Part II: Database Security

Database management system (DBMS): a suite of programs for constructing and maintaining the database and for offering ad hoc query facilities to multiple users and applications. A query language provides a uniform interface to the database for users and applications.

Figure **??** provides a diagram of a simplified DBMS architecture. Developers use DDL (data definition language) to define the database logical structure and procedural properties, which are presented by a set of database description tables. A data manipulation language (DML) provides a powerful set of tools for application developers. Query language are declarative languages designed to support ean users. The database description tables are used to to manage the physical database. The interface to the database is through a file manager module and a transaction manager module. The DBMA uses authorization tables to ensure the user has permission to execute the query language statement on the database. The concurrent access table prevents conflicts when simultaneous, conflicting commands are executed.

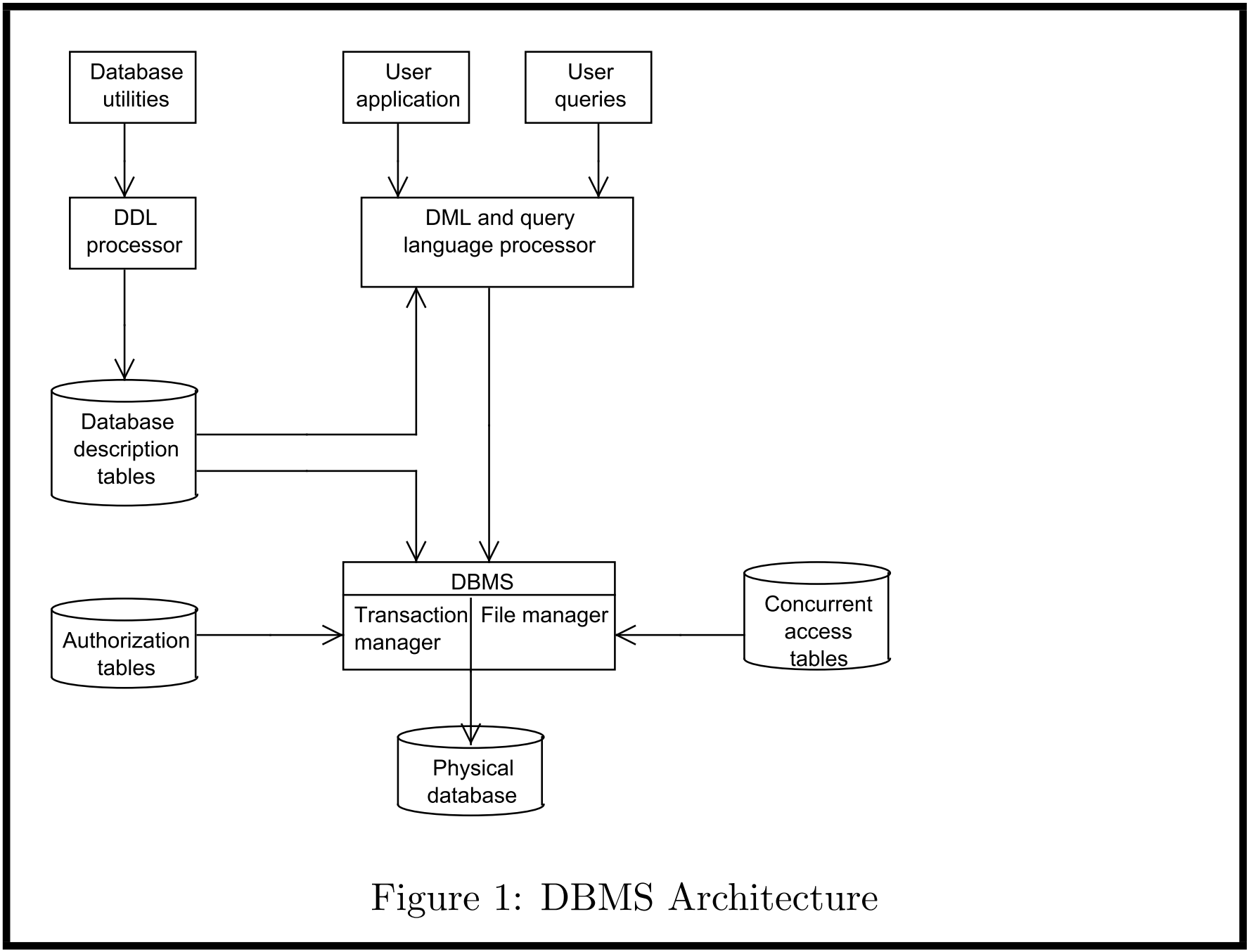

Figure 1: DBMS Architecture

Relational database:

- The basic building block of a database is a table of data, consisting of rows and columns, similar to a spreadsheet. Drawback: possible blank cells and difficult to add new columns. The relational database structure enables the creation of multiple tables tied together by a unique identifier that is present in all tables.
- The basic building block of a relational database is a relation, which is a flat table. Rows are referred to as tuples and columns are referred to as attributes. A primary key is used to uniquely identify a row in a table. The primary key consists of one or more column names.
- To create a relationship between two tables, the attributes that define the primary key in one table must appear as attributes in another table, where they are referred as a foreign key.

Did	Dname	Dacctno
4	account	528221
13	education	202035
15	services	223945
...	...	...

Ename	Did	S code	Eid	Ephone
Robin	15	23	2345	6127092485
Neil	13	12	5088	6127092246
Code	15	22	9664	6127093148
Holly	4	26	7712	6127099348
...	...	...	...	...

Table 1: Two tables in a relational database

In Table **??**, one table is Department Table and another is Employee Table. Did is the primary key of Department Table. Eid is the primary key of Employee Table, while Did is the foreign key of Employee Table. A foreign key value can appear multiple times in a table.

- A view is a virtual table which is a result of a query. Table **??** is an example of a view. Views are often used for security purpose. A view can provide restricted access to a relational database so that a user only has access to certain rows or columns.

Dname	Ename	Eid	Ephone
service	Robin	2345	6127092485
education	Neil	5088	6127092246
service	Code	9664	6127093148
account	Holly	7712	6127099348
...	...	...	... ...

Table 2: A view derived from the database

- Structured query language: There are several versions of the ANSI/ISO standard and a variety of different implementations of SQL, but the basic syntax and semantics are the same.

Example:

```
CREATE TABLE department (
  Did INTEGER PRIMARY KEY,
  Dname CHAR (30),
  Dacctno CHAR(6) )

CREATE TABLE employee (
  Ename CHAR (30),
  Did INTEGER,
  Scode INTEGER,
  Eid INTEGER PRIMARY KEY,
  Ephone CHAR (10),
  FOREIGN KEY (Did) PREFERENCES department (Did) )
```

The basic command for retrieving information is the SELECT statement. The following query returns the Ename, Did and Ephone fields from the Employee table for all employees assigned to department 15.

```
SELECT Ename, Eid, Ephone
  FROM Employee
  WHERE Did = 15
```

The view in Figure **??** is created using the following statement:

```
CREATE VIEW newtable (Dmane, Ename, Eid, Ephone)
AS SELECT D.Dmane, E.Ename, E.Eid, E.Ephone
FROM Department D Employee E
WHERE E.Did = D.Did
```

Other common commands are `INSERT`, `UPDATE`, `DELETE`, `AND`, `OR`, `UNION` etc.

```
DELETE FROM employee WHERE Scode < 22

INSERT INTO employee
VALUES  ('David', 4, 25, 4576, '6127099234')

UPDATE employee
SET Scode=25
WHERE Emane='Holly'
```

Google big tables: Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. A simple data model provided by Bigtable gives clients dynamic control over data layout and format.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases and main-memory databases have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings.

Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. And Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes. The row keys in a table are arbitrary strings. Every read or write of data under a single row key is atomic (regardless of the number of different columns being read or written in the row), a design decision that makes it easier for clients to reason about the system's behavior in the presence of concurrent updates to the same row.

Column keys are grouped into sets called column families, which form the basic unit of access control. All data stored in a column family is usually of the same type. A column family must be created before data can be stored under any column key in that family; after a family has been created, any column key within the family can be used. In contrast, a table may have an unbounded number of columns. Each cell in a Bigtable can contain multiple versions of the same data; these versions are indexed by timestamps.

Data warehouse: A centralized data warehouse is usually used to store enterprise data. The data are organized based on a star schema, which usually has a fact table with part of the attributes called dimensions and the rest called measures. Each dimension is associated with a dimension table indicating a dimension hierarchy. The dimension tables may contain redundancy, which can be removed by splitting each dimension table into multiple tables, one per attribute in the dimension table. The result is called a snowflake schema.

The following is a simple example of star schema.

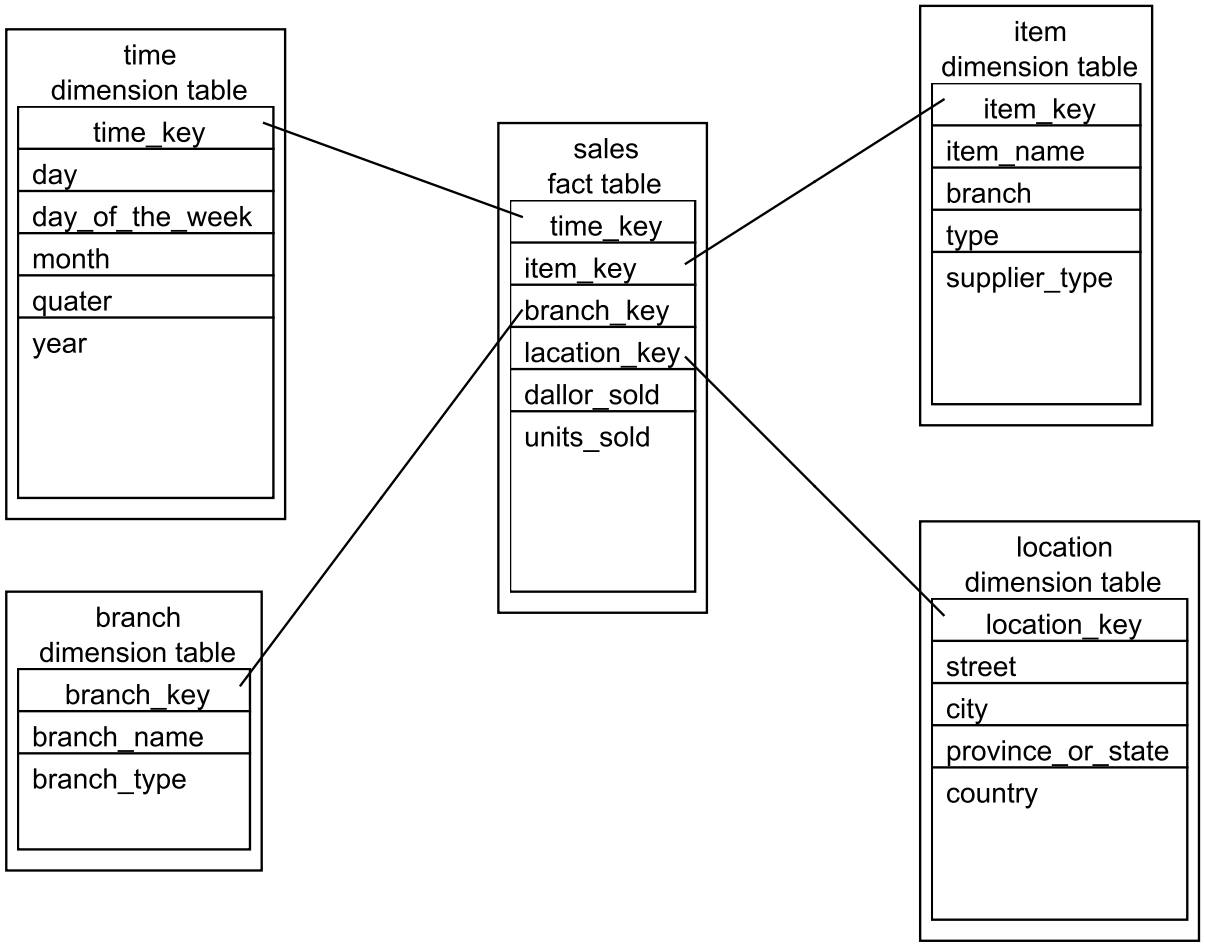

Figure 2: Star schema of a data warehouse for sales

A data warehouse usually stores data collected from multiple data sources, such as transactional databases throughout an organization. The data are cleaned and transformed to a common consistent format before they are stored in the data warehouse. Subsets of the data in a data warehouse can be extracted as data marts to meet the specific requirements of an organizational division. Unlike in transactional databases where data are constantly updated, typically the data stored in a data warehouse are refreshed from data sources only periodically.

OLAP stands for On-Line Analytical Processing. Unlike statistical databases which usually store census data and economic data, OLAP is mainly used for analyzing business data collected from daily transactions, such as sales data and health care data. The main purpose of an OLAP system is to enable analysts to construct a mental image about the underlying data by exploring it from different perspectives, at different level of generalizations, and in an interactive manner.

Popular architectures of OLAP systems include ROLAP (relational OLAP) and MOLAP (multidimensional OLAP). ROLAP provides a front-end tool that translates multidimensional queries into corresponding SQL queries to be processed by the relational backend. MOLAP does not rely on the relational model but instead materializes the multidimensional views. Using MOLAP for dense parts of the data and ROLAP for the others leads to a hybrid architecture, namely, the HOLAP or hybrid OLAP.

As a component of decision support systems, OLAP interacts with other components, such as data mining, to assist analysts in making business decisions. While data mining algorithms automatically produce knowledge in a pre-defined form, such as association rule or classification. OLAP does not directly generate such knowledge, but instead relies on human analysts to observe it by interpreting the query results. On the other hand, OLAP is more flexible than data mining in the sense that analysts may obtain all kinds of patterns and trends rather than only knowledge of fixed forms.

OLAP and data mining can also be combined to enable analysts in obtaining data mining results from different portion of the data and at different level of generalization.

The requirements on OLAP systems have been defined differently, such as the FASMI (Fast Analysis of Shared Multidimensional Information) test and the Codd rules.

- To make OLAP analysis an interactive process, the OLAP system must be highly efficient in answering queries. OLAP systems usually rely on extensive pre-computations, indexing, and specialized storage to improve the performance.
- To allow analysts to explore the data from different perspectives and at different level of generalization, OLAP organizes and generalizes data along multiple dimensions and dimension hierarchies. The data cube model is one of the most popular abstract models for this purpose.

Data cube was proposed as a SQL operator to support common OLAP tasks like histograms and sub-totals. Even though such tasks are usually possible with standard SQL queries, the queries may become very complex. The number of needed unions is exponential in the number of dimensions of the base table. Such a complex query may result in many scans of the base table, leading to poor performance. Because sub-totals are very common in OLAP queries, it is desired to define a new operator for the collection of such sub-totals, namely, data cube.

Figure **??** displays the lattice of cuboids, making up a 4-D data cube for the dimensions time, item, location, and supplier. Each cuboid represents a different degree of summarization. The apex cuboid (0-D), typically denoted by all, summarized over all four dimensions. The 4-D cuboid is the base cuboid.

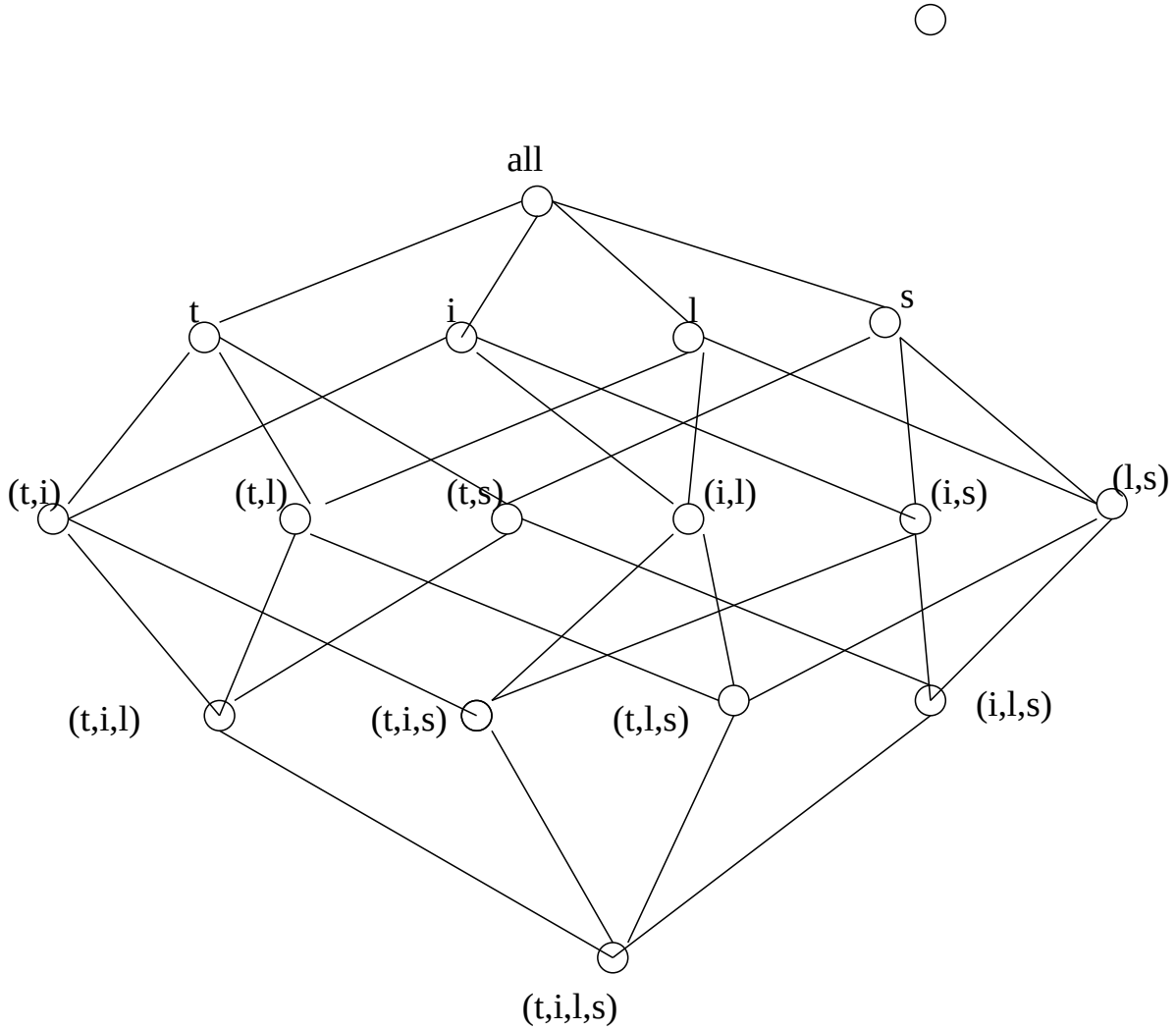

Figure 3: Lattice of cuboids, making up a 4-D datacube

Outsourced database

Recently, there has been growing interest in outsourcing database services in both the commercial world and the research community. Although the outsourced database service model is emerging as an efficient replacement solution for traditional in-house database management systems, its clients, however, have to store their private data at an external service provider, who is typically not fully trusted, and so it introduces numerous security research challenges.

Geospatial database

When the location of facilities is established, spatial coordinates such as latitude and longitude are generated from global positioning systems (GPS) or from Geographic Information Systems (GIS). The information including latitude and longitude is aggregated into a set of database records, the result is the creation of a Geospatial Database. Geospatial databases are very useful when used to support single or multiple well permitting projects. What is really powerful about using geospatial database files is that they are easily shared amongst construction service companies, the operators and with other agencies without the need to send hardcopy or scanned image maps. Using just geospatial database files, any or all of the information in the file can be easily plotted to create a variety of maps as determined by the database record values.

Geospatial data is increasing availability and the tools to integrate and visualize the various types of data facilitate conducting sophisticated analysis and discovering hidden patterns. Therefore, uncontrolled dissemination of geospatial data may have grave consequences for national security and personal privacy. Access control for this data is based on its geospatial location, content and context, the credentials and characteristics of the users requesting access as well as the time at which the data is captured and requested. Since geospatial data is increasingly obtained from third party Web services, the security models presented in the area of geospatial Web services are under researches.

The vector and raster models are two principal spatial data organization scheme.

- The vector model uses the geometry shapes such as points, lines, polygons, etc. These spatial attributes record data about the location, topology and geometry of geospatial data. This model also consists some thematic attributes such as annual rainfall, vegetation type, census tracts, etc.
- Under the raster data model, the spatial data, such as satellite images, elevation maps, or digitized maps, is represented as a grid of columns and rows, i.e. as a matrix of cells (called pixels). Each layer of grid cells records a separate attribute. Each cell carries the non-spatial data too. Spatial coordinates are not usually explicitly stored for each cell, but implicitly represented with the ordering of the pixels. Typically, each layer contains information about the number of rows and columns and the geographic location of the origin.

Hippocratic Databases: Hippocratic databases (HDBs) are a class of database systems that accept responsibility for the privacy and security of information they manage without impeding legitimate use and disclosure. HDBs ensure that only authorized individuals have access to sensitive information and that any disclosure of this information is for proper purposes. They empower individuals to consent to specific uses and disclosures of their information and to verify the enterprises compliance with its privacy policies. HDBs also employ technical safeguards to ensure the security of the information they manage. Further, they use advanced information sharing and analytics to enable enterprises to gain maximum value from information without compromising security or individual privacy.

Active enforcement technology supports the principles of purpose specification, consent, limited use and limited disclosure for relational database and XML-based systems. Compliance auditing, query ranking, and curation auditing technologies support the compliance principle. Sovereign Information Integration enables limited use and limited disclosure in distributed environments with no trusted third parties, while order-preserving encryption and the Partition Plaintext Ciphertext storage model promote the safety principle. The growth of electronic medical and financial records accompanied by many highly publicized privacy breaches in recent years underscore the importance of continuing research in HDB technology.

Database access control

The DBMS operates on the assumption that the computer system has authenticated each user. As an additional line of defence, the computer system may use the overall access control system to determine whether a user may access to the database as a whole. For users that are granted access to the database, a database access control system provides a specific capability that controls access to portions of the database.

Commercial DBMSs provide discretionary or role-based access control. A DBMS can support a range of administrative policies, such as

- Centralized administration: a small number of privileged users may grant and revoke access rights.
- Ownership-based administration: The creator of a table may grant and revoke access right to the table.
- Decentralized administration: The owner of the table may grant and revoke access rights to other uses.

SQL-based access definition

Two commands are used for managing access rights, GRANT and REVOKE. Examples of syntax are as follows:

```
GRANT            {privileges | role}
[ON              table]
TO               {user | role | PUBLIC}
[IDENTIFIED BY   password]
[WITH            GRANT OPTION]
```

This command can be used to grant one or more access rights or can be used to assign a user to a role.

Typically, SQL provides different ranges of access rights, such as: Select, Insert, Update, Delete, References, etc.

The statement: `GRANT SELECT ON ANY TABLE TO rwei`

enables user `rwei` to query any table in the database.

```
REVOKE          {privileges | role}
[ON             table]
FROM            {user | role | PUBLIC}
```

The statement: `REVOKE SELECT ON ANY TABLE FROM rwei`

will revoke the access rights of the preceding example.

The grant option enables an access tight to cascade through a number of users. When user A revokes an access tight, any cascaded access tight is also revoked, unless that access right would exist even if the original grant from A had never occurred.

Figure **??** is an example, where t is the time.

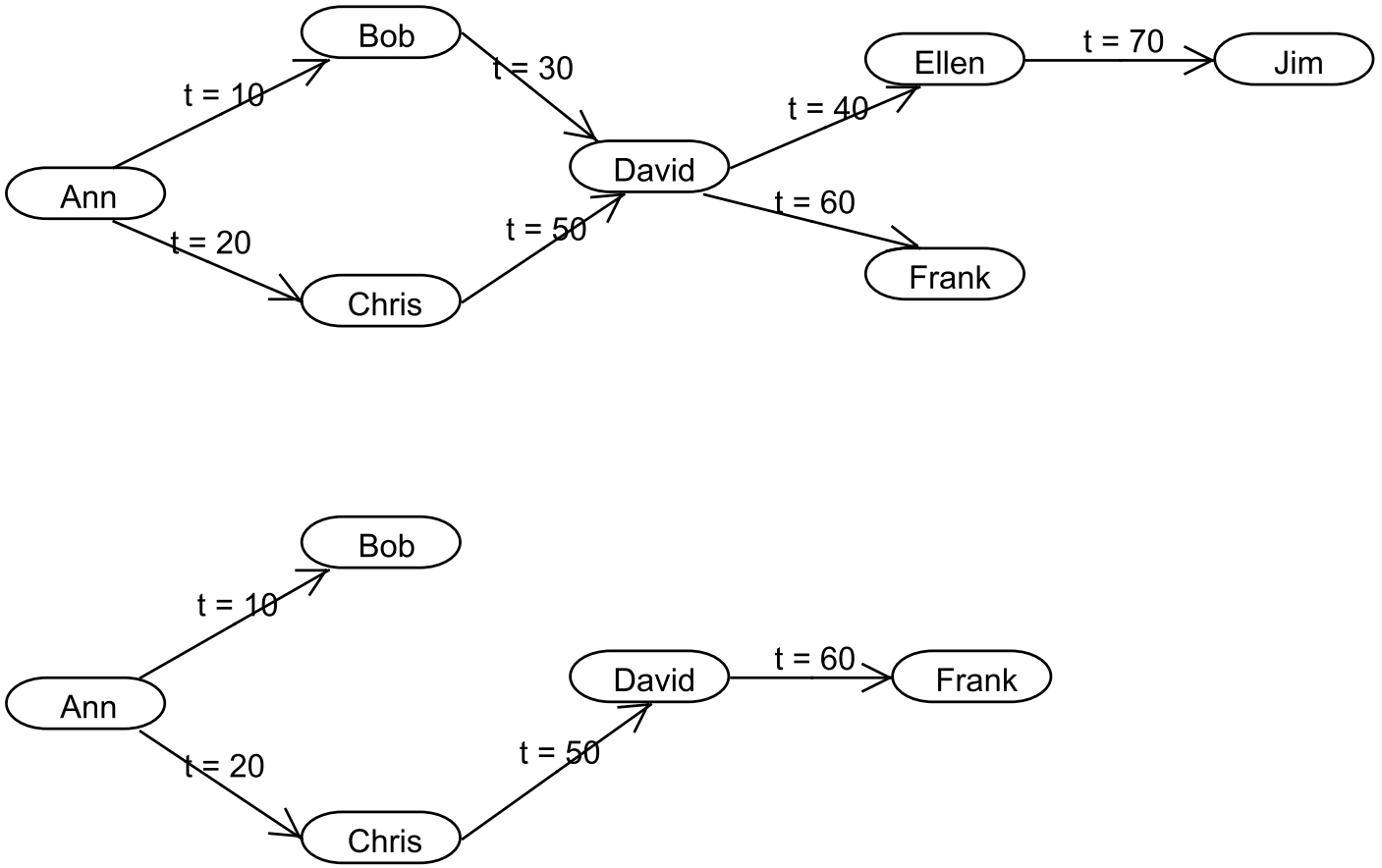

Figure 4: Bob revokes privilege from David

Multilevel security

Multilevel security is of interest when there is a requirement to maintain a resource, such as a file system or database in which multiple levels of data sensitivity are defined.

The addition of multilevel security to a database system increases the complexity of the access control function and of the design of the database itself. The following are possible methods of imposing multilevel security on a relational database, in terms of the granularity of classification.

- Entire database
- Individual tables (relations)
- Individual columns (attributes)
- Individual rows (tuples)
- Individual elements

The granularity of the classification scheme affects the way in which access control is enforced. For example, suppose the following SQL query is issued:

```
SELECT Ename
 FROM employee
 WHERE Salary > 50K
```

If the classification granularity is the entire database or at the table level, then it is straightforward. However, if it is at the column level, then it will not reject only if both the column of salary and column of name are accessible.

When a user with a low clearance (unrestricted) requests the insertion of a row with the same primary key as an existing row where the row or one of its elements is at a higher level. The DBMS has essential three choices:

- Notify the user that a row with the some primary key already exists and reject the insertion. This will cause some inference problem.
- Replace the existing row with the new row classified at the lower level. This will result overwrite data not visible to the user.
- Insert the new row at the lower level without modifying the existing row at the higher level. This is know as polyinstantiation, which will creates a database with conflicting entries.

Classical access control models

1. Discretionary access control

While convenient for their expressiveness and flexibility, in high security settings discretionary access control results limited for its vulnerability to Trojan horses. The reason for this vulnerability is that discretionary access control does not distinguish between users and subjects . A discretionary access control system evaluates the requests made by a subject against the authorizations of the user who generated the corresponding process. It is then vulnerable from processes executing malicious programs that exploit the authorizations of the user invoking them. Protection against these processes requires controlling the flows of information within processes execution and possibly restricting them. Mandatory policies provide a way to enforce information flow control through the use of labels.

2. Mandatory security policies enforce access control on the basis of regulations mandated by a central authority. The most common form of mandatory policy is the multilevel security policy, based on the classifications of subjects and objects in the system. Each subject and object in the system is associated with an access class, usually composed of a security level and a set of categories. Security levels in the system are characterized by a total order relation, while categories form an unordered set. As a consequence, the set of access classes is characterized by a partial order relation, denoted $\geq$ and called dominance. Given two access classes c_1 and c_2, c_1 dominates c_2, denoted $c_1 \geq c_2$, if and only if the security level of c_1 is greater than or equal to the security level of c_2 and the set of categories of c_1 includes the set of categories of c_2. Access classes together with their partial order dominance relationship form a lattice. Mandatory policies can be classified as secrecy-based and integrity-based, operating in a dual manner.

The main goal of secrecy based mandatory policies is to protect data confidentiality. As a consequence, the security level of the access class associated with an object reflects the sensitivity of its content, while the security level of the access class associated with a subject, called clearance, reflects the degree of trust placed in the subject not to reveal sensitive information. The set of categories associated with both subjects and objects defines the area of competence of users and data. The main goal of integrity-based mandatory policies is to prevent subjects from indirectly modifying information they cannot write. The integrity level associated with a user reflects then the degree of trust placed in the subject to insert and modify sensitive information. The integrity level associated with an object indicates the degree of trust placed on the information stored in the object and the potential damage that could result from unauthorized modifications of the information.

A major drawback of mandatory policies is that they control only flows of information happening through overt channels, that is, channels operating in a legitimate way. As a consequence, the mandatory policies are vulnerable to covert channels, which are channels not intended for normal communication but that still can be exploited to infer information. For instance, if a low level subject requests the use of a resource currently used by a high level subject, it will receive a negative response, thus inferring that another (higher level) subject is using the same resource.

3. Role-based access control

A RBAC scheme for database will simplify the administrative work and improve the security. However, not all the database softwares provide RBAC mechanism.

It is important to note that roles and groups of users are two different concepts. A group is a named collection of users and possibly other groups, and a role is a named collection of privileges, and possibly other roles. Furthermore, while roles can be activated and deactivated directly by users at their discretion, the membership in a group cannot be deactivated. The main advantage of RBAC, with respect to DAC and MAC, is that it better suits to commercial environments. In fact, in a company, it is not important the identity of a person for her access to the system, but her responsibilities. Also, the role-based policy tries to organize privileges mapping the organization's structure on the roles hierarchy used for access control.

Credential-based access control

A client and a server, interacting through a predefined negotiation process. The server is characterized by a set of resources. Both the client and the server have a portfolio, which is a collection of credentials (i.e., statements issued by authorities trusted for making them) and declarations (statements issued by the party itself).

Credentials correspond to digital certificates and are guaranteed to be unforgeable and verifiable through the public key of the issuing authority. To the aim of performing gradual trust establishment between the two interacting parties, the server defines a set of service accessibility rules, and both the client and the server define their own set of portfolio disclosure rules.

The service accessibility rules specify the necessary and sufficient conditions for accessing a resource, while portfolio disclosure rules define the conditions that govern the release of credentials and declarations. Both the two classes of rules are expressed by using a logic language.

The information about the state is classified as persistent state, when the information is stored at the site and spans different negotiations, and negotiation state, when it is acquired during the negotiation and is deleted when the same terminates.

A number of trust negotiation strategies have been proposed in the literature, which are characterized by the following steps.

- The client first requests to access a resource.
- The server then checks if the client provided the necessary credentials. In case of a positive answer, the server grants access to the resource; otherwise it communicates the client the policies that she has to fulfill.
- The client selects the requested credentials, if possible, and sends them to the server.
- If the credentials satisfy the request, the client is granted access to the resource.

The main advantage of this proposal is that it maximizes both server and client's privacy, by minimizing the set of certificates exchanged. In particular, the server discloses the minimal set of policies for granting access, while the client releases the minimal set of certificates to access the resource. To this purpose, service accessibility rules are distinguished in prerequisites and requisites. Prerequisites are conditions that must be satisfied for a service request to be taken into consideration (they do not guarantee that it will be granted); requisites are conditions that allow the service request to be successfully granted. Therefore, the server will not disclose a requisite rule until the client satisfies a prerequisite rule.

Policy Composition

In many real word scenarios, access control enforcement needs to take into consideration different policies independently stated by different administrative subjects, which must be enforced as if they were a single policy.

Example: consider an hospital, where the global policy may be obtained by combining together the policies of its different wards and the externally imposed constraints (e.g., privacy regulations).

Policy composition is becoming of paramount importance in all those contexts in which administrative tasks are managed by different, non collaborating, entities.

The main desiderata for a policy composition framework can be summarized as follows.

- Heterogeneous policy support. The framework should support policies expressed in different languages and enforced by different mechanisms.
- Support of unknown policies. The framework should support policies that are not fully defined or are not fully known when the composition strategy is defined. Consequently, policies are to be treated as black-boxes and are supposed to return a correct and complete response when queried at access control time.
- Controlled interference. The framework cannot simply merge the sets of rules defined by the different administrative entities, since this behavior may cause side effects.

- Expressiveness. The framework should support a number of different ways for combining the input policies, without changing the input set of rules or introducing ad-hoc extensions to authorizations.
- Support of different abstraction levels. The composition should highlight the different components and their interplay at different levels of abstraction.
- Formal semantics. The language for policy composition adopted by the framework should be declarative, implementation independent, and based on a formal semantic to avoid ambiguity.

Different solutions have been proposed for combining different policies.

- Focus on the secure behavior of program modules.
- Algebra of security, which is a Boolean algebra that enables to reason about the problem of policy conflict, arising when different policies are combined.
- Meta-policies, which are defined as policies about policies. Metapolicies are used to coordinate the interaction about policies and to explicitly define assumptions about them. Some researcher formalizes the combination of two policies with a function, called policy combiner, and introduces the notion of policy attenuation to allow the composition of conflicting security policies.
- Other approaches are targeted to the development of a uniform framework to express possibly heterogeneous policies.

An algebra for combining security policies together with its formal semantics. A policy, denoted P_i, is defined as a set of triples of the form (s, o, a), where s is a constant in (or a variable over) the set of subjects S, o is a constant in (or a variable over) the set of objects O, and a is a constant in (or a variable over) the set of actions A. Policies of this form are composed through a set of algebra operators whose syntax is represented by the following BNF:

$$\begin{aligned} E &::= \mathbf{id} | E + E | E \& E | E - E | E \wedge C | o(E, E, E) | E * R | T(E) | (E) \\ T &::= \tau \mathbf{id}.E \end{aligned}$$

where **id** is a unique policy identifier, E is a policy expression, T is a construct, called template, C is a construct describing constraints, and R is a construct describing rules. The order of evaluation of algebra operators is determined by the precedence, which is (from higher to lower) $\tau, ., +$ and $\&$ and $-, *$ and $\wedge$.

The semantic of algebra operators is defined by a function that maps policy expressions in a set of ground authorizations (i.e., a set of authorization triples). The function that maps policy identifiers into sets of triples is called environment, and is formally defined as follows.

Definition An environment e is a partial mapping from policy identifiers to sets of authorization triples. By $e[X/S]$ we denote a modification of environment e such that

$$e[X/S](Y) = \begin{cases} S & \text{if } Y = X \\ e(Y) & \text{otherwise} \end{cases}$$

The semantic of an identifier X in the environment e is denoted by $[[X]]e = e(X)$. The operators defined by the algebra for policy composition basically reflect the features supported by classical policy definition systems.

The set of operators together with their semantic is briefly described in the following.

- Addition (+). It merges two policies by returning their union.

$$[[P_1 + P_2]]_e = [[P_1]]_e \cup [[P_2]]_e$$

  Intuitively, additions can be applied in any situation where accesses can be authorized if allowed by any of the component policies (maximum privilege principle).

- Conjunction (&). It merges two policies by returning their intersection.

$$[[P_1 \& P_2]]_e = [[P_1]]_e \cap [[P_2]]_e$$

  This operator enforces the minimum privilege principle.

- Subtraction ($-$). It deletes from a first policy, all the authorizations specified in a second policy.

$$[[P_1 - P_2]]_e = [[P_1]]_e \setminus [[P_2]]_e$$

Intuitively, subtraction operator is used to handle exceptions, and has the same functionalities of negative authorizations in existing approaches.

- Closure ($*$). It closes a policy under a set of derivation rules.

$$[[P * R]]_e = closure(R, [[P]]_e)$$

The *closure* of policy P under derivation rules R produces a new policy that contains all the authorizations in P and those that can be derived evaluating R on P, according to a given semantics. The derivation rules in R can enforce, for example, an authorization propagation along a predefined subject or object hierarchy.

- Scoping Restriction ($\wedge$). It restricts the applicability of a policy to a given subset of subjects, objects, and actions of the system.

$$[[P \wedge c]]_e = \{t \in [[P]]_e |\ t \text{ satisfy } c\}$$

  where c is a condition. It is useful when administration entities need to express their policy on a confined subset of subjects and/or objects (e.g., each ward can express policies about the doctors working in the ward).

- Overriding (o). It overrides a portion of policy P_1 with the specifications in policy P_2; the fragment that is to be substituted is specified by a third policy P_3.

$$[[o(P_1, P_2, P_3)]]_e = [[(P_1 - P_3) + (P_2 \& P_3)]]_e$$

- Template (τ). It defines a partially specified (i.e., parametric) policy that can be completed by supplying the parameters.

$$[[\tau X.P]]_e(S) = [[P]]_{e[S/X]}$$

where S is the set of all policies, and X is a parameter. Templates are useful for representing policies as black-boxes. They are needed any time when some components are to be specified at a later stage. For instance, the components might be the result of a further policy refinement, or might be specified by a different authority.

Due to the formal definition of the semantic of algebra operators, it is possible to exploit algebra expressions to formally prove the security properties of the obtained (composed) policy. Once the policies have been composed through the algebraic operators described above, for their enforcement it is necessary to provide executable specifications compatible with different evaluation strategies.

Three main strategies to translate policy expressions into logic programs.

- Materialization. The expressions composing policies are explicitly evaluated, by obtaining a set of ground authorizations that represents the policy that needs to be enforced. This strategy can be applied when all the composed policies are known and reasonably static.
- Partial materialization. Whenever materialization is not possible since some of the policies to be composed are not available, it is possible to materialize only a subset of the final policy. This strategy is useful also when some of the policies are subject to sudden and frequent changes, and the cost of materialization may be too high with respect to the advantages it may provide.

- Run-time evaluation. In this case no materialization is performed and runtime evaluation is needed for each request (access triple), which is checked against the policy expressions to determine whether the triple belongs to the result.

A method ($pe2lp$) for transforming algebraic policy composition expressions into a logic program. The method proposed can be easily adapted to one of the three materialization strategies introduced above. Basically, the translation process creates a distinct predicate symbol for each policy identifier and for each algebraic operator in the expression. The logic programming formulation of algebra expressions can be used to enforce access control. As already pointed out while introducing algebra operators, this policy composition algebra can also be used to express simple access control policies, such as open and closed policy, propagation policies, and exceptions management.

For example, let us consider a hospital composed of three wards, namely Cardiology, Surgery, and Orthopaedics. Each ward is responsible for granting access to data under its responsibility. Let $P_{Cardiology}, P_{Surgery}$ and $P_{Orthopaedics}$ be the policies of the three wards. Suppose now that an access is authorized if any of the wards policies state so and that authorizations in policy $P_{Surgery}$ are propagated to individual users and documents by classical hierarchy-based derivation rules, denoted R_H. In terms of the algebra, the hospital policy can be represented as follows.

$$P_{Cardiology} \& P_{Surgery} * R_H \& P_{Orthopaedics}.$$

Inference

Inference is the process of performing authorized queries and deducing unauthorized information from the legitimate responses received. Inference problem arises when the combination of a number of data item is more sensitive than the individual items, or when a combination of data item can be used to infer data of a higher sensitivity. The attacker may use nonsensitive data as well as metadata to infer sensitive information. Metadata refers to knowledge about correlations of dependencies among data items that can be used to deduce information not otherwise available to a particular user. The information transfer path by which unauthorized data in obtained is referred to as an inference channel.

Two inference techniques can be used to derive additional information: analyzing functional dependencies between attributes within a table, or across tables, and merging views with the same constraints.

Name	Position	Salary($)	Department	Dept.Manager
Andy	senior	43,000	strip	Cathy
Calvin	junior	35,000	strip	Cathy
Cathy	senior	48,000	strip	Cathy
Dennis	junior	38,000	panel	Herman
...		...		...

Table 3: Employee table

Suppose general user are not authorized to access the relationship between Name and Salary in the Employee table (Figure **??**). However, users can perform the following two queries:

```
CREATE view V1 AS
SELECT positon, Salary
FROM Employee
WHERE Department = "strip"

CREATE view V2 AS
SELECT Name, Department
FROM Employee
WHERE Department = "strip"
```

Position	Salary($)
senior	43,000
junior	35,000
senior	48,000

Name	Department
Andy	strip
Calvin	strip
Cathy	strip

Table 4: Two views

The result views are displayed in Table **??**. If a user who knows the structure of the Employee table and who knows that the view tables maintain the same order as the Employee table is then able to merge the two views to construct a table show in Figure **??**. This violates the access control policy that the relationship of attributes Name and Salary must not be disclosed.

Name	Position	Salary($)	Department
Andy	senior	43,000	strip
Calvin	junior	35,000	strip
Cathy	senior	48,000	strip

Table 5: Combination of two views

In general, there are two approaches to dealing with the threat of disclosure by inference:

- Inference detection during database design. This approach removes inference channels by altering the database structure (such as split a table) or by changing the access control regime. Techniques in this category often result in unnecessary stricter access controls that reduce availability.
- Inference detection at query time. This approaches seeks to eliminate an inference channel violation during a query or series of queries. If an inference channel is detected, the query is denied or altered. This method usually needs to record user query history.

In both approaches, the important thing is that some detection algorithms for inference channels are needed. This is a difficult problem. The inference problem is even more difficult if we consider multiple databases with metadata.

ID	QI			SV
	Zip Code	Age	Nationality	Condition
1	13053	28	Russian	Heart Disease
2	13068	29	American	Heart Disease
3	13068	21	Japanese	Viral Infection
4	13053	23	American	Viral Infection
5	14853	50	Indian	Cancer
6	14853	55	Russian	Heart Disease
7	14850	47	American	Viral Infection
8	14850	49	American	Viral Infection
9	13053	31	American	Cancer
10	13053	37	Indian	Cancer
11	13068	36	Japanese	Cancer
12	13068	35	American	Cancer

Figure 5: Inpatient microdata

Figure **??** shows medical records from a fictitious hospital. Although the name of patients are omitted in the table, people can still inference some information about the sensitive value of the disease condition. For example, if we know someone at the hospital who is less than 30 years old, then we know that his disease is not cancer.

This example shows that even we hid some identities from the database, it is still possible leaking some sensitive data. Some data anonymization techniques are used to prevent some inference problems. We will discuss them later.

Statistical database

A statistical database (SDB) is one that provides data of a statistical nature, such as counts and averages.

- Pure statistical database: only stores statistical data, such as a census database.
- Ordinary database with statistical access: contains individual entries. The database supports a population of nonstatistical users who are allowed access to selected portions of the database. In addition, the database supports a set of statistical users who are only permitted statistical queries. For the latter users, aggregate statistics based on the underlying raw data are generated in response to a user query, or may be pre-calculated and stored as part of the database.

The access control objective for an SDB system is to provide users with the aggregate information without compromising the confidentiality of any individual entity represented in the database. The database administrator must prevent or at least detect, the statistical user who attempts to gain individual information through one or a series of statistical queries.

An example the following is a database containing 13 confidential records of students in a university that has 50 departments.

(a) Database with statistical access with $N = 13$ students

Name	Sex	Major	Class	SAT	GP
Allen	Female	CS	1980	600	3.4
Baker	Female	EE	1980	520	2.5
Cook	Male	EE	1978	630	3.5
Davis	Female	CS	1978	800	4.0
Evans	Male	Bio	1979	500	2.2
Frank	Male	EE	1981	580	3.0
Good	Male	CS	1978	700	3.8
Hall	Female	Psy	1979	580	2.8
Iles	Male	CS	1981	600	3.2
Jones	Female	Bio	1979	750	3.8
Kline	Female	Psy	1981	500	2.5
Lane	Male	EE	1978	600	3.0
Moore	Male	CS	1979	650	3.5

(b) Attribute values and counts

Attribute A_j	Possible values	$\lvert A_j \rvert$
Sex	Male, Female	2
Major	Bio, CS, EE, Psy,...	50
Class	1978, 1979, 1980, 1981	4
SAT	$310, 320, \ldots 790, 800$	50
GP	$0.0, 0.1, 0.2, \ldots, 3.9, 4.0$	41

Statistics are derived from a database by means of a characteristic formula, C, which is a logical formula over the values of attributes. A characteristic formula users the operations OR, AND, and NOT $(+, \cdot, \sim)$, written here in order of increasing priority. A characteristic formula specifies a subset of the records in the database. For example, the formula

$$(Sex = Male) \cdot ((Major = CS) + (Major = EE))$$

specifies all male students majoring in either CS or EE. For numerical attributes, relational operators may be used. For example, $(GP > 3.7)$ specifies all students whose grade point average exceeds 3.7. For simplicity, attribute names are omitted in what follows. Thus, the preceding formula becomes $Male \cdot (CS + EE)$.

The `query set` of characteristic formula C, denoted as $X(C)$, is the set of records matching that characteristic. For example, for $C = Female \cdot CS$, the set $X(C)$ consists of records 1 and 4, the records of Allen and Davis.

A statistical query is a query that produces a value calculated over a query set. Table **??** lists some simple statistics that can be derived from a query set. Examples, count(Female·CS) = 2; sum(Female·CS,SAT) = 1400.

Name	Formula	Description
count(C)	$\|X(C)\|$	Number of records in the query set
sum(C, A_j)	$\sum_{i \in X(C)} x_{ij}$	Sum of the values of numerical attribute A_j over all the records in $X(C)$
rfreq(C)	$\frac{\text{count}(C)}{N}$	Fraction of all records that are in $X(C)$
avg(C, A_j)	$\frac{\text{sum}(C, A_j)}{\text{count}(C)}$	Mean value of numerical attribute A_j, over all the records in $X(C)$
median(C, A_j)		The $\lceil \|X(C)\|/2 \rceil$ largest value of attribute over all the records in $X(C)$. Note that when the query set size is even, the median is the smaller of the two middle values.
max(C, A_j)	$\text{Max}_{i \in X(C)}(x_{ij})$	Maximum value of numerical attribute A_j over all the records in $X(C)$
min(C, A_j)	$\text{Min}_{i \in X(C)}(x_{ij})$	Minimum value of numerical attribute A_j over all the records in $X(C)$

Table 6: Some queries of a statistical database

Inference from a statistical database

The inference problem in an SDB is that a user may infer confidential information about individual entities represented in the database. Such an inference is called a compromise.

The compromise is positive if the user deduced the value of an attribute associated with an individual entity and is negative if the user deduces that a particular value of an attribute is not associate with an individual.

For example, the statistic $\text{sum}(EE \cdot Female, GP) = 2.5$ compromises the database if the user knows the Baker is the only female EE student (using metadata). Or by the sequence of two queries:

$\text{count}(EE \cdot Female) = 1$

$\text{sum}(EE \cdot Female, GP) = 2.5.$

Another example: consider a personal database in which the sum of salaries of employees may be queried. Suppose a questioner knows the following information

Salary range for a new systems analyst with BS degree is $[50K, 60K]

Salary range for a new systems analyst with MS degree is $[60K, 70K]

Suppose two new systems analysts are added to the payroll and the change in the sum of the salaries is $130K, Then the questioner knows that both new employees have an MS degree.

In general terms, the inference problem for an SDB can be stated as follows. A characteristic function C defines a subset of records (rows) within the database. A query using C provides statistics. If the subset is small enough, or even a single record, the questioner may be able to infer characteristics of a single individual or a small group.

Two distinct approaches can be used to protect inference attacks for SDB:

1. Query restriction: Rejects a query that can lead to a compromise. The answers provided are accurate.

   Some query restriction techniques:

   - Query size restriction: For a database of size N (number of rows, or records), a query $q(C)$ is permitted only if the number of records that match C satisfies

     $$k \leq |X(C)| \leq N - k,$$

     where k is a fixed integer greater than 1. Note that since $q(C) = q(All) - q(\sim C)$, the upper bound is necessary.

A questioner may divides information of an individual into parts, such that queries can be made based on the parts without violating the query size restriction. The combination of parts is called a tracker if it can be used to track down characteristics of an individual. For example, if the formula $C \cdot D$ corresponds to zero or one record, so that the query count$(C \cdot D)$ is not permitted. Suppose that the formula C can be decomposed into two parts $C = C_1 \cdot C_2$, such that the query sets for both C_1 and $T = (C_1 \cdot \sim C_2)$ satisfy the query size restriction. Then count$(C) =$ count$(C_1) -$ count(T). And count$(C \cdot D) =$ count$(T + C_1 \cdot D) -$ count(T).

- Query set overlap control: A query $q(C)$ is permitted only if

$$|X(C) \cap X(D)| \leq r$$

  for all $q(D)$ that have been answered for this user, where r is a fixed integer. This method may limiting the usefulness of the database. It also cannot prevent the cooperation of several users to compromise the database and a user profile has to be kept up to date.

- Partition: The records in the database are partitioned into a number of mutually exclusive groups. The user may only query the statistical properties of each group as a whole (the user may not select a subset of a group). The drawbacks of this method are that the user's ability to extract useful statistics is reduced, and there is a design effort in constructing and maintaining the partition.s

A general problem with query restriction techniques is that the denial of a query may provide some clues that an attacker can deduce underlying information.

2. Perturbation: Provides answers to all queries, but the answers are approximate. The data in the SDB are modified or the system can generate statistics that are modified from that the original database would provide.
   - Data perturbation: Modify the data to prevent inference.
     - Data swapping. Attribute values are exchanged (swapped) between records in sufficient quantity so that nothing can be deduced from the disclosure of individual records. The swapping is done in such a way that the accuracy of at least low-order statistics is preserved.

D			D'		
Sex	Major	GP	Sex	Major	GP
Female	Bio	4.0	Male	Bio	4.0
Female	CS	3.0	Male	CS	3.0
Female	EE	3.0	Male	EE	3.0
Female	Psy	4.0	Male	Psy	4.0
Male	Bio	3.0	Female	Bio	3.0
Male	CS	3.0	Female	CS	3.0
Male	EE	4.0	Female	EE	4.0
Male	Psy	4.0	Female	Psy	4.0

Table 7: Data swapping

Table **??** is a simple example, transforming the database D into database D'. D' has the same statistics as D for statistics derived from one or two attributes. However, count (Female·Bio·4.0) has the value 1 in D but the value 0 in D'.

– Using the estimated underlying probability distribution of attribute values to modify database. For each confidential or sensitive attribute, determine the probability distribution function that best matches the data and estimate the parameters of the distribution function. Then generate a sample series of data from the estimated density function for each sensitive attribute. Finally, substitute the generated data of the confidential attribute for the original data in the same rank order (smallest value replace the smallest original value).

- Output perturbation: Database is not modified, but the output of a query is modified.
  - Random-sample query: suitable for large database. Suppose a user's query is $q(C)$ that is to return a statistical value. The query set is $X(C)$. Then the system replaces $X(C)$ with a sampled query set, which is a properly selected subset of $X(C)$. The system calculates the requested statistic on the sampled query set and returns the value.
  - Calculate the statistics on the requested query set and then adjusting the answer up or down by a given amount in some systematic or randomizes fashion.

To using perturbation techniques, there is a potential loss of accuracy as well as the potential for a systematic bias in the results. The main challenge in the use of perturbation techniques is to determine the average size of the error to be used. If there is too little error, a user can infer close approximations to protected values. If the error is too great, the resulting statistics may be unusable. For a small database, it is difficult to add sufficient perturbation to hide data without badly distorting the results.

Some researchers reported the following. Assume the size of the database (number of data items or records) is n. If the number of queries from a given source is linear to n, then a substantial amount of noise (at least of order $\sqrt{n}$) must be added to the system in terms of perturbation, to preserve confidentiality. This amount of noise may be sufficient to make the database effective unusable. However, if the number of queries is sublinear (e.g., of order $\sqrt{n}$), then much less noise must be added to the system to maintain privacy. For a large database, limiting queries to a sublinear number may be reasonable.

Database encryption

Encryption is a popular technique for ensuring confidentiality of sensitive data. While data encryption is able to enhance security greatly, it can impose substantial overhead on the performance of a system in terms of data management. Management of encrypted data needs to address several new issues like choice of the appropriate encryption algorithms, deciding the key management architecture and key distribution protocols, enabling efficient encrypted data storage and retrieval, developing techniques for querying and searching encrypted data, ensuring integrity of data etc.

There are some disadvantages to database encryption:

- Key management: Because a database is typically accessible to a wide range of users and a number of applications, providing secure keys to selected parts of the database to authorized users and applications is complex task.
- Inflexibility: When part or all of the database is encrypted, it becomes more difficult to perform record searching.

Encryption can be applied to the entire database, at the record level, at the attribute level, or at the level of the individual field.

We use outsourced database as an example. There are four entities involved.

- Data owner: An organization that produces data.
- User: The entity that presents queries to the system.
- Client: Front end that transforms user queries into queries on the encrypted data stored on the server.
- Server: An organization that receives the encrypted data from a data owner and makes them available for clients.

For the security reason, the database are encrypted, but the encryption/decryption keys are not available for the server. So the data are secure at the server. The client system has a encryption key. A user at the client can retrieve a record form the database as follows.

1. The user issues an SQL query for fields from one or more records with a specific value of the primary key.
2. The query processor at the client encrypts the primary key, modifies the SQL query accordingly, and transmits the query to the server.
3. The server processes the query using the encrypted value of the primary key and returns the appropriate records.
4. The query processor of the client decrypts the data and returns the results.

The above method lacks flexibility. For example, suppose a Employee table contains a salary attribute and the user wishes to retrieve all records for salaries less than $70K. Then the above method does not work.

To provide more flexible queries, the following approach is taken. Each row (record) in the database is encrypted as a block. To assist in data retrieval, attributes indexes are associated with each table.

eid	ename	salary	addr	did
23	Tom	70K	Maple	45
860	Mary	60K	Main	83
320	John	50K	river	50
875	Jerry	55K	Hopewell	92

Table 8: Employee Table

Suppose that the employee ID (eid) values lie in the range [1, 1000]. We divide these values into five partitions: [1, 200], [201,400], [401,600], [601, 800] and [801,1000], and then assign index values 1, 2, 3, 4 and 5, respectively. For the attribute emame, we assign index 1 to values starting with A or B, index 2 to values starting with C and D, and so on. Similar partitioning schemes can be used for each of the attributes. Table **??** shows the result table.

E(K,B)	I(eid)	I(ename)	I(salary)	I(addr)	I(did)
1100100011001010...	1	10	3	7	4
1011010100101010...	5	7	2	7	8
1110010100010101...	2	5	1	9	5
0011100000101001...	5	5	2	4	9

Table 9: Encrypted Employee Table with Indexes

The values in the first column represent the encrypted values for each row in the original table. The remaining columns show index values for the corresponding attribute values. The mapping function between attribute values and index values constitute metadata that are stored at the client and data owner locations but not at the server.

This arrangement provides for more efficient data retrieval. Some randomized index can be used to further secure the database. For example, the eid values could be partitioned by mapping [1, 200], [201, 400], [401, 600], [601, 800] and [801, 1000] into 2, 3, 5, 1 and 4, respectively.

Other features may be added to this scheme. The system could use the encrypted value of the primary key attribute values, or a hash value to encrease the efficiency of accessing records by means of the primary key. Different portions of the database could be encrypted with different keys, so that users would only have access to that portion of the database for which they had the decryption key.

Approaches based on new encryption techniques are also discussed: These techniques can support either arithmetic and/or comparison operators directly on encrypted representation. Encryption techniques that support limited computation without decryption have been explored in cryptographic literature in the past. Amongst the first such technique is the privacy homomorphism (PH) developed that supports basic arithmetic operations. While PH can be exploited to compute aggregation queries at the remote server, it does not allow comparison and, as such, cannot be used as basis for designing techniques for relational query processing over encrypted data.

Some researchers developed a data transformation technique that preserves the order in the original data. Such a transformation serves as an order-preserving encryption and can therefore support comparison operators. Techniques to implement relational operators such as selection, joins, sorting, grouping can be built on top of the order preserving encryption. The encryption mechanism, however, cannot support aggregation at the server. While new cryptographic approaches are interesting, one of the limitation of such approaches has been that they are safe only under limited situations where the adversary's knowledge is limited to the ciphertext representation of data. These techniques have either been shown to break under more general attacks (e.g., PH is not secure under chosen plaintext attack), or the security analysis under diverse types of attacks has not been performed.

For special database, such as text database, XML database, etc., more specific methods are discussed in literatures.

Data anonymization

Given a microdata table T, the objective of privacy preserving publication is to release a distorted version T' of T such that T' does not allow an adversary to confidently derive the sensitive data of any individual, and yet, T' can be used to analyze the statistical patterns significant in T. The existing methods of privacy preserving publication is essentially the integration of an anonymization framework and an anonymization principle. Specifically, a framework describes how anonymization is performed, whereas a principle measures whether a sufficient amount of anonymization has been applied.

Linkning attacks

As a concrete application example, consider that the publisher is a hospital, and T is given in Table **??**. Here, T' has three non-sensitive attributes $A_1^q = Age, A_2^q = Sex, A_3^q = Zipcode$, and a sensitive attribute $A^s = Disease$. The column Name specifies the owners of the tuples, e.g., Tuple 1 indicates that Andy, aged 5, lives in a neighborhood with Zipcode 12000, and he contracted gastric-ulcer. Obviously, Name should not be published along with T, since it explicitly reveals the identities of all individuals.

row#	name	age	sex	zipcode	disease
1	Andy	5	m	12000	gastric ulcer
2	Bill	9	m	14000	dyspepsia
3	Ken	6	m	18000	pneumonia
4	Nash	8	m	19000	bronchitis
5	Sarah	28	f	37000	pneumonia
6	Mary	56	f	33000	flu
7	Jame	26	f	36000	Alzeimer
..	...	...	...	..	...

Table 10: Microdata for a hospital

Let T' be the resulting table after removing Name from T. At first glance, it appears that we can simply release T'. This naive approach fails, because an adversary may combine T' with certain additional information, to recover the owner of a tuple. For instance, imagine that a neighbor of Sarah knows the age 28 of Sarah, and that Sarah has been hospitalized before, and thus, must have a record in T'. Since this neighbor has all the non-sensitive values of Sarah, s/he easily finds out that the last-but-one tuple in T' belongs to Sarah.

The above process exemplifies a type of privacy inferences called linking attacks, where an adversary accurately infers the sensitive value of a victim, via the victim's non-sensitive values.

Since the non-sensitive attributes may be utilized to pinpoint the tuple owned by a person, they are commonly referred to as the *quasi-identifier (QI)* attributes.

In reality, the QI-values of an individual may be acquired by an adversary through several channels. Knowing the victim is an obvious channel, as the earlier example where the adversary is the neighbor of Sarah. Alternatively, an adversary may also obtain the QI-values from an external database, which can be completely separate from T, and its accessibility to the public cannot be controlled by the publisher of T. For instance, a worker in the government may have access to the voter registration list in Table **??**, which includes some of the QI-attributes in the microdata of Table **??**, together with people's names.

name	age	sex	zipcode
Andy	5	m	12000
Bill	9	m	14000
Ken	6	m	18000
Nash	8	m	19000
Sarah	28	f	37000
...	...	...	...

Table 11: Voting registration list

k-anonymity

Given a microdata table T, the generalization anonymization framework replaces each QI-value with a less specific form, such that the QI-values of a tuple become indistinguishable from those of some other tuples. Table **??** demonstrates a generalized version of the microdata in Table **??**. For example, the age 5 of Tuple 1 in table has been generalized to an interval [1, 10] in Table **??**. Notice that Tuples 1 and 2 have exactly the same generalized value on every QI attribute, and therefore, constitute a "QI-group". Formally, a QI group is a group resulting from grouping the tuples in a relation by all the QI attributes. Clearly, Table **??** involves several QI-groups: 1, 2 (indicated by tuple IDs), 3, 4, and so on.

row#	age	sex	zipcode	disease
1	[1,10]	m	[10001,15000]	gastric ulcer
2	[1,10]	m	[10001,15000]	dyspepsia
3	[1,10]	m	[15001,20000]	pneumonia
4	[1,10]	m	[15001, 20000]	bronchitis
5	[21,60]	f	[30001,60000	pneumonia
6	[21,60]	f	[30001,60000]	flu
7	[21,60]	f	[30001,60000]	Alzeimer
...	...	...	..	...

Table 12: 2-anonymous version of Table **??**

Assume that the publisher releases Table **??**. Consider the linking attack launched by the neighbor of Sarah who possesses the QI values 28, F, 37000 of Sarah. To guess which tuples may belong to Sarah, the adversary settles on a fuzzy fact: Sarah may have got flu, pneumonia, or Alzeimer's.

Formally, a generalized table is k-anonymous if each QI-group contains at least k-tuples.In general, a higher k provides stronger protection because, in general, k-anonymity guarantees that an adversary has at most $1/k$ probability of finding out the actual tuple owned by the victim individual. As a trade-off, however, increasing k also brings down the utility of the generalized table, since more information must be lost in generalization.

There are different generalization methods discussed and different metrics to calculate information loss. However, researchers found that the optimal k-anonymity generalization is NP-hard, even for simple information-loss metrics. This fact forces a practitioner to accept an approximate algorithm. However, the existing solutions cannot ensure a small approximation ratio (which varies according to the information-loss norm). In particular, the best known ratio is $O(k)$, which implies that the output of an algorithm may considerably deviate from the optimal quality, when strong privacy protection is required (e.g., $k = 20$). It is also known that, when the number of QI attributes is large, k-anonymous generalization inevitably lose a huge amount of information, even for $k = 2$.

l-diversity

There are some crucial defects for k-anonymity. For example, Figure **??** shows the microdata in Figure **??** has been anonymized for $k = 4$.

ID	QI			SV
	Zip Code	Age	Nationality	Condition
1	130**	2*	*	Heart Disease
2	130**	2*	*	Heart Disease
3	130**	2*	*	Viral Infection
4	130**	2*	*	Viral Infection
5	1485*	> 40	*	Cancer
6	1485*	> 40	*	Heart Disease
7	1485*	> 40	*	Viral Infection
8	1485*	> 40	*	Viral Infection
9	130**	3*	*	Cancer
10	130**	3*	*	Cancer
11	130**	3*	*	Cancer
12	130**	3*	*	Cancer

Figure 6: 4-anonymous inpatient microdata

The sensitive values in last q-block are all "Cancer". That means it is possible for people to use QI values to determine the sensitive attribute. This is called Homogeneity attack. Another kind of attack is called background attack. Now let us consider, once again, the neighbor of Sarah. As explained before, from Table **??**, the neighbor can only figure out that Sarah may have had flu, Alzeimer's, or pneumonia. This conjecture, however, may be further improved, if the neighbor utilizes her/his "background knowledge". For example, s/he may know that a flu-vaccine shot had been offered to all the residents in the neighborhood a month before Sarah was hospitalized. Hence, it is rather unlikely that the hospitalization was caused by flu. Furthermore, obviously, Sarah is too young to get infected with Alzeimer's disease. Thus, the adversary becomes (almost fully) confident that Sarah contracted pneumonia.

l-diversity was exactly motivated by this observation. It requires that, after generalization, every QI-group should contain at least l "well represented" sensitive values. Intuitively, this requirement does not allow an adversary to accurately recover the sensitive value of any individual o, provided that the adversary can exclude up to $l-2$ values (i.e., leaving at least 2 possibilities for o). Thus, with a sufficiently large l, l-diversity can effectively prevent privacy breaches. There are multiple ways to interpret the meaning of "well-represented".

Definition A QI-group fulfills distinctness l-diversity, if it contains at least l different sensitive values.

Although this interpretation can be easily understood, it does not offer strong privacy guarantees from a probabilistic point of view. For example, imagine a QI-group with 1000 tuples, 900 of which carry the same sensitive value HIV, and the remaining 100 tuples have distinct values different from HIV. Clearly, the QI-group satisfies distinctness 101-diversity. Nevertheless, the privacy of HIV patients is poorly preserved. Specifically, consider an adversary who aims at inferring the disease of such a patient o, and has no background knowledge, i.e., s/he cannot exclude any disease before studying the published table. With a random guess, the adversary concludes that o had HIV with probability $900/1000 = 90\%$.

Notice that this process of privacy inference essentially captures homogeneity attacks as a special case; hence, we refer to the process as a probabilistic homogeneity attack. This phenomenon leads to an improved version of l-diversity.

Definition A table fulfills frequency l-diversity if, in each QI-group, at most $1/l$ of the tuples carry the sensitive value.

Frequency l-diversity does not provide adequate protection to background attacks. To understand this, consider a QI-group with 1000 tuples, 500 of which have the sensitive value HIV, 499 tuples have pneumonia, and the remaining tuple carries flu. This QI-group qualifies frequency 2-diversity, since the most frequent value HIV belongs to 50% of the tuples. Let o be an HIV patient. Now, imagine an adversary who knows that this group contains the record of o, and that o does not have pneumonia. As a result, the record of o must be one of the 500 HIV-tuples, or the flu-tuple. At this point, the adversary cannot exclude any other disease; hence, taking a random guess, s/he conjectures that o contracted HIV with an exceedingly high probability $500/501 > 99.8\%$.

The cause of the above problem is as follows: after removing the 2nd frequent sensitive value (i.e., pneumonia) in a QI-group, the most frequent sensitive (HIV) value accounts for an excessively high proportion of the remaining tuples in the group. The implication is that, it is not enough to limit the frequency of the most popular sensitive value with respect to the QI-group size (as is the case in frequency l-diversity). Instead, we should limit the frequency according to the number of remaining tuples, after eliminating those having the 2nd frequent sensitive value.

Carrying the reasoning to the general scenario, if an adversary can exclude at most $l-2$ values, we ought to constrain the frequency of the most sensitive value, with respect to the remaining tuples, after discarding the 2nd, 3rd, ..., $(l-1)$-th most frequent sensitive values. This leads to the next version of l-diversity.

Definition Given a QI-group, use $n_1, n_2, \ldots, n_m$ to denote the number of tuples having the most, 2nd most, ..., the least frequent sensitive values in the group, respectively. The QI-group obeys recursive (c, l)-diversity, if the next inequality holds:

$$n_1 \leq c \cdot (n_l + n_{l+1} + ... + n_m),$$

where c is a certain constant, and l is an integer at most m.

A QI-group obeys (c, l)-diversity, if and only if, after eliminating the tuples with any l different sensitive values, the remaining set of tuples still obeys frequency $c/(c+1)$-diversity. This connection leads to a crucial property: if all the QI-groups in a generalized table satisfy recursive (c, l)-diversity, an adversary can correctly discover the sensitive value of an individual with probability at most $c/(c+1)$, provided that the adversary can preclude at most $l-2$ values as belonging to the victim individual.

The Figure **??** demonstrate a 3-diversity generalization. The q-blocks are divided by a line.

ID	QI			SV
	Zip Code	Age	Nationality	Condition
1	1305*	< 40	*	Heart Disease
4	1305*	< 40	*	Viral Infection
9	1305*	< 40	*	Cancer
10	1305*	< 40	*	Cancer
5	1485*	> 40	*	Cancer
6	1485*	> 40	*	Heart Disease
7	1485*	> 40	*	Viral Infection
8	1485*	> 40	*	Viral Infection
2	1306*	< 40	*	Heart Disease
3	1306*	< 40	*	Viral Infection
11	1306*	< 40	*	Cancer
12	1306*	< 40	*	Cancer

Figure 7: 3-diversity inpatient microdata

Anatomy

So far our discussion has employed generalization as the underlying anonymization framework. In this section, we proceed to introduce another framework: anatomy. As with generalization, anatomy can be combined with k-anonymity and l-diversity. The following analysis focuses on l-diversity, due to its obvious advantages over k-anonymity. In particular, we adopt frequency l-diversity, to avoid the complication of recursive (c, l)-diversity. Accordingly, we will assume that probabilistic homogeneity attacks are the objective of privacy protection.

Although generalization preserves privacy, it often loses considerable information in the microdata, which severely compromises the accuracy of data analysis.

Anatomy overcomes the above defect of generalization, by releasing the exact QI-distribution without compromising the quality of privacy preservation. Specifically, anatomy releases a quasi-identifier table (QIT) and a sensitive table (ST), which separate QI-values from sensitive values. For example, Tables **??** (a) and (b) demonstrate the QIT and ST obtained from the microdata Table **??** (a), respectively.

Construction of the anatomized tables can be (informally) understood as follows. First, we partition the tuples of the microdata into several QI-groups, based on a certain strategy. We use an example in following Table to explain.

tuple ID	age	sex	zipcode	disease
1	23	m	12000	pneumonia
2	27	m	14000	dyspepsia
3	35	m	18000	dyspepsia
4	59	m	19000	pneumonia
5	61	f	37000	flu
6	65	f	33000	gastritis
7	65	f	36000	flu
8	70	f	30000	bronchitis

(a) The microdata

tuple ID	age	sex	zipcode	disease
1	[21,60]	m	[10001,60000]	pneumonia
2	[21,60]	m	[10001,60000]	dyspepsia
3	[21,60	m	[10001,60000]	dyspepsia
4	[21,60]	m	[10001,60000]	pneumonia
5	[61,70]	f	[10001,60000]	flu
6	[61,70]	f	[10001,60000]	gastritis
7	[61,70]	f	[10001,60000]	flu
8	[61,70]	f	[10001,60000]	bronchitis

(b) A 2-diverse table

Here, following the grouping in Table , let us place tuples 1-4 (or 5-8) of Table (b) into QI-group 1 (or 2). Then, we create the QIT. Specifically, for each tuple in Table (a), the QIT includes all its exact QI-values, together with its group membership in a new column Group-ID. However, QIT does not store any Disease value. Finally, we produce the ST Table **??** (b) , which retains the Disease statistics of each QI-group. For instance, the first two records of the ST (to avoid confusion, we use record instead of couple for the data of an ST) indicate that, two tuples of the first QI-group are associated with dyspepsia, and two with pneumonia. Similarly, the next three records imply that, the second QI-group has a tuple associated with bronchitis, two with flu, and one with gastritis.

Anatomy preserves privacy because the QIT does not indicate the sensitive value of any tuple, which must be randomly guessed from the ST. To explain this, consider the adversary who has the age 23 and zipcode 11000 of Bob. Hence, from the QIT (Table **??** (a)), the adversary knows that tuple 1 belongs to Bob, but does not obtain any information about his disease so far. Instead, s/he gets the id 1 of the QI-group containing tuple 1. Judging from the ST (Table **??** (b)), the adversary realizes that, among the 4 tuples in QI-group 1, 50% of them are associated with dyspepsia (or pneumonia) in the microdata. Note that s/he does not gain any additional hints, regarding the exact diseases carried by these tuples. Hence, s/he arrives at the conclusion that Bob could have contracted dyspepsia (or pneumonia) with 50% probability. This is the same conjecture obtainable from the generalized Table (b).

tuple ID	age	sex	zipcode	group ID
1	23	m	12000	1
2	27	m	14000	1
3	35	m	18000	1
4	59	m	19000	1
5	61	f	37000	2
6	65	f	33000	2
7	65	f	36000	2
8	70	f	30000	2

(a) The quasi-identifier table (QIT)

group ID	disease	count
1	dyspepsia	2
1	pneumonia	2
2	bronchitis	1
2	gastritis	1
2	flu	2

(b) The sensitive table (ST)

Table 13: The anatomized table

By announcing the QI values directly, anatomy permits more effective analysis than generalization. Given query for pneumonia and age ≤ 30, we know, from the ST, that 2 tuples carry pneumonia in the microdata, and they are both in QI-group 1. Hence, we proceed to calculate the probability p in the age-zipcode plane. This calculation does not need any assumption about the data distribution in the Age-Zipcode plane, because the distribution is precisely released. Specifically, the QIT shows that tuples 1 and 2 in QI-group 1, leading to the exact $p = 50\%$.

Database watermarking

The motivation for database watermarking is to protect databases, especially those published online (e.g., parametric specifications, surveys, and life sciences data), from tampering and pirated copies. A watermark can be considered to be some kind of information that is embedded into underlying data for tamper detection, localization, ownership proof, and/or traitor tracing purposes.

Database watermarking consists of two basic processes: watermark insertion and watermark detection.

The existing database watermarking schemes can be classified along various dimensions, such as

- Data type: Different schemes are designed for watermarking different types of data, including numerical data and categorical data.
- Distortion to underlying data: While some watermarking schemes inevitably introduce distortions/errors to the underlying data, others are distortion-free.
- Sensitivity to database attacks: A watermarking scheme can be either robust or fragile to database attacks. A scheme is robust (fragile, respectively) if it is difficult to make an embedded watermark undetectable (unchanged, respectively) in database attacks, provided that the attacks do not degrade the usefulness of the data significantly.
- Watermark information: The watermark information that is

embedded into a database can be a single-bit watermark, a multiple-bit watermark, a fingerprint, or multiple watermarks in different watermarking schemes.

- Verifiability: A watermark solution is said to be private if the detection of a watermark can only be performed by someone who owns a secret key and can only be proved once to the public (e.g., to the court). After this one-time proof, the secret key is known to the public and the embedded watermark can be easily destroyed by malicious users. A watermark solution is said to be public if the detection of a watermark can be publicly proved by anyone, as many times as necessary.
- Data structure: Different watermarking schemes are designed to accommodate different structural information of the underlying data, including relational databases (with or without primary keys), data cubes, streaming data, and XML data.

Watermarking numerical data

The fundamental assumption is that the watermarked database can tolerate a small amount of errors: it is acceptable to change a small number of ξ least significant bits in some numeric values; however, the value of data is significantly reduced if a large number of the bits are changed. The basic idea is to ensure that those bit positions contain specific values determined by a secret key K. The bit pattern constitutes a watermark.

For watermark insertion, the scheme scans each tuple r in a relation R and seeds a cryptographically secure pseudo-random sequence generator S with the secret key K in concatenation with the tuples primary key $r.P$. Let S_i be the i-th number generated by S. If S_1 satisfies $(S_1 \bmod \gamma = 0)$, then the current tuple r is selected, otherwise the tuple is ignored, where γ is a watermarking parameter used to control the percentage of tuples being selected. Because S_1 is pseudo-random, roughly η/γ tuples are selected, where η is the total number of tuples in relation R. Then, for each selected tuple, the scheme selects one attribute with index $(S_2 \bmod \nu)$ out of ν watermarkable numerical attributes indexed from 0 to $\nu - 1$. For the selected attribute of a selected tuple, the scheme selects one bit with index $(S_3 \bmod \xi)$ out of ξ least significant bits indexed from 0 to $\xi - 1$, where ξ is a watermarking parameter used to control the error that each numerical value can tolerate.

The scheme then assigns the selected bit of the selected attribute in the selected tuple with a mark value (S_4 mod 2). With a probability of 1/2, the underlying bit value is changed in this process. Due to the use of a cryptographically secure pseudo-random sequence generator, it is computationally infeasible for an attacker, without knowing the secret key, to derive where the watermark bits are embedded, what the mark bits are, and the correlations among the embedded locations and the embedded values.

For watermark detection, the scheme scans all the tuples in a suspicious database relation R', locates the marked bit positions, and computes the mark values at those bit positions exactly as in watermark insertion. To detect a watermark, the scheme compares the computed mark values to the corresponding bit values stored in R'. A watermark is detected if the percentage of matches in such comparison is greater than τ , where $\tau \geq 0.5$ is a parameter that is related to the assurance of the detection process.

Distortion-free watermarking

In this solution, all η tuples in a database relation R are first securely divided into g groups according to a secret key K. A different watermark is embedded and verified in each group independently. As a result, any modifications to the watermarked data can be detected and localized to the group level with high probabilities.

First, a (keyed) tuple hash and a (keyed) primary key hash are computed for each tuple r_i using a HMAC function. The tuple hash values are computed based on a fixed order of attributes. Based on the primary key hash values, all tuples are securely divided into g groups. The grouping is only a virtual operation, which means that it does not change the physical position of the tuples. After grouping, all tuples in each group are sorted according to their primary key hash. Like grouping, the sorting operation does not change the physical position of tuples either. Each group is then watermarked independently.

Algorithm 1 Watermark embedding

For all $k = 1, \ldots g, q_k = 0$

for $i = 1$ to η do

$h_i = HMAC(K, r_i)$ // tuple hash

$h_i^p = HMAC(K, r_i.P)$ // primary key hash

$k = h_i^p \bmod g$

$r_i \rightarrow G_k$ // Virtual operation: assign tuple r_i to group k

q_k++

end for

for $k = 1$ to g do

watermark embedding in G_k // See algorithm 2

end for

Algorithm 2 shows the embedding process in each group. A (keyed) group hash value is computed based on the tuple hash values in a sorted order. A watermark, the length of which is equal to the number of tuple-pairs in the current group, is extracted from the group hash value. To embed the watermark, for each tuple pair, the order of the two tuples are changed or unchanged (physically in the original database) to represent a corresponding watermark bit 1 or 0, where 0 is encoded by the ascendant order and 1 by the descendant order. Since only the order of the tuples is changed, the watermark insertion does not introduce any error to the underlying data.

Algorithm 2 Watermark embedding in G_k

sort tuples in G_k in ascendant order according to their primary key hash values// Virtual operation

$H = HMAC(K, h'_1); H = HMAC(K, H|h'_2); \ldots H =$
$HMAC(K, H|h'_{qk})$ /* group hash, where $h'_i (i = 1, \ldots q_k)$ is the tuple hash of the ith tuple after ordering */

W = extractBits$(H, q_k/2)$ // See function below

for $i = 1, i < q_k, i = i + 2$ do

if $W[i/2] == 1$ then

switch the position of r_i and $r_i + 1$ physically in DB

end if

end for

```
function extractBits(H, ℓ){
   if length(H) ≥ ℓ then
    W = concatenation of first ℓ selected bits from H
/* in most cases, H is longer than ℓ */
   else
    m = ℓ− length(H)
    W = concatenation of H and extractBits(H, m)
   end if
return W}
```

Algorithms 3 and 4 describe the watermark detection process. The primary key hash is computed for each tuple and all tuples are divided into groups. In a group, the tuples are first sorted according to their primary key hash values. Like watermark insertion, the sorting is a virtual operation and does not involve order change of any tuples. Based on the tuple hash of the sorted tuples, a group hash value is computed. Then, a watermark W is extracted from the group hash. The watermark W is the one that is supposed to have been embedded if the underlying data were watermarked. On the other hand, a binary string W' is extracted from the tuples in this group. For every tuple pair, if their tuple hash values are in ascendant order, the corresponding bit in W' is extracted to be zero; otherwise, it is one. If W' matches W, the data in the group are authentic; otherwise, the data in this group have been modified or tampered with.

Algorithm 3 Watermark detection

For all $k = 1, \ldots g, q_k = 0$

  for $i = 1$ to η do

   $h_i = HMAC(K, r_i)$

   $h_i^p = HMAC(K, r_i.P)$

   $k = h_i^p \bmod g$

   $r_i \rightarrow G_k$

   $q_k + +$

  end for

  for $k = 1$ to g do

   watermark verification in G_k // See algorithm 4

  end for

Algorithm 4 Watermark verification in G_k

sort tuples in G_k in ascendant order according to their primary key hash values// Virtual operation

$H = HMAC(K, h'_1); H = HMAC(K, H|h'_2); \ldots H = HMAC(K, H|h'_{qk})$

$W = extractBits(H, q_k/2)$ // See function in algorithm 2

for $i = 1, i < q_k, i = i + 2$ do

if $h_i < h_{i+1}$ then

$W'[i/2] = 0$

else

$W'[i/2] = 1$

end if

end for

if $W' == W$ then

$\nu = TRUE$

else

$\nu = FALSE$

end if

switch the position of r_i and $r_i + 1$ physically in DB

end if

end for

The number g of groups is used to make a trade-off between security and localization. On the one hand, the smaller the value of g, the larger the probability of detecting modifications in watermark detection, and the more secure the proposed scheme. On the other hand, this leads to a larger group size; thus, one can localize modifications less precisely as there are more tuples in each group.

Robust watermarking

To confuse watermark detection, various database attacks may be launched to watermarked databases. Typical database attacks include tuple/ attribute insertion/ deletion/ reorganization, value modification/ suppression (including random/ selective bit-flipping/ value-rounding), invertibility attack (attacker successfully discovers a fictitious watermark which is in fact a random occurrence from a watermarked database), additive attack (attacker embeds some additional watermarks into a watermarked database), and the brute force attack against the secret key. The brute-force attack can be thwarted by assuming that the key is long enough (e.g., 160 bits) in watermarking.

The following probabilities reflect the succeeded attacks.

- False hit: the probability of the original watermark being detected from unmarked data or a fictitious watermark being detected from watermarked data (i.e., invertibility attack).
- False miss: the probability of not detecting the original watermark from watermarked data.

The robustness of watermarking can be achieved by using majority vote in watermark detection.

Consider Bernoulli trials with probability p of success and q of failure. Let $b(k; m, p) = \binom{m}{k} p^k q^{m-k}$ be the probability that m Bernoulli trials result in k successes and $m - k$ failures, where $\binom{m}{k} = m!/k!(m-k)!, 0 \leq k \leq m$.

Let $B(k; m, p) = \sum_{i=k+1}^{m} b(i; m, p)$ be the probability of having more than k successes in m Bernoulli trials. A watermark is detected if more than τ in percentage of the embedded bits are detected correctly. If the watermark detection is applied to unmarked data (or watermarked data with a different secret key), the detection can be considered as Bernoulli trials with a probability of 1/2 that a correct value will be found in a specific bit position. Assuming ω bits are checked in watermark detection, then the false hit rate is $B(\lfloor \tau\omega \rfloor; \omega, 0.5)$. The false hit rate is extremely low if τ and ω are reasonably large. For example, the false hit rate can be as low as 10^{10} for $\tau \geq 0.6$ and $\omega \geq 1000$.

The false miss rate can be analyzed under various attack scenarios. A typical modification attack is that an attacker randomly flips every least significant bit with a probability $p < 0.5$ (if $p \geq 0.5$, one can flip every bit back before watermark detection). For the detection algorithm to fail to recover the correct watermark, at least $\omega - \lfloor \tau\omega \rfloor$ embedded bits must be toggled. Thus, the false miss rate is $B(\omega - \lfloor \tau\omega \rfloor - 1; \omega, p)$. An attacker has to flip a significant portion of tuples in order to get a high probability of success in this attack.

This scheme relies on the following assumptions to maintain its robustness. First, the watermarked relation has a primary key attribute that either does not change or else can be recovered. The rationale behind this is that a primary key attribute contains essential information and that modification or deletion of this information will substantially reduce the value of the data. With this assumption, the watermark detection is robust against tuple insertion/deletion and it is not affected by tuple reorganization. Second, the names of some, if not all, of the watermarked attributes either do not change or else can be recovered in watermark detection. Under the above two assumptions, the scheme is robust against attribute operations including insertion, deletion, and reorganization.