Information Security CS3002 (Sections BDS-7A/B) Lecture 18

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science

21 October, 2024

SQL Access Controls

- Does the user have access to the entire database or just portions of it?
- Two commands:

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

- e.g. `GRANT SELECT ON ANY TABLE TO john`

`WITH GRANT OPTION: whether grantee can grant "GRANT" option to other users`

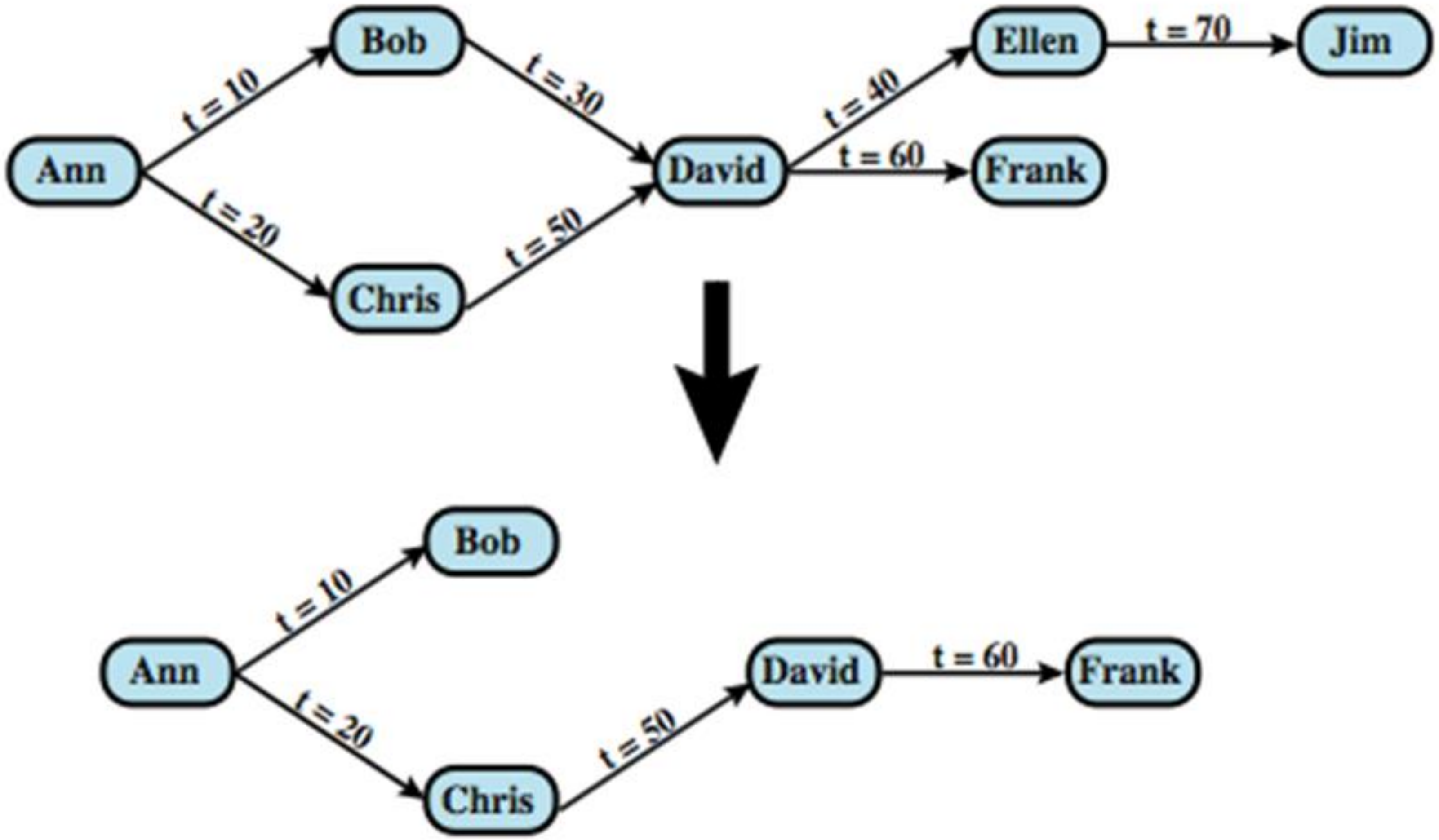
SQL Access Controls

- Does the user have access to the entire database or just portions of it?
- Two commands:

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

- `e.g. REVOKE SELECT ON ANY TABLE FROM john`
- Typical access rights are:
  - `SELECT, INSERT, UPDATE, DELETE, REFERENCES`

Cascading Authorizations



Cascading Authorizations

- The grant option enables an access right to cascade through a number of users.
- We consider a specific access right and illustrate the cascade phenomenon in Figure 5.4.
- The figure indicates that Ann grants the access right to Bob at time *t* = 10 and to Chris at time *t* = 20.
- Assume that the grant option is always used.
- Thus, Bob is able to grant the access right to David at *t* = 30.
- Chris redundantly grants the access right to David at *t* = 50.
- Meanwhile, David grants the right to Ellen, who in turn grants it to Jim; and subsequently David grants the right to Frank.

Cascading Authorizations

- Just as the granting of privileges cascades from one user to another using the grant option, the revocation of privileges also cascaded.
- Thus, if Ann revokes the access right to Bob and Chris, then the access right is also revoked to David, Ellen, Jim, and Frank.
- A complication arises when a user receives the same access right multiple times, as happens in the case of David.
- *Suppose that Bob revokes the privilege from David.*
- David still has the access right because it was granted by Chris at $t = 50$.
- However, David granted the access right to Ellen after receiving the right, with grant option, from Bob but prior to receiving it from Chris.

Cascading Authorizations

- Most implementations dictate that in this circumstance, the access right to Ellen and therefore Jim is revoked when Bob revokes the access right to David.
- This is because at *t* = 40, when David granted the access right to Ellen, David only had the grant option to do this from Bob.
- *When Bob revokes the right, this causes all subsequent cascaded grants that are traceable solely to Bob via David to be revoked.*
- Because David granted the access right to Frank after David was granted the access right with grant option from Chris, *the access right to Frank remains*. These effects are shown in the bottom of Figure 5.4.

Role-Based Access Control (RBAC)

- Role-based access control work well for DBMS
    - eases admin burden, improves security

- Categories of database users:
    - Application Owner
    - End User
    - Administrator

- DB RBAC must manage roles and their users

Database User Categories

- **Application owner**: An end user that owns database objects (tables, columns, rows) as part of an application. The application owner may assign roles to end users.
- **End user other than application owner**: An end user that operates on database objects via a particular application but does not own any of the database objects.
- **Administrator**: User who has administrative responsibility for part of all of the database.
  - Administrators are responsible for more sensitive or general roles, including those having to do with managing physical and logical database components, such as data files, users, and security mechanisms.
  - The system needs to be set up to give certain administrators certain privileges.
  - Administrators in turn can assign users to administrative-related roles.

Capabilities Needed (RBAC Facility in DB)

- Create and delete roles
- Define permissions for a role
- Assign and cancel assignment of users to roles

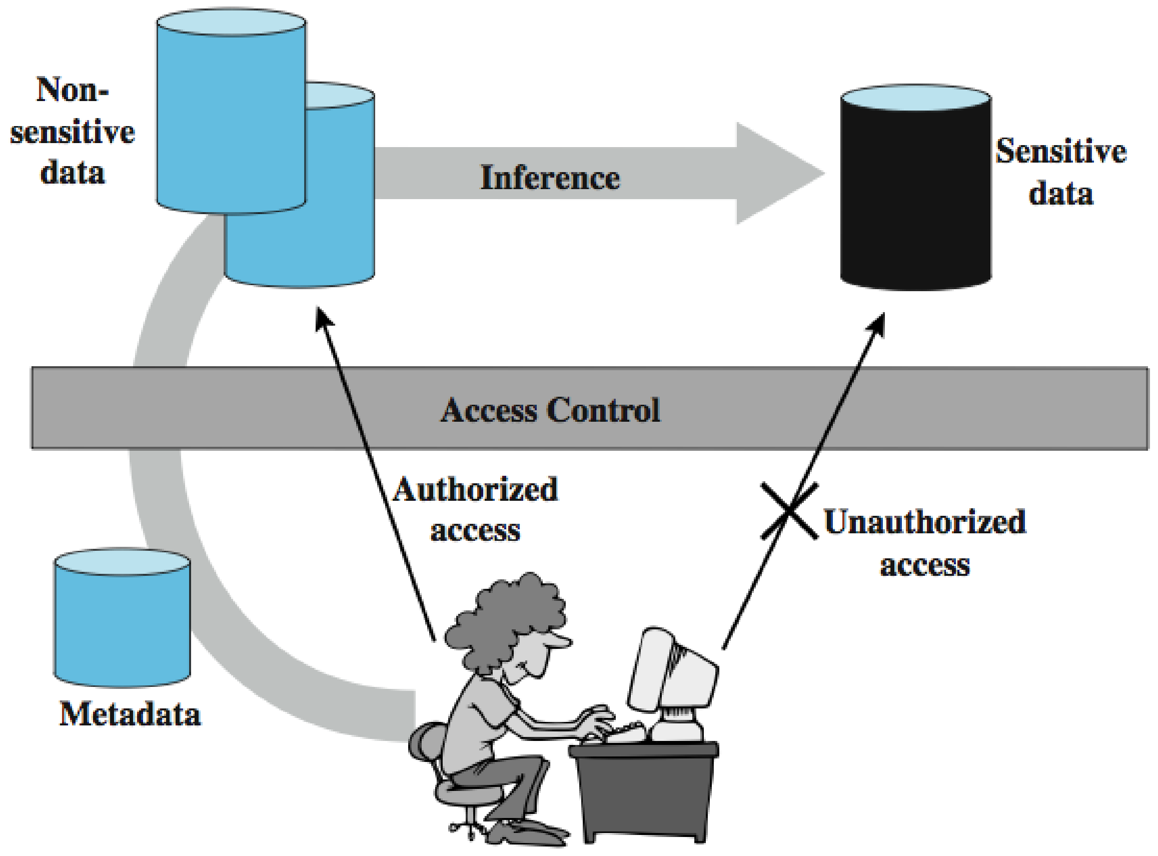
- A good example of the use of roles in database security is the RBAC facility provided by Microsoft SQL Server.
- SQL Server supports three types of roles:
  - server roles,
  - database roles, and
  - user-defined roles.
    - See text for details. (Table 5.2, Chapter 5, Stallings)

Database Security - II

- Database Inference attacks and countermeasures
- Database encryption methods

Inference

- The process of *performing authorized queries and deducing unauthorized information from the legitimate responses received*
- A combination of data items can be used to *infer data of a higher sensitivity*



- 2 inference techniques can be used to derive additional information:
  - analyzing *functional dependencies between attributes* within a table or across tables
  - *merging views* with the same constraints

Inference Example

Name	Position	Salary (S)	Department	Location
Andrew	Programmer Analyst	$80,000	Software	Jackson, MS
Robert	Quality Control	$65,000	Software	Memphis, TN
Sheela	Software Developer	$90,000	Software	Atlanta, GA
Victor	Systems Engineer	$75,000	Systems	San Jose, CA
Mary	Systems Administrator	$95,000	Systems	Seattle, WA
Ryan	Network Engineer	$87,000	Systems	Boston, MA

View V1

Name	Position
Andrew	Programmer Analyst
Robert	Quality Control
Sheela	Software Developer

View V2

Salary (S)	Department	Location
$80,000	Software	Jackson, MS
$65,000	Software	Memphis, TN
$90,000	Software	Atlanta, GA

```
CREATE VIEW V1 AS
SELECT Name, Position
FROM Employee
WHERE Department = “Software”
```

```
CREATE VIEW V2 AS
SELECT Salary, Location
FROM Employee
WHERE Department = “Software”
```

Inference Example

- There is no *functional relationship between Name and Salary* such that knowing Name and perhaps other information is sufficient to deduce Salary.
- However, suppose the *two views are created with the access constraint that Name and Salary cannot be access together*.
- A user who knows the structure of the Employee table and who knows that the view tables *maintain the same row order* as the Employee table is then able to merge the two views to construct the table shown in the previous slide.
- This *violates the access control policy that the relationship of attributes Name and Salary must not be disclosed*.

Another Inference Example (p. 190, Stallings)

Database containing personnel information, including names, addresses, and salaries of employees.

Individually, the name, address, and salary information is available to a subordinate role, such as *Clerk*, but the association of names and salaries is restricted to a superior role, such as Administrator.

Possible Solution: *construct three tables*, which include the following information:

```
Employees (Emp#, Name, Address)
Salaries (S#, Salary)
Emp-Salary (Emp#, S#)
```

In this case, each employee is assigned a unique employee number (Emp#) and a unique salary number (S#).
The Employees table and the Salaries table are accessible to the Clerk role, but the Emp-Salary table is only available to the Administrator role.
In this structure, the sensitive relationship between employees and salaries is protected from users assigned the Clerk role.

Another Inference Example

- Suppose we want to add a new attribute, employee start date, which is not sensitive. This could be added to the Salaries table as follows:

```
Employees (Emp#, Name, Address)
Salaries (S#, Salary, Start-Date)
Emp-Salary (Emp#, S#)
```

- However, an employee’s start date is an easily observable/discoverable attribute.
- Thus, a user in the Clerk role should be able to infer (or partially infer) the employee’s name. This would compromise the relationship between employee and salary.
- Possible Solution → remove the inference channel by moving the start-date column to the Employees table

```
Employees (Emp#, Name, Address , Start-Date)
Salaries (S#, Salary)
Emp-Salary (Emp#, S#)
```

Inference Countermeasures

- Inference detection at ***database design***
  - alter database structure or access controls
  - Splitting a table into multiple tables
  - More fine grain access control roles
- Inference detection at ***query time***
  - by monitoring and altering or rejecting queries

Statistical Databases

- Provides data of a statistical nature
    - e.g. counts, averages
- Two types:
    - *pure statistical database*
        - Only stores statistical data (like census database)
    - *ordinary database with statistical access*
        - Contains individual entries
        - some users have normal access, others statistical
- *Access control objective to allow statistical use without revealing individual entries*
- Security problem is one of inference

SDB: Characteristic Formula

- Statistics are derived from a database by means of a logical Boolean formula (referred as ***Characteristic formula***) over the values of attributes.
- A Characteristic formula uses the operators OR, AND, and NOT (+, *, ~), written here in the increasing order of priority.
  - E.g., `(Gender = ‘Male’) * ( (Major = ‘CS’) + (Major = ‘EE’))` specifies all male students majoring in either CS or EE
  - For numerical attributes, relational operators may be used, e.g., `(GPA > 3.7)`
- For simplicity, we may omit the attribute names if they are clear from context. E.g., `‘Male’ * (‘CS’ + ‘EE’)`

SDB Example: DB with Statistical Access with 13 Students

Name	Gender	Major	Class	SAT	GPA
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	IT	1979	580	2.8
Iles	Male	CS	1981	600	3.2
Jones	Female	BIO	1979	750	3.8
Kline	Female	IT	1981	500	2.5
Lane	Male	EE	1978	600	3.0
Moore	Male	CS	1979	650	3.5

Attribute A_j	**Possible values**	**$\|A_j\|$**
Gender	Male, Female	2
Major	BIO, CS, IT, EE, ..	30
Class	1978, 1979, 1980, 1981	4
SAT	310, 320, 330, …, 790, 800	50
GPA	0.0, 0.1, 0.2, …., 3.9, 4.0	41

SDB: Characteristic Formula

- The query set of characteristic formula C, denoted as X(C), is the set of records matching that characteristic
- For example, for `C = ‘Male’ * ‘EE’, X(C) = 3`, matching the records of Cook, Frank and Lane (all ‘Male’ and ‘EE’ majors).
- A statistical query is a query that produces a value calculated over a query set. For example:
  - `count (‘Female’ * ‘CS’) = 2`
  - `sum (‘Female’ * ‘CS’, ‘SAT’) = 1400`
- ***Inference***: Assume a questioner knows that Baker is a female EE student; but, he does not know that she is the only one. The following sequence of two queries will reveal Baker’s GPA.
  - `count (‘EE’ * ‘Female’) = 1`
  - `sum (‘EE’ * ‘Female’, ‘GPA’) = 2.5`

Solution for SDB Inference:

Query Restriction
Perturbation

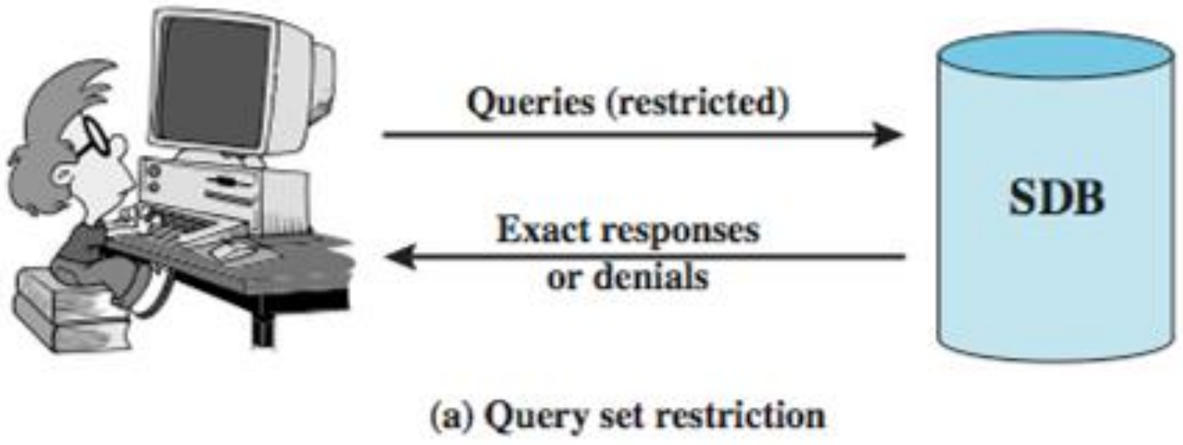
Example

- Salary range for system analyst with BS (50K, 60K)
- Salary range for system analyst with MS (65K, 70K)

- What if the change in payroll is 70K or 140K?

Protecting Against Inference

Query restriction:



- Rejects a query that can lead to a compromise.
- The answers provided are accurate.
- The simplest form of query restriction is 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 <= |X(C)| where k is a fixed integer greater than 1*.
- In practice, queries of the form q(All) are allowed, enabling users to easily access statistics calculated on the entire database.

Query Restriction

- Why is there a need for the Upper Bound?
- The upper bound is needed to avoid someone from inferring using a query of type q(All) that would return statistics based on the entire database.
  - *Instead of directly querying using C to collect statistics on an individual record, one could collect statistics by computing:*

```
q(All) – q(~C)
```

Tracker: Example

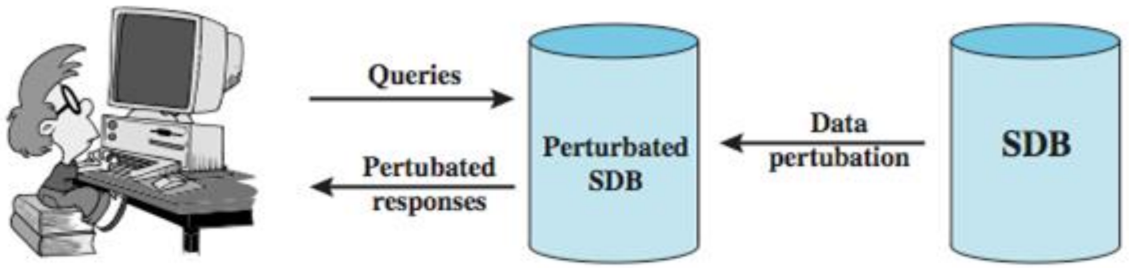
- Consider the table below. Suppose that we want to know whether *Evans scored 600 or above in SAT.*
- Assume that we know that Evans is a Bio major of class 1979.
- If we directly launch the query based on the formula →

  `C = Male * Bio * 1979`

  on the database table, the count (C) = 1 and the results of the query will not be returned.
- Suppose, we break `C = C1 * C2`; where
  - `C1 = Male`
  - `C2 = Bio * 1979`

Perturbation

- Add noise to statistics generated from data
  - will result in differences in statistics

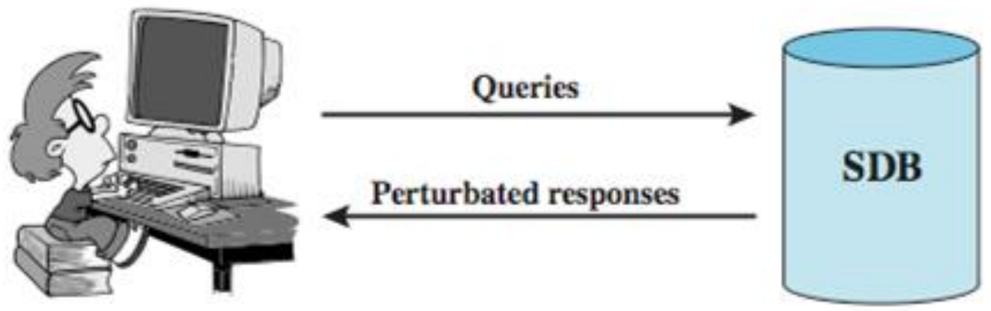


(b) Data perturbation

- Data perturbation techniques
  - ***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.

Perturbation

- Output perturbation techniques
  - ***Random-sample query***
    - Calculate the statistics on a properly selected sampled query subset
  - ***Statistic adjustment***
    - Adjusting the answer up or down by a given amount in some systematic fashion



(c) Output perturbation

  - Must minimize loss of accuracy in results

Database Encryption

- Databases typically a *valuable info resource*
  - protected by multiple layers of security: firewalls, authentication, O/S access control systems, DB access control systems, and database encryption

- Can encrypt
  - *entire database* - very inflexible and inefficient
  - *individual fields* - simple but inflexible
  - *records (rows)* or *columns (attributes)* - best
    - also need attribute indexes to help data retrieval

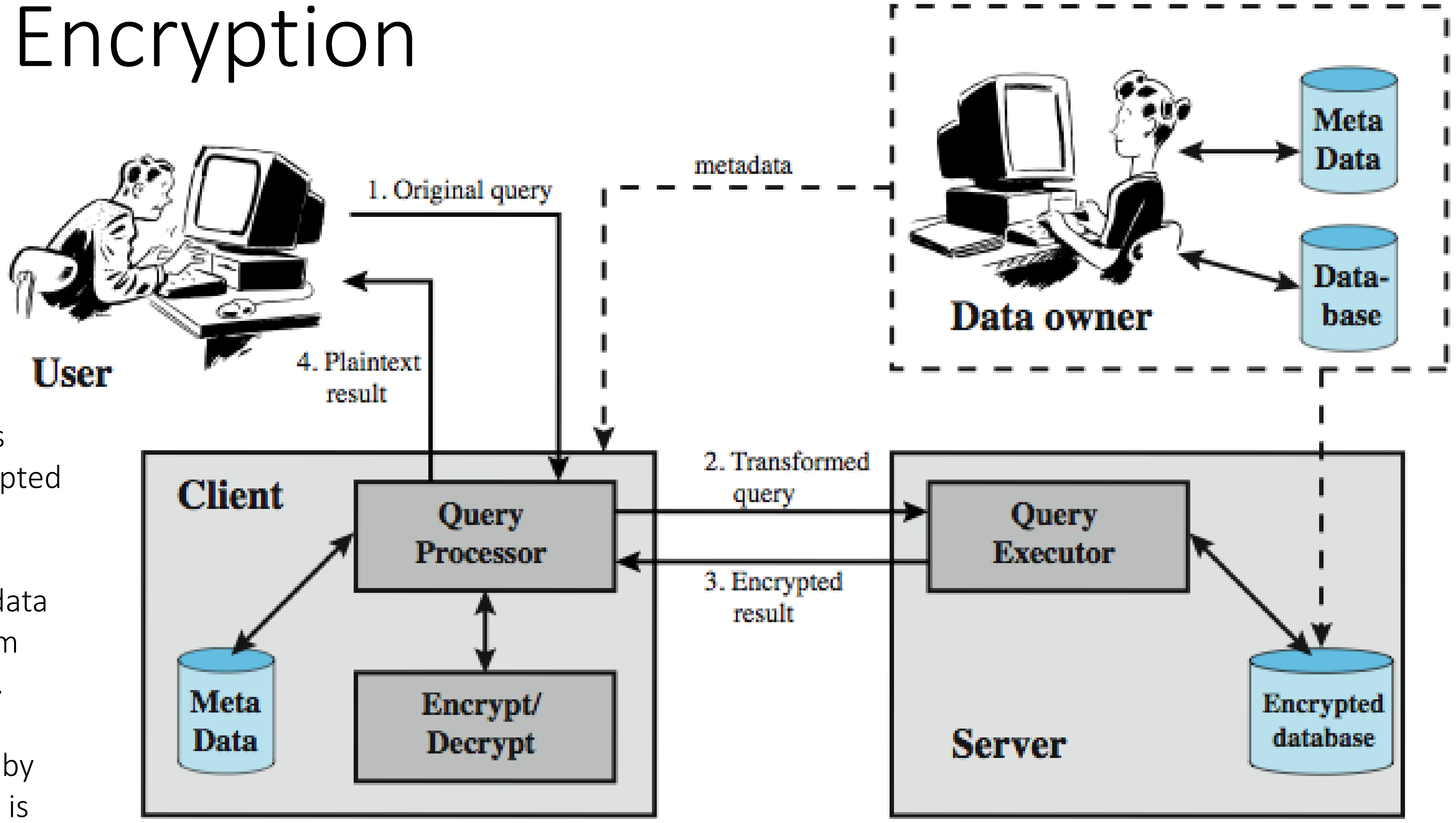
- Varying trade-offs

Database Encryption

Data owner: organization that produces the sensitive data

- **User:** that presents requests (queries) to the system.
- **Client:** Front-end that transforms user queries into queries on encrypted data
- **Server:** that receives encrypted data from a data owner and makes them available for distribution to clients.

The server could in fact be owned by the data owner but more typically, is owned and maintained by an external provider.



Secured operations

- Suppose that each individual item in the database is encrypted separately.
- The encrypted database is stored at the server, but the server does not have the key, so that the data is secure at the server. The client system does have a copy of the encryption key.
- A user at the client can retrieve a record from the database with the following sequence:
    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 record or records*.
    4. The query processor *decrypts the data and returns the results*.

Example

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

Assume that the encryption key *k* is used and that the encrypted value of the department id 15 is

```
 E(k, 15) = 1000110111001110.
```

Then the query processor at the client could transform the preceding query into

```
SELECT Ename, Eid, Ephone
FROM Employee
WHERE Did = 1000110111001110
```

Issues

- This method is certainly straightforward but *lacks flexibility*.

  For example, suppose the Employee table contains a salary attribute and the user wishes to *retrieve all records for salaries less than 70K*.

  There is no obvious way to do this, because the *attribute value for salary in each record is encrypted*.

  The set of encrypted values do not preserve the ordering of values in the original attribute.

Record level encryption and indexing

- Each record (row) of a table in the database is *encrypted as a block which is treated as a contiguous block*.

- To assist in data retrieval, *attribute indexes are associated with each table*. For some or all of the attributes an index value is created.

- For each row in the original database, *there is one row in the encrypted database*.

- For any attribute, the *range of attribute values is divided into a set of non-overlapping partitions that encompass all possible values*, and an index value is assigned to each partition.

Example

- Suppose that employee ID (*eid*) values lie in the range [1, 1000]. We can 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 a text field, we can derive an index from the first letter of the attribute value. For the attribute ***ename***, let us assign index **1** to values starting with **A or B**, index **2** to values starting with **C or D**, and so on.

- Similar partitioning schemes can be used for each of the attributes.

Example: Employee Table (with encryption)

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

Employee Table

E(k, B)	I(eid)	I(ename)	I(salary)	I(addr)	I(did)
1100110011001011…	1	10	3	7	4
0111000111001010…	5	7	2	7	8
1100010010001101…	2	5	1	9	5
0011010011111101…	5	5	2	4	9

Encrypted Employee Table with Indexes