal definition of K-anonymity is as follows:

Let D be a database that holds information on individuals in some population. Each record in the table has several attributes, and we distinguish between identifiers, quasi-identifiers, and classified attributes. Identifiers are attributes which may uniquely identify the individual, e.g. name or id. Quasi-identifiers are attributes, such as age or zip-code, that appear also in publicly-accessible databases and may be used in order to identify a person. The classified attributes are those that carry information that cannot be found in publicly available databases, like a medical diagnosis or the salary of the person. K -anonymity is a model that was proposed in order to prevent the disclosure of the classified attributes for the purpose of protecting the privacy of the individuals that are represented in the database. When discussing k -anonymization, the identifiers are suppressed. Hence, assuming that there are d quasi-identifiers, say $A_1 \dots A_d$, and, for the sake of simplicity, one classified attribute, A_{d+1} , here, A_j is the domain which the j th attribute takes values from (say, if A_j is the age then it is a bounded non-negative number). Next, we define the notion of generalization. Assume that each domain of a quasi-identifier, A_j , is accompanied by a *taxonomy* T_j , which is a generalization hierarchy tree in which each node corresponds to a subset of A_j values (e.g. a tree branch for the address attribute may be: (Queens, New-York city, New-York state, USA, etc.)). The tree has $|T_j|$ leaves – one for each singleton subset of A_j ; the root corresponds to the whole set; and the subset of each node is the union of the subsets that correspond to the direct descendants of that node. Then if $R = (A(1) \dots A(d))$ is a projection of a record of D onto the space of quasi-identifiers, a generalization of R is a record $R = (R(1) \dots R(d))$ where $R(j)$ is a node in the taxonomy tree T_j . A k -anonymization of D is a generalization $D = \{R_1 \dots R_n\}$ where each generalized record has at least $k - 1$ other generalized records that are equal to it.

There are many algorithms to perform k -anonymity, most of them are based either on Partitioning – where one starts with the most generalized record and specializes them so long as k -anonymity is preserved ([partition2001]), or Clustering – where one starts with the original database records and starts merging them by finding the lowest common node in the taxonomy tree which covers them until all records are clustered in clusters of at least size k ([aggregative2003]). All algorithms attempt to minimize the loss of information by trying to optimize the

algorithm with respect to the information loss measure (i.e. the less information loss, the more utility is preserved...). Information loss is a function of the amount of generalization each record "suffers". (see [Tass2008] for details)

PPDM was surveyed in [Bertino2005]. They discuss the knowledge discovery process in general and then state:

"Such a knowledge discovery process, however, can also return sensitive information about individuals, compromising the individual's right to privacy. Moreover, data mining techniques can reveal critical information about business transactions, compromising the free competition in a business setting. Thus, there is a strong need to prevent disclosure not only of confidential personal information, but also of knowledge which is considered sensitive in a given context. For this reason, recently much research effort has been devoted to addressing the problem of privacy

preserving in datamining. As a result, several datamining techniques, incorporating privacy

protection mechanisms, have been developed based on different approaches. For instance,

various sanitization techniques have been proposed for hiding sensitive items or patterns

that are based on removing reserved information or inserting noise into data. Privacy

preserving classification methods, instead, prevent a miner from building a classifier able

to predict sensitive data. Additionally, privacy preserving clustering techniques have been

recently proposed, which distort sensitive numerical attributes, while preserving general

features for clustering analysis"

In the rest of [Bert2005] the authors describe their framework for evaluating PPDM and apply it to three main methods. The criteria they use for evaluation are:

- *efficiency*, that is, the ability of a privacy preserving algorithm to execute with good

performance in terms of all the resources implied by the algorithm;

- *scalability*, which evaluates the efficiency trend of a PPDM algorithm for increasing sizes

of the data from which relevant information is mined while ensuring privacy;

- *data quality* after the application of a privacy preserving technique, considered both as

the quality of data themselves and the quality of the data mining results after the hiding

strategy is applied;

- *hiding failure*, that is, the portion of sensitive information that is not hidden by the application of a privacy preservation technique;
- *privacy level* offered by a privacy preserving technique, which estimates the degree of uncertainty, according to which sensitive information, that has been hidden, can still be predicted.

They apply their method on several algorithms and especially on the problem of hiding sensitive association rules by eliminating their supporting large item-sets. The problem of *association rule hiding* (see [Zaine2005]) can be stated as follows: given a database D , a set R of relevant rules that are mined from D and a subset R_h of those sensitive rules included in R , we want to transform D into a database D' in such a way that the rules in R can still be mined, except for the rules in R_h . They evaluate several such algorithms using the above criteria and discuss their experimental evaluation.

Distributed privacy preserving data mining was discussed in [Clifton2002] and in [Rosenberg2006]. The problem has basically two formats. In the first format the data is distributed horizontally. This means that the same schema exists in each of the databases but with different data. This can suit for example a database of personnel partitioned across sites with the same definitions in each site but different people (e.g. customers of two branches of a bank). The second format is the vertical partition. In this case, different databases hold different schema (attributes) but contain information on the same transaction-ids (e.g. people). This can correspond to several different shops in which the same people shop but the items bought are quite different. We next present briefly the two algorithms.

The *Horizontal* algorithm works as follows. First each site finds all its frequent item-sets locally. It then broadcasts an encrypted version of them to all other sites. After all sites have finished broadcasting, the set of potential candidates is known (this is because a globally frequent item-set must be locally frequent in at least one database). Now we need to compute the global support of each candidate. This is done by computing the local support and then summing the individual supports. In order not to reveal the individual local support values, one uses a simple protocol of hiding them using a random number generated by the first site and using secure sum computation by the last site, as follows:

- Site 0 generates random R
- Sends $R + count_0 - frequency * dbsize_0$ to site 1
- Site k adds $count_k - frequency * dbsize_k$, sends to site $k+1$
- Final result: Is sum at site $n - R$.
- The final sum is obtained using Secure Two-Party Computation

The first part of the algorithm is also depicted in Figure 6.22.

The *Vertical* algorithm works as follows. For each global item-set, each site finds if its own part is locally frequent. If it is then it saves the Ids of the transactions supporting it. Next, all sites do

secure intersection on their local Id sets. If the size of the intersection is larger than the support threshold then the item is globally frequent.

For example assuming site A contains attributes A1, A3, A5 and site B containing attributes B2, B3 then to find out if {A1, A3, A5, B2, B3} is frequent

- A forms the binary vector $X = A1\ A3\ A5$
- B forms the binary vector $Y = B2\ B3$
- Securely compute the dot product of X and Y

In the binary vector, a '1' means that the item set is present in this transaction, and a '0' means it isn't present in this transaction. The secure computation algorithms are quite standard and will not be described here, since they appear in most cryptography books [Shcneyer2000].

(The classic problem of secure computation is as follows:

Two millionaires Bob and Alice like to find out who is richer without disclosing to each other the amount of money they have. This problem is solved by a standard secure computation protocol.)

To summarize this section, we briefly presented several aspects of privacy in data mining. The interested reader should look at the references for more details.

Computing Candidate Sets

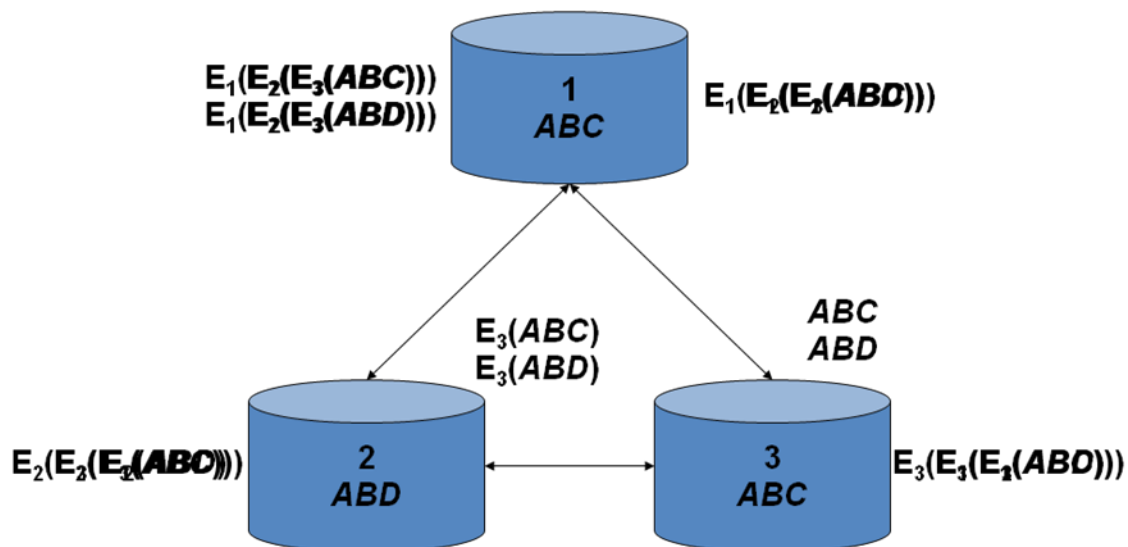


Figure 6.22: Horizontal Frequent Item-set computation

6.9 Security of Web Databases

Until now we have assumed that users and administrators interact directly with the database and therefore the use of the standard mechanisms of SQL like Views and Grant/Revoke is applied. The situation in the Web world is different. In this world, users are connected to a Web server which itself interacts with the database (this e.g. is the situation with all popular web sites like Amazon, E-bay or CNN). This is depicted in Figure 6.23.

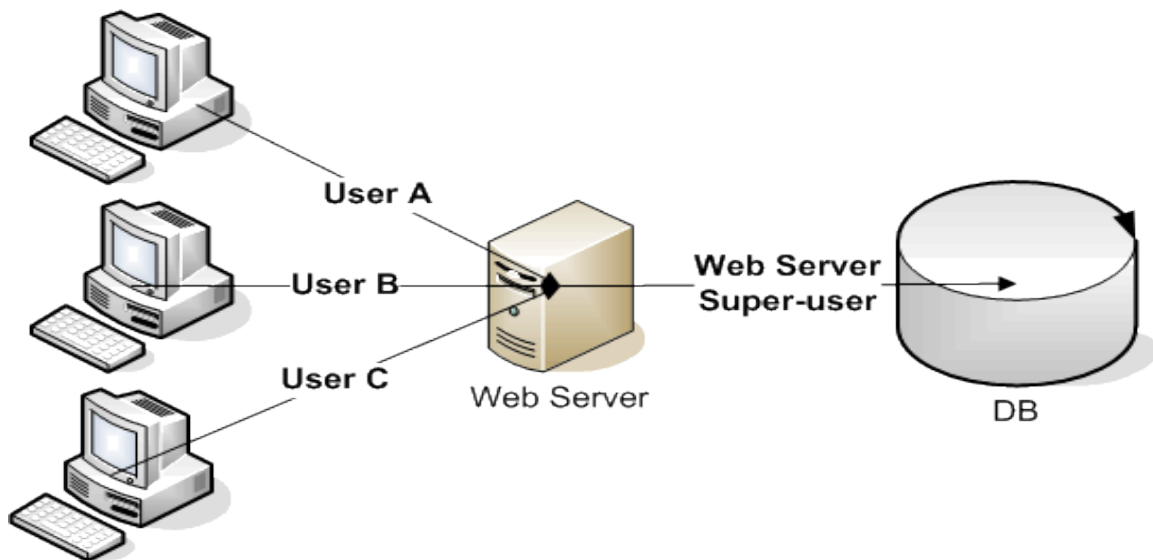


Figure 6.23: a Typical Web/Database architecture

In this architecture all the communication to the database is done by the web server which interacts with the users. In contrast with the standard database architecture where there are a small number of known users which are authenticated to the DBMS at session initiation time, and where it is easy to associate transactions to the user performing them, in the Web/Database case we have the following situation:

- Users are casual and unknown
- Users' number is not limited
- Users do not connect directly to the DB

Furthermore, often the technique of *Connection Pooling* is used. In this case, different web users can run their SQL statements on the same DB connection and one user can run her SQL statements on different connections. This technique contributes to application efficiency since the time to open and close the connection is saved per each request, but it has serious implications on the database's access control mechanism. Now, the DBMS cannot identify the real application user who accesses it, and usually the only user accessing the database is the 'user' of the web application server which has "super-user" privileges. This means that at the database level, no user-based access control and no RBAC can be applied, and the principle of minimal privilege is violated.

Now, this situation would be ok if Web applications would have been written by security experts, but this is often not the case and as a result the Web application and the database itself are very vulnerable.

The most common attack on Web databases is the SQL injection attack. This attack happens when a web application displays a web page to the user asking him to fill various parameters for example his employee number and the current date. These parameters are transferred to the Web server by an HTTP Get or Post command, and the Web server in turn fetches these parameters and insert them into an SQL template which exists in the application, and then sends the resulted dynamic SQL to the DBMS. As an example suppose the SQL statement template is:

```

strSQL= "SELECT Salary FROM Salary_Table
        WHERE Employee_No = emp_NoParam
        AND Salary_Date = '' + dateParam + ''"
  
```

But the user enters into the form in the Date field another string which results with the following SQL statement:

```
SELECT Salary FROM Salary_Table
WHERE Employee_No = 123
AND Salary_Date = '01/2007' OR '1' = '1'
```

This resulted statement is very dangerous since it enables the user 123 to retrieve the salaries of all other employees. Similarly, instead of using "1=1" the user could have entered an Update or a Delete SQL statement causing disastrous consequences. The attack above is possible because the Web server has complete access to the database and is not limited by the DBMS access control mechanism.

In spite of the fact that this kind of attack has been known for several years, and even though the easiest way to solve it is to make sure the application checks the validity of the HTTP parameters, such a check is not always performed or sufficient, and these kind of attacks are still very common. Here is a list of some recent incidents:

- On November 01, 2005 a high school student used SQL injection to break into the site of a Taiwanese information security magazine and steal customer's information (http://www.webappsec.org/projects/whid/list_id_2005-46.shtml).
- On January 13, 2006 hackers broke into a Rhode Island government web site and stole credit card data from individuals who have done business online with state agencies (http://www.webappsec.org/projects/whid/list_id_2006-3.shtml).
- On March 29, 2007 a SQL injection flaw in official Indian government site was discovered (http://www.webappsec.org/projects/whid/list_id_2007-12.shtml).
- On June 29, 2007 hackers defaces Microsoft U.K. web page by using an SQL injection attack (<http://www.cgisecurity.com/2007/06/27>).
- On August 12, 2007 the United Nations web site was defaced by using SQL injection

In recent years there appeared research which on the one hand attempts to prevent such attacks by transferring some form of user-identity to the DBMS and using parameterized views (see [Roich2008]), and in case prevention is not possible work has been done on Intrusion Detection systems (IDS) which attempt to detect the violation after the fact. This is done, by applying machine learning and data mining techniques on web or database logs, trying to detect anomalies in the submitted SQL statements or parameters (see [Bert2007, Roich2008b]). These methods require first the unique identification of users and session boundaries in the logs which is a problem by itself. Once user sessions are identified, the method applied by [Roich2008b] trains a classifier by the accesses performed in one session denoting it as a session vector. The sessions vectors are clustered using hierarchical clustering to get a set clusters which can be considered as the valid roles. During the testing phase, if the tested session vector's distance from any of the clustered roles is above some threshold then an intrusion is announced.

Because of the always increasing sophistication of hackers and intruders, the IDS method combined with network and application firewalls seem to be the most effective method to protect Web database systems.

6.10 Database Encryption

Conventional database security solutions and mechanisms are divided into three layers; physical security, operating system security and DBMS (Database Management System) security. With

regard to the security of stored data, access control (i.e., authentication and authorization) has proved to be useful, as long as that data is accessed using the intended system interfaces. However, access control is useless if the attacker simply gains access to the raw database data, bypassing the traditional mechanisms. This kind of access can easily be gained by insiders, such as the system administrator and the database administrator (DBA) (also by a theft of physical disks.).

The aforementioned layers are therefore not sufficient to guarantee the security of a database when the database content is kept in a clear-text, readable form. One of the advanced measures being incorporated by enterprises to address this challenge of private data exposure, especially in the banking, government, and healthcare industries, is *database encryption*. While database-level encryption does not protect data from all kinds of attacks, it offers some level of data protection by ensuring that only authorized users can see the data, and it protects database backups in case of loss, theft, or other compromise of backup media.

Following is characterization of the challenges, problems and solutions of database encryption, based mainly on [shmueli2009].

DB encryption challenges

The challenges of DB encryption can be classified as follows:

- Protection against the various attacks
- Minimizing the encryption overhead
- Reducing the integration effort within existing DBMSs
- Managing the protection keys

Next we present more details on these challenges.

Protecting against attacks

We can distinguish between Passive and Active attacks.

Passive attacks

- (1) Static leakage - Gaining information on the database plaintext values by observing a snapshot of the database at a certain time. For example, if the database is encrypted in a way that equal plaintext values are encrypted to equal ciphertext values, statistics about the plaintext values, such as their frequencies can easily be learned.
- (2) Linkage leakage - Gaining information on the database plaintext values by linking a table value to its position in the index. For example, if the table value and the index value are encrypted the same way (both ciphertext values are equal), an observer can search the table cipher text value in the index, determine its position and estimate its plaintext value.
- (3) Dynamic leakage - Gaining information about the database plaintext values by observing and analyzing the changes performed in the database over a period of time. For example, if a user monitors the index for a period of time, and if in this period of time only one value is inserted (no values are updated or deleted), the observer can estimate its plaintext value based on its position in the index.

Active attacks

In addition to the passive attacks that observe the database, active attacks that modify the database should also be considered. Active attacks are more problematic in the sense that they may mislead the user. Unauthorized modifications can be made in several ways:

- (1) Spoofing - Replacing a ciphertext value with a generated value. A possible attacker might try to generate a valid ciphertext value, and substitute the current valid value stored on the disk. Assuming that the encryption keys were not compromised, this attack poses a relatively low risk.

- (2) Splicing - Replacing a ciphertext value with a different cipher text value. Under this attack, the encrypted content from a different location is copied to a new location under attack.
- (3) Replay - Replacing a cipher text value with an old version previously updated or deleted. Note that each of the above attacks is highly correlated to the leakage vulnerabilities discussed before: static leakage and spoofing, linkage leakage and splicing and dynamic leakage and replay attack.

Encryption Overhead

Added security measures typically introduce significant computational overhead to the running time of general database operations. However, it is desirable to reduce this overhead to the minimum that is really needed, and thus:

- (1) It should be possible to encrypt only sensitive data while keeping insensitive data unencrypted.
- (2) Only data of interest should be encrypted/decrypted during queries' execution.
- (3) Some vendors do not permit encryption of indexes, while others allow users to build indexes based on encrypted values. The latter approach results in a loss of some of the most obvious characteristics of an index - range searches, since a typical encryption algorithm is not order-preserving.
- (4) In addition, it is desirable that the encrypted database should not require much more storage than the original one.

Integration Footprint

Incorporating an encryption solution over an existing DBMS should be easy to integrate, namely, it should have:

- (1) Minimal influence on the application layer
- (2) Minimal influence on the DBA work
- (3) Minimal influence on the DBMS architecture

Handling Encryption Keys

The way encryption keys are being used can have a significant influence on both the security of the database and the practicality of the solution. The following issues should be considered:

- (1) Cryptographic Access Control – Encrypting the whole database using the same key, even if access control mechanisms are used is not enough. For example, an insider who has the encryption key and bypasses the access control mechanism can access data that are beyond his security group. Encrypting objects from different security groups using different keys ensures that a user who owns a specific key can decrypt only those objects within his security group. However, that requires secure storage and management of the keys which is not a simple task.
- (2) Secure Key Storage – Encryption keys should be kept securely, e.g., storing the keys inside the database server allows an intruder which has penetrated the DBMS server access to both the keys and the encrypted data.
- (3) Key Recovery – If encryption keys are lost or damaged, the encrypted data is worthless. Thus, it should be possible to recover encryption keys whenever needed.

The above problems pose a big challenge for database encryption. There is no one encryption scheme which can answer all the above challenges, and that may explain why database encryption is not so common solution within existing DBMS products. Next we describe some of the possible solutions suggested in the research literature.

DB encryption solutions


The DB encryption solutions may be classified along several lines:

- What is the granularity of encryption?

- Is the encryption/decryption done by the server or by the client?
- Are indexes encrypted as well?
- How keys are managed and is access control such as DAC supported?

Encryption granularity.

Encryption may be applied at the OS file level, or at the database level. If applied at the database level, it may apply to a full table, to a single record or to a single cell. The scheme suggested in [Gudes94] enables the encryption of files and in addition supports a DAC access control. The idea is depicted in Figure 6.24



K'_{j1}	K'_{j2}	...	K'_{jn_j}
Validation Record – k'_j			
File F_j			

Figure 6.24: The “keys record” scheme

In this scheme, every user has only one user key – K_{ui} . The file is encrypted using a single key K_{Fj} (with symmetric encryption). The file contains an ACL which includes an entry for every user who has access to the file in the form of $K'_{ij} = \text{Enc}(K_{ui}, K_{Fj})$. Therefore only users who have legal access to the file can decrypt this ACL entry and get the file decryption key. This idea can also be used to protect full tables in a database.

Another scheme which enables encryption at the cell level was suggested by Davida in [Davida79]. This scheme is based on the Chinese remainder theorem and allows different users to extract the values of different fields from the same record, even though the record is encrypted as a single unit.

Another scheme which encrypts at the cell level is the SPDE scheme that will be discussed below.

Client vs Server-Side Encryption

As mentioned above, storing the encryption keys at the server side may expose the database to risks from hackers who get control of the DBMS. Therefore many authors have proposed to use Client-side encryption (see [Hacig2002]). This is also called DAS – “data as a service” meaning that the data owner outsources the management of the database but would to protect its sensitive information from the outsourcing party as well. In the case of client-side encryption of [Hacig2002], the following occurs. The query is encrypted and sent to the server, and the server

searches for data which corresponds to this encryption. (Actually, the attribute values which are used in the query are mapped to hashed values which usually contain more values than asked by the query.) The scheme is depicted in Figure 6.25. There are two major problems with this approach. First, a general SQL query cannot be supported since many SQL operators cannot be performed on the encrypted data (Homomorphic encryption is suggested, but that's is not a general solution.) This means also that this scheme cannot be integrated within a general DBMS software package. Secondly, because of the hashing scheme, the server usually returns to the client more data than it requested, and the client filters the desired data at his side. This raises both performance and privacy issues.

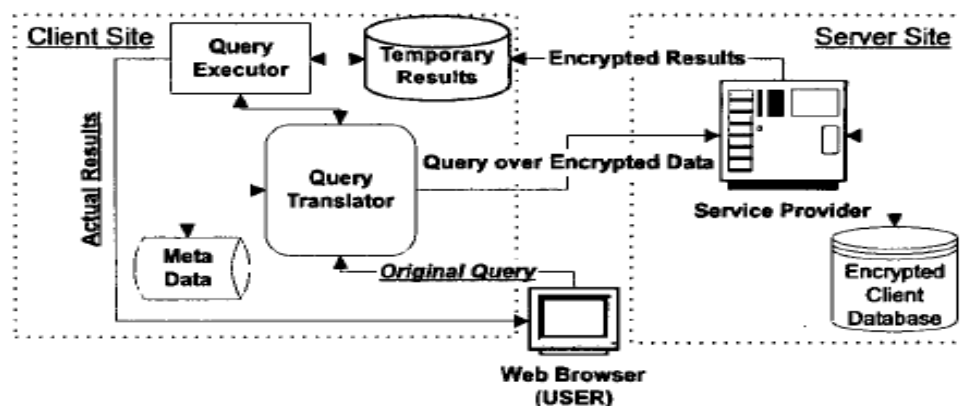


Figure 6.25: the Client-side scheme in [Hagic2002]

A different scheme called the SPDE scheme was suggested by [Shmueli2006] (see Fig. 6.26). This scheme assumes that the keys are stored in the server side but in a secure data storage device, thus they are not exposed to DBMS intruders (however clear data is exposed in the DBMS buffer and is assumed protected by memory protection mechanisms...). The SPDE scheme works at the cell level, and encrypts each cell in the database individually together with its cell coordinates (table name, column name and row-id). In this way static leakage attacks are prevented since equal plaintext values are encrypted to different cipher-text values. Furthermore, splicing attacks are prevented since each cipher-text value is correlated with a specific location, trying to move it to a different location will be easily detected. Further security analysis and fixes to this scheme can be found in [Shmueli2006]. The main advantage of this scheme is that it can be integrated easily into an existing DBMS, since the SQL run-time module works on clear data. The main disadvantage is that it is somewhat more vulnerable at the server side than the client-side scheme.

SPDE - A New Database Encryption Scheme

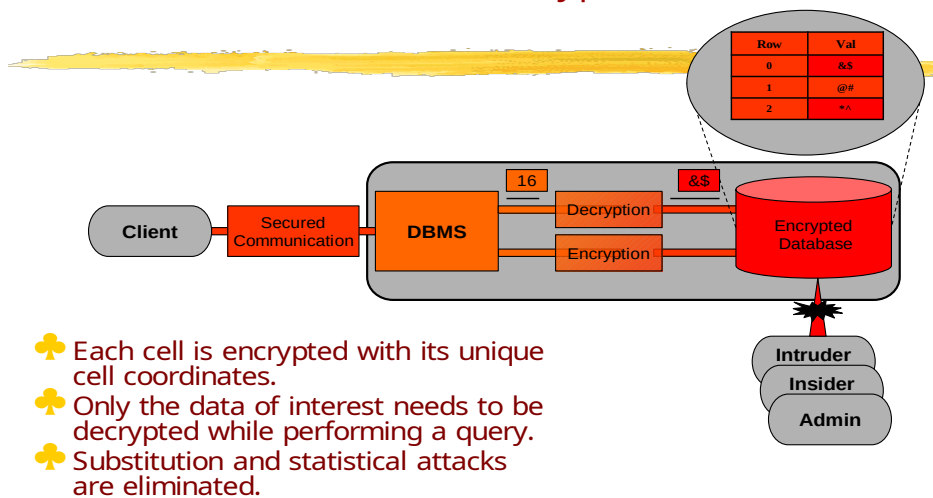


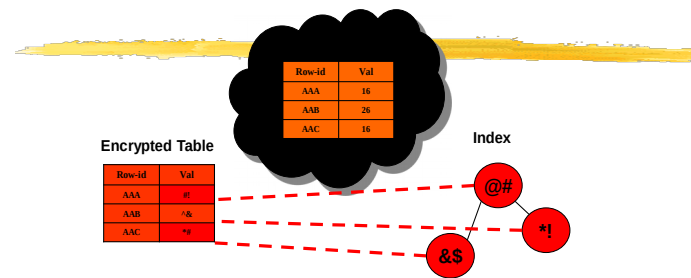
Figure 6.26; The SPDE Encryption scheme

Indexing Encrypted Data

Several proposals for encrypting database indexes have appeared in the literature. The indexing scheme proposed in [Bayer76] suggests encrypting the whole database row and assigning a set identifiers to each value in this row. The indexing scheme in [Dam2003] suggests building a B-Tree index over the table plaintext values and then encrypting the table at the row level and the B-Tree at the node level. The indexing scheme in [Iyer2004] is based on constructing the index on the plaintext values and encrypting each page of the index separately. The main problem with the above schemes is that a single key is used to encrypt each node of the index and that requires to disclose such key to all users who access the index. Another problem is scanning the index in the order of the keys to enable the fast search provided by an index. The encryption function suggested in [Ageaw2004] preserves order, and thus allows range queries to be directly applied to the encrypted data without decrypting it. In addition it enables the construction of standard indexes on the cipher-text values. However, the order of values is sensitive information in most cases and should not be exposed.

In addition to table encryption, the SPDE scheme that is presented in [Shmueli2006] offers a novel method for indexing encrypted columns. Since the keys are available at the server side, one can maintain order even with encrypted entries. Also the linkage of values in the index and in the table is avoided since the encryption function uses different row-ids. See Figure 6.27 for a schematic description of SPDE index encryption.

A New Database Indexing Scheme



- ✿ Each index value is the result of encrypting a plaintext value in the database with its row-id.
- ✿ This ensures that the index does not reveal the statistics or order of the database values.
- ✿ The new database indexing scheme preserves the index structure.

Figure 6.27: The SPDE indexing scheme

Keys' Management

Many techniques for generating encryption keys were mentioned in the literature; however, most of them are neither convenient nor flexible in the real applications. The scheme in [Chen2006] proposes a novel database encryption scheme for enhanced data sharing inside a database, while preserving data privacy. In this scheme, a pair of keys is generated for each user. The key pair is separated when it is generated. The private key is kept by user at the client end, while the public key is kept in the database server. However, the general problem of managing encryption keys to support a flexible DAC access control model is not solved yet.

The following two tables taken from [Shmueli2006] summarizes the above discussion and the current state of database encryption.

Table 6.6 compares several database encryption deployment configurations.

Table 6.7 summarizes the influence of encryption granularity on several aspects including performance and preserving the structure of the database.

Table 6.8 summarizes the dependency between the trust in the server and the keys' storage. If we have no trust in the database server, we would prefer to keep the encryption keys only at the client side. In cases where the database server itself is fully trusted, but its physical storage is not, we can store the keys at the server side in some protected region.

	File-Sys- tem En- ryption	DBMS Encryption	Application Encryption	Encryption at the Cli- ent Side
Finest encryption granu- larity supported	Page	Cell	Cell	Cell
Support for internal DBMS mechanisms (e.g. index, foreign key...).	+	+	-	-
Support for cryptograph- ic access control	-	+	+	+
Performance	Best	Medium	Low	Worst
Compatibility with leg- acy applications	+	+	-	-

Table 6.6. Comparing Different Database Encryption Configurations.

	Information Leakage	Unauthorized Modifications	Structure Perseverance	Performance
Single Values	Worst	Worst	Best	Best
Record/ Nodes	Low	Low	Medium	Medium
Pages	Medium	Medium	Low	Low
Whole	Best	Best	Worst	Worst

Table 6.7. Risk in Different Levels of Encryption Granularities.

	Server Side	Keys per Session	Client Side
Absolute	+	+	+
Partial	-	+	+
None	-	-	+

Table 6.8. Keys Storage Options and Trust in Server.

What can be seen from the above tables is that a general solution for the DB encryption problem is still far away.

6.11 Security in commercial DBMSs

Most commercial database systems follow the SQL standards (92 and 99) as described in Section 6.3. In this section we will discuss some additional features provided by the leading products.

Oracle9 provides the following additional features:

User authentication. Various options may be controlled by the DBA for the authentication process. The DBA may set a limit on the number of Logon attempts or the password life-time. It may also set restrictions on the type of passwords allowed. Oracle also supports strong authentication provided by Kerberos or smart cards. Note, however, that often the authentication may not be done by the DBMS at all! But by an external system such as a Web server or an Enterprise software.

Powerful access control. In addition to individual and Role-based security described in Section 6.3, Oracle 9i allows to define a user *profile* which can be used to limit the use of resources by each user. It also provides a much wider set of access privileges e.g. privileges such as: Analyze Backup, Manage and more, which can be assigned and granted to various users or roles. The role structure may also be more complex and provide for global roles, enterprise roles which can be part of an LDAP directory, or secure application roles which may be very useful in a web environment. Stored procedures may also be used to manage privileges.

Virtual private database. Oracle allows the definition of a virtual private database, which is a database accompanied by a set of authorization rules. These authorization rules act on the input query in a way very similar to Ingres query modification method described in Section 6.4. Virtual private database provides fine-grained access control which is data-driven and row-based without the need to construct complex views.

We describe the Oracle VPD in more detail in section 6.10.1 below.

Sensitive data protection by using Encryption. Oracle provides functionality to encrypt data stored in the database. However, the functionality is limited to whole tables and the keys must be managed by the DBMS.

Multi-level support. Oracle provides support for multi-level databases discussed in Section 6.7. It is called Label-based access control. Label-based access control allows organizations to assign sensitivity labels to data rows, control access to data based on those labels, and ensure that data is marked with the appropriate sensitivity label. However, note that Oracle multi-level support does not follow the standard Bel-Lapadulla model (see Section 2).

MS SQL server provides similar features. One additional feature is an extensive set of Application roles and various tools to generate them. SQL server security is tied very strongly with MS-Windows security, so many of the features are provided via the Windows OS, such as Authentication or Encryption. For example, Windows extensive Logging features may be used to monitor database access as well.

IBM DB2 also provides similar features. One additional feature of DB2 is that the privileges are organized in a hierarchy, and there are up to five levels of administration types of authorization in such hierarchy, such as:

System Administrator (SYSADM) authority

System Control (SYSCTRL) authority

System Maintenance (SYSMAINT) authority

Database Administrator (DBADM) authority

Load (LOAD) authority

There are also more schema types of privileges and view and table privileges. Also the GRANT and REVOKE commands have more power and flexibility than in standard SQL, and depend on the authorization level. Table 6.9 depicts some of the various capabilities of Grant and Revoke on the added privileges in IBM DB2:

If a User Holds...	They Can Grant...	They Can Revoke...
System Administrator (SYSADM) authority	System Control (SYSCTRL) authority	System Control (SYSCTRL) authority
	System Maintenance (SYSMAINT) authority	System Maintenance (SYSMAINT) authority
	Database Administrator (DBADM) authority	Database Administrator (DBADM) authority
	Load (LOAD) authority	Load (LOAD) authority
	Any database privilege, including CONTROL privilege	Any database privilege, including CONTROL privilege
	Any object privilege, including CONTROL privilege	Any object privilege, including CONTROL privilege
System Control (SYSCTRL) authority	The USE tablespace privilege	The USE tablespace privilege
System Maintenance (SYSMAINT) authority	No authorities or privileges	No authorities or privileges
Database Administrator (DBADM) authority	Any database privilege, including CONTROL privilege	Any database privilege, including CONTROL privilege
	Any object privilege, including CONTROL privilege	Any object privilege, including CONTROL privilege
Load (LOAD) authority	No authorities or privileges	No authorities or privileges
CONTROL privilege on an	All privileges available (with the	All privileges available (with the

If a User Holds...	They Can Grant...	They Can Revoke...
object (but no other authority)	exception of the CONTROL privilege) for the object the user holds CONTROL privilege on	exception of the CONTROL privilege) for the object the user holds CONTROL privilege on
A privilege on an object that was assigned with the WITH GRANT OPTION option specified	The same object privilege that was assigned with the WITH GRANT OPTION option specified	No authorities or privileges

Table 6.9: Grant/Revoke functionality in IBM DB2

Obviously, it is out of the scope of this book to provide a full comparison of the leading products, but the interested reader should consult with the Homepages of the various companies, and be aware that this information is rapidly changing.

6.11.1 Oracle VPD

The Virtual Private Database notion, developed by Oracle, is a fine grained access control mechanism that helps to resolve some of the challenges associated with views Error: Reference source not found. VPD's row-level security allows the user to restrict access to records based on a security policy implemented in PL/SQL. A security policy, as used here, simply describes the rules governing access to the data rows. This process is done by creating a PL/SQL function that returns a string. The function is then registered against the tables, or views, the user wants to protect by using the DBMS_RLS PL/SQL package. When a query is issued against the protected object, Oracle effectively appends the string returned from the function to the original SQL statement, thereby **filtering** the data records. The actual PL/SQL implementation that enforces the VPD can be based on whatever is relevant - IP address, time of day, application context values. The policies also are transparent to queries on the protected objects.

New to Oracle Database 10g is the ability to support column-sensitive policies, which allows a more selective invocation of the DBMS_RLS PL/SQL mechanisms. This is very practical and allows the user to more easily store data with different sensitivities within the same table.

An "application context" enables access conditions to be based on virtually any attribute an administrator deems significant, such as organization, subscriber number, account number, or position. For example, a warehouse of sales data can enforce access based on customer number, and whether the user is a customer, a sales representative, or a marketing analyst. In this way, customers can view their order history over the web but only for their own orders, while sales representatives can view multiple orders but only for their own customers, and analysts can analyze all sales from the previous two quarters.

Application contexts act as a secure cache of data that may be applied to a fine-grained access control policy on a particular object. Upon user login to the database, Oracle sets up an application context to each session. Information in the application context is defined by a developer based on information relevant to the particular application.

To demonstrate the notion of the VPD mechanism, let us assume that only *george_simmons* is allowed to operate on the first ten rows in *app_table*. Let us assume also that the IDs of these rows are less than 11.

To create a VPD policy, we write

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema=> 'app_user',
    object_name=> 'app_table',
    policy_name=> 'important_rows',
    policy_function=> 'vpd_function',
    statement_type=> 'SELECT,INSERT,UPDATE,DELETE');
END;
```

This policy applies to the table *app_user.app_table* (we assume that *app_user* created the table during the application setup), on the actions SELECT, UPDATE, INSERT and DELETE. In effect, this policy adds a WHERE clause to each of these functions. The where clause is defined by the stored procedure *vpd_function*.

```
CREATE OR REPLACE FUNCTION
  vpd_function(v_schema VARCHAR2, v_objname VARCHAR2)

  RETURNS VARCHAR2 AS
    u_name VARCHAR2(200);
BEGIN
  u_name := SYS_CONTEXT('userenv', 'session_user');
  IF (u_name = 'george_simmons') THEN
    RETURN "";
  END IF;
  RETURN v_schema || '.' || v_objname || '.id > 10';
END;
```

Given a schema name and an object name in the schema, this function returns an empty string (NULL) when the user is *george_simmons*. Otherwise, the clause allows work only where *id* is greater than 10.

Note that this function can be used in other objects with an *id* column where only *george_simmons* is allowed to access rows with *id* smaller or equal to 10.

Oracle Label Security (OLS)

OLS is the second type of fine-grained access control in Oracle. It allows the user to define a security policy that is implemented by marking the data records with security labels. The label markings indicate what rights a person must possess in order to read or write the data. The labels are stored with each record in a special column that is added to the table. The database users are also given labels that indicate their access rights to the data records. When the user accesses the table records, the database's OLS engine compares the user's label with the row's label marking to determine if the user can have access. This mechanism is usually used in military applications, but does not follow the standard Bell-Lapadulla model, and does not address the problems described in Section 6.6, e.g. poly-instantiation.

6.12 Database Security - Summary

This chapter discussed many different problems of security and privacy in database systems. We discussed the most common mechanisms such as the SQL Grant/Revoke mechanism. We discussed some of the special issues in object-oriented and XML databases. We discussed security of multi-level databases and of Web databases. We discussed the privacy problem in statistical

databases, and in data mining applications and we discussed database encryption. Finally we discussed some of the special features provided by the leading commercial databases. Much of the research work was only briefly described and the reader is invited to study the references for more in-depth discussions.

6.12 References

- [Adam89] Nabil R. Adam, [John C. Wortmann](#): Security-Control Methods for Statistical Databases: A Comparative Study. [ACM Comput. Surv. 21](#)(4): 515-556 (1989)
- [Agraw94] Rakesh Agrawal, [Ramakrishnan Srikant](#): Fast Algorithms for Mining Association Rules in Large Databases. [VLDB 1994](#): 487-499
- [Bark01] [S. Barker](#), A. Rosenthal: Flexible Security Policies in SQL. [DBSec 2001](#): 167-180
- [Bert99] E. Bertino, [S. Jajodia](#), [P. Samarati](#): A Flexible Authorization Mechanism for Relational Data Management Systems. [ACM Trans. Inf. Syst. 17](#)(2): 101-140 (1999)
- [Bert00] Elisa Bertino, [Silvana Castano](#), [Elena Ferrari](#), [Marco Mesiti](#): Specifying and enforcing access control policies for XML document sources. [World Wide Web 3](#)(3): 139-151 (2000)
- [Ber05] E. Bertino and R. Sandhu, "Database security—concepts, approaches, and challenges", IEEE Trans. On Dependable and Secure Computing, vol. 2, No 1, 2005, 2-19.
- [Bis00] J. Biskup: For unknown secrets refusal is better than lying. [Data Knowl. Eng. 33](#)(1): 1-23 (2000)
- [Bis01] J. Biskup, [P. A. Bonatti](#): Lying versus refusal for known potential secrets. [Data Knowl. Eng. 38](#)(2): 199-222 (2001)
- [Cas94] S. Castano, M. Fugini, G. Martella, and P. Samarati, *Database security*, Addison-Wesley 1994.
- [Cup01] F. Cuppens, [A. Gabillon](#): Cover story management. [Data Knowl. Eng. 37](#)(2): 177-201 (2001)
- [Dam02] [E. Damiani](#), [S. Capitani di Vimercati](#), [S. Paraboschi](#), P. Samarati: A fine-grained access control system for XML documents. [ACM Trans. Inf. Syst. Security. 5](#)(2): 169-202 (2002)
- [Den79] D. E. Denning, [P. J. Denning](#), [M. D. Schwartz](#): The Tracker: A Threat to Statistical Database Security. [ACM Trans. Database Syst. 4](#)(1): 76-96 (1979)
- [Den80] D. E. Denning, [J. Schlörer](#): A Fast Procedure for Finding a Tracker in a Statistical Database. [ACM Trans. Database Syst. 5](#)(1): 88-102 (1980)
- [Dob79] D. P. Dobkin, [A. K. Jones](#), [R. J. Lipton](#): Secure Databases: Protection Against User Influence. [ACM Trans. Database Syst. 4](#)(1): 97-106 (1979)
- [Downs77] [D. Downs](#), [G. J. Popek](#): A Kernel Design for a Secure Data Base Management System. Proceedings of [VLDB 1977](#): 507-514
- [Elov04] [Y. Elovici](#), [R. Waisenberg](#), [E. Shmueli](#), E. Gudes: A Structure Preserving Database Encryption Scheme. Proceedings, [Secure Data Management 2004](#): 28-40

- [Fer75] E. B. Fernandez, R. C. Summers, and C. B. Coleman, "An authorization model for a shared data base," *Proceedings of the 1974 SIGMOD International Conference*, ACM, New York, pp. 23-31, 1975.
- [Fer75a] E.B.Fernandez, R.C.Summers, and T.Lang, "Definition and evaluation of access rules in data management systems", *Procs. First Int. Conf. on Very Large Databases*, Boston, MA, 1975, 268-285.
- [Fer81] E.B.Fernandez, R.C.Summers, and C. Wood, *Database security and integrity*, Addison-Wesley 1981.
- [Fer93] E.B.Fernandez,M.M.Larrondo-Petrie and E.Gudes, "A method-based authorization model for object-oriented databases", *Proc. of the OOPSLA 1993 Workshop on Security in Object-oriented Systems* , 70-79.
- [Fer94] [E. B. Fernández](#), E. Gudes, [H. Song](#): A Model for Evaluation and Administration of Security in Object-Oriented Databases. [IEEE Trans. Knowl. Data Eng.](#) **6**(2): 275-292 (1994)
- [Fer94a] E. B. Fernandez, J. Wu, and M. H. Fernandez, "User group structures in object-oriented databases", *Proc. 8th Annual IFIP W.G.11.3 Working Conference on Database Security*, Bad Salzdetfurth, Germany, August 1994.
- [Grif76] P. Griffiths, [B. W. Wade](#): An Authorization Mechanism for a Relational Database System. [ACM Trans. Database Syst.](#) **1**(3): 242-255 (1976)
- [Gus98] D. Gusfield, A Graph Theoretic Approach to **Statistical** Data Security , *Siam J. of Computing*, Volume 17 , Issue 3 (June 1988) 552 – 571
- [Han2000] Jiawei Han, [Micheline Kamber](#): Data Mining: Concepts and Techniques Morgan Kaufmann 2000
- [Hac02] [H. Hacigümüs](#), [B. R. Iyer](#), [Chen Li](#), S. Mehrotra: Executing SQL over encrypted data in the database-service-provider model. [SIGMOD Conference 2002](#): 216-227
- [Hun04] A. Hundepool: The ARGUS Software in the CASC-Project.,*Proceedings of [Privacy in Statistical Databases 2004](#)*: Barcelona, Spain, 2004, 323-335
- [Jaj90] S. Jajodia, [B. Kogan](#): Integrating an Object-Oriented Data Model with Multilevel Security. [IEEE Symposium on Security and Privacy 1990](#): 76-85
- [Jaj90a] S. Jajodia, [B. Kogan](#): Transaction Processing in Multilevel-Secure Databases Using Replicated Architecture. [IEEE Symposium on Security and Privacy 1990](#): 360-368
- [Jaj91] S. Jajodia, [R. S. Sandhu](#): Towards a Multilevel Secure Relational Data Model. *Proceedings of [SIGMOD Conference 1991](#)*: 50-59
- [Lar90] M. M. Larrondo-Petrie, E. Gudes, H. Song, E. B. Fernandez, "Security Policies in Object-Oriented Databases," in *Database Security III: Status and Prospectus*, D.L. Spooner and C. Landwehr (Eds.), Elsevier Science Publishers (North-Holland) 1990, 257-268.

- [Lunt90] T. F. Lunt, [D. E. Denning](#), [R. R. Schell](#), [M. Heckman](#), [W. R. Shockley](#): The SeaView Security Model. [IEEE Trans. Software Eng.](#) **16**(6): 593-607 (1990)
- [Medina2009] [Carlos Blanco](#), Eduardo Fernández-Medina, [Juan Trujillo](#), [Mario Piattini](#): Data Warehouse Security. [Encyclopedia of Database Systems 2009](#): 675-679
- [Pernul2001] [Torsten Priebe](#), Günther Pernul: A Pragmatic Approach to Conceptual Modeling of OLAP Security. [ER 2001](#): 311-324
- [Rab91] [F. Rabitti](#), [E. Bertino](#), [W. Kim](#), D. Woelk: A Model of Authorization for Next-Generation Database Systems. [ACM Trans. Database Syst.](#) **16**(1): 88-131 (1991)
- [Ram2003] R. Ramakrishnan, J. Gehrke, *Database management systems*, McGraw-Hill, 3rd edition, 2003.
- [salt2001] [Marta Oliva](#), Fèlix Saltor: Maintaining the Confidentiality of Interoperable Databases with a Multilevel Federated Security System. [DBSec 2001](#): 269-282
- [San96] R. S. Sandhu, [E. J. Coyne](#), [H. Feinstein](#), [C. E. Youman](#): Role-Based Access Control Models. [IEEE Computer](#) **29**(2): 38-47 (1996)
- [Sic83] [G. L. Sicherman](#), [W. de Jonge](#), R. P. van de Riet: Answering Queries Without Revealing Secrets. [ACM Trans. Database Syst.](#) **8**(1): 41-59 (1983)
- [Spo84] D.L.Spooner and E. Gudes, "A unifying approach to the design of a secure database operating system", *IEEE Trans. on Soft. Eng.*, vol. SE-10, No. 3, May 1984, 310-319.
- [Ston76] M. Stonebraker, [E. Wong](#), [P. Kreps](#), [G. Held](#): The Design and Implementation of INGRES. [ACM Trans. Database Syst.](#) **1**(3): 189-222 (1976)

6.13 EXERCISES

1. Using the database in Figure 6.1, define two different views in SQL, and show two different possible update problems on this views.
2. Give examples for three different integrity constraints in SQL for the database in Figure 6.1.
3. Give examples for security or privacy problems which are unique to databases.
4. Give examples to security policies which are most relevant to databases.
5. Using the rules for Ingres in Section 6.4, show the result of the query: *Retrieve (E.DPT)*.
6. Show a set of timestamps and a Revoke command that will transfer the graph in Figure 6.9c to the graph in Figure 6.9d..

7. Assume the following set of GRANT commands:

A GRANT READ,INS,DEL	ACCESS TO B WITH GRANT OPTION AT TIME 10
B GRANT READ,INS	ACCESS TO D WITH GRANT OPTION AT TIME 20
B GRANT READ,DEL	ACCESS TO C WITH GRANT OPTION AT TIME 30
C GRANT READ,DEL	ACCESS TO D WITH GRANT OPTION AT TIME 40
D GRANT READ	ACCESS TO E AT TIME 50
A GRANT READ,DEL	ACCESS TO C WITH GRANT OPTION AT TIME 60

Show the authorization graph and table SYSAUTH at this stage.

Now show the result of the following operation:

A REVOKES ALL RIGHTS FROM B (CASCADE IN SQL)

Show the SYSAUTH table after this operation, and explain the differences between System R results and SQL results.

8. Assume Joe gave Select right on table Employee to Bill. Will Bill be able to define a new table with a foreign key constraint to Employees? Explain when yes and when no.

9. Assume Joe gave Select right on table Employee to Bill. Bill defined a View V1 on the table and gave select right to Bob. Bob defined a new view V2 on V1 and gave select right to Mary. Later Joe added the Insert right on Employee to Bill. Show a sequence of commands that will cause Mary: a) to also get the Insert command b) not getting it

10. Using the relevant companies literature, compare in details the behavior of GRANT/REVOKE commands in two different commercial products, and their differences with SQL.

11. Read the paper [Bert99] and discuss in details the new features of the model vs. SQL

12. Using the Internet references, discuss in detail the new security features in SQL-99 over SQL-92.

13. Give a detailed example for the use of the ROLE-based model in a database. Give exact SQL-99 commands to implement your example.

14. Read paper [Fer90] and explain in detail the way the expected results are derived for queries Q1 and Q2 in Section

15. Provide an individual tracker to compute the salary of Wond in table 6.5

16. Using table 6.1, show the results of the following two statements in a row, both by user with label S:

a) INSERT into EMP
Values Sam, Dept1, 10K

b) Update Emp
SET Salary = "20K"
Where Name = "Ann"

17. Implement the Grant/Revoke algorithm of System R. Build a suitable user-interface and solve question 7 with it.

18. Implement a multi-level relational database by adding suitable attributes using MS-Access or another similar product.
19. Implement the general tracker algorithm based on paper [Den80] and apply it to a sample database of your choice.
20. You are the DBA for the Security_Startup Company, and you create a relation called Employees with attributes E_id (primary key), Ename , Dept, Age and Salary. For authorization reasons, you also define views EmployeeNames (with Ename as the only attribute) and DeptInfo with fields dept and avgsalary. The latter lists the average salary for each department.
 1. Show the view definition statements for EmployeeNames and DeptInfo.
 2. Give an example of a view update on the above schema that cannot be implemented through updates to Employees
 3. What privileges should be granted to a user who needs to know only average department salaries for the Marketing and Research departments?
 4. You want to authorize your secretary Joe to fire people (but not to hire people) (you'll probably tell him whom to fire, but you want to be able to delegate this task), to check on who is an employee, and to check on average department salaries. What privileges should you grant to the secretary?
 5. You want to give your secretary the authority to allow other people to read the EmployeeNames view. Show the appropriate command.
 6. Your secretary defines two new views using the EmployeeNames view. The first is called AtoSNames and simply selects names that begin with a letter in the range A to S. The second is called AgesbyDept and displays the average age of employees in each department. Next you decide to give your secretary the right to insert tuples into the EmployeeNames view. Show the appropriate command, and describe what privileges your secretary has after this command is executed to the newly created views by him.
 7. Your secretary allows Todd to read the EmployeeNames relation and later quits. You then revoke the secretary's privileges. What happens to Todd's privileges?
 8. Your secretary wants to define a new table called young_employees with a foreign key constraints to employees. Show the relevant commands needed.
 9. You go on a trip. when you come back you see that your secretary Joe has been quite busy. He has defined a view called AllNames using the view EmployeeNames, and given his secretary Bob the right to read from the AllNames view. Bob has passed this right on to his friend Susan. You decide that even at the cost of annoying Joe by revoking some of his privileges, you simply have to take away Bob and Susan's rights to see your data. What REVOKE statement would you execute? What rights does Joe have on Employees after this statement is executed? What views are dropped as a consequence?
21. Many companies have a set of databases, maybe from different vendors. Indicate a way to access securely a set of corporate databases from a smart phone. By securely we mean that authorized employees can perform specific accesses to information in the databases.