Information Security

CS3002

Lecture 15

9th October 2024

Dr. Rana Asif Rehman

Email: r.asif@lhr.nu.edu.pk

Database Security

Lecture Outline

- Database Access Control
- Inference
- Database Encryption

1. Database Access Control

Database Access Control

- DBMS provide access control for database
- It operates on assumption that user is already authenticated

- DBMS provides specific access rights to portions of the database
  - e.g. create, insert, delete, update, read, write
  - to entire database, tables, selected rows or columns
  - possibly dependent on contents of a table entry (i.e. salary info.)

- Can support a range of policies:
  - Centralized administration (small number of privileged users grant access)
  - Ownership-based administration (the owner i.e. creator of a table may grant)
  - Decentralized administration (in addition to ownership based rights, the owner may grant or revoke the authorization rights to other users)

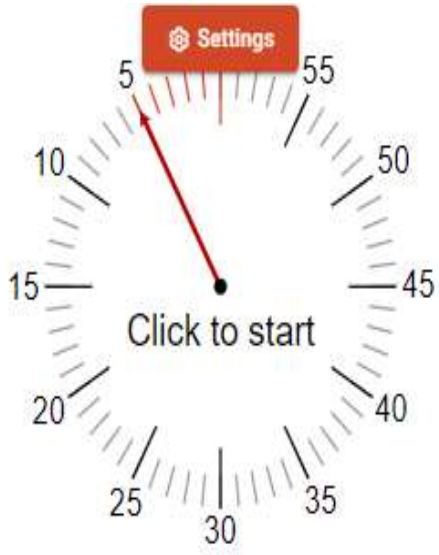
Database Access Control

- SQL Access Controls
- Cascading Authorizations
- Role-based Access Control (RBAC)

1.1 SQL Access Controls

- If the user has access to the entire database or just portions of it
- Two commands in SQL for managing access rights/roles
  - `GRANT {privileges | role} [ON table] TO {user | role | PUBLIC} [IDENTIFIED BY password] [WITH GRANT OPTION]`
    - e.g. GRANT SELECT ON ANY TABLE TO john
  - `REVOKE {privileges | role} [ON table] FROM {user | role | PUBLIC}`
    - e.g. REVOKE SELECT ON ANY TABLE FROM john
  - `WITH GRANT OPTION: whether grantee can grant "GRANT" option to other users`
  - `IDENTIFIED BY: specifies a password that must be used to revoke the access rights of this grant`
- Typical access rights are:
  - `SELECT, INSERT, UPDATE, DELETE, REFERENCES (i.e. define foreign keys in another table)`

Activity Time

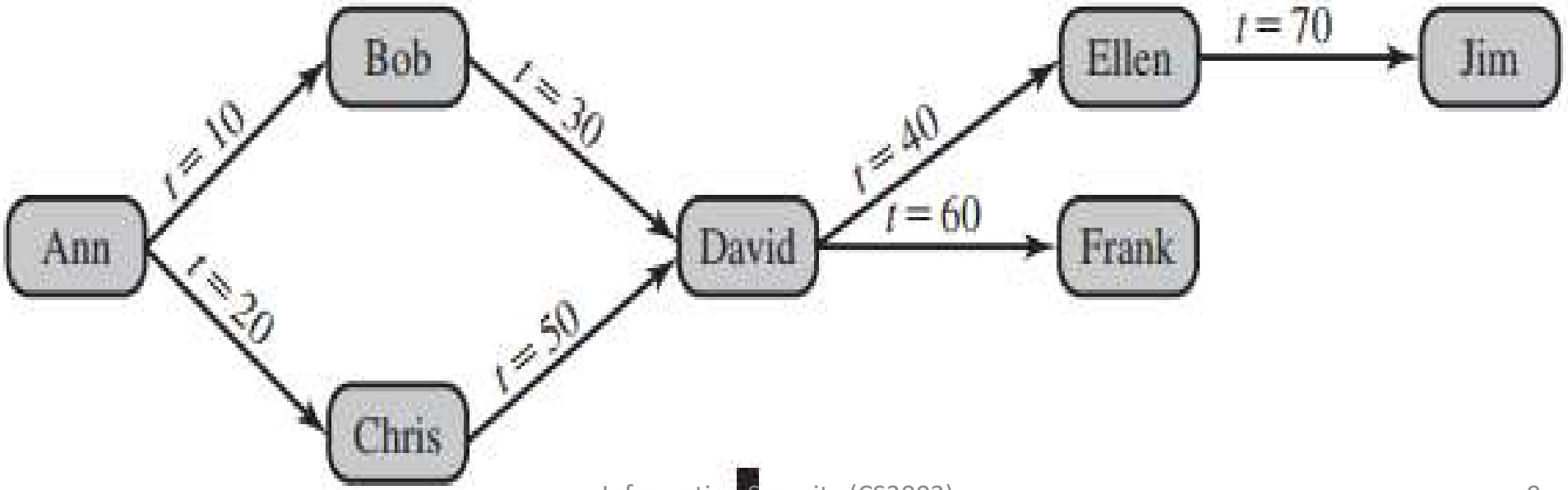


Allow a user 'peter' to create new rows or delete some rows in 'Repository' table. He should not be able to see table data or modify anything within the table rows.

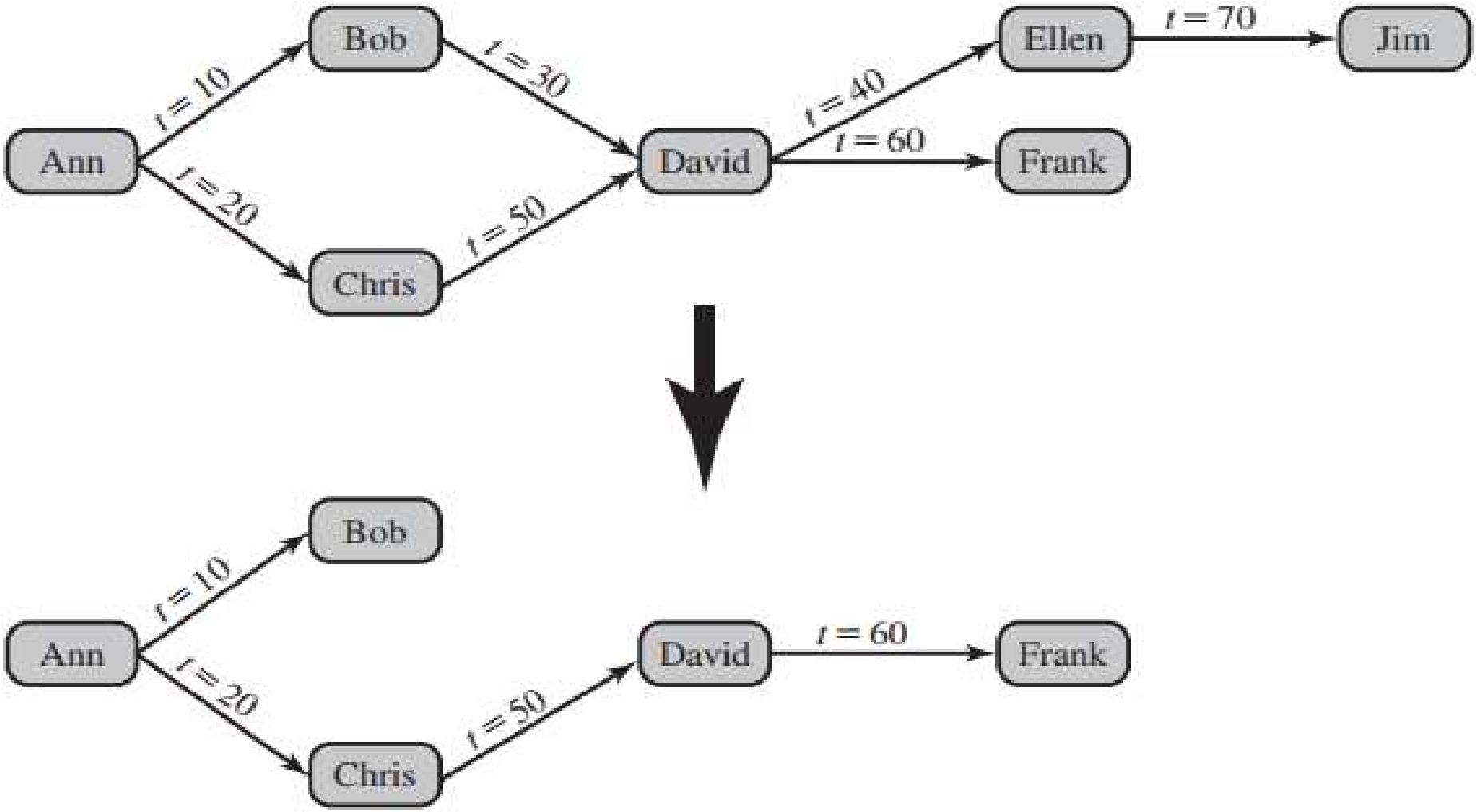
Answer: GRANT INSERT, DELETE ON Repository TO peter;

1.2 Cascading Authorizations

- The grant option enables an access right to cascade through a number of users
- The revocation of privileges should also be cascaded i.e. if Ann revokes the access rights
- Complication arises when a user receives the same access right multiple times, as happens in the case of David if Bob revoke the right



Cascading Authorizations



1.3 Role-Based Access Control (RBAC)

- Role-based access control work well (natural fit) for DBMS
  - In DB system, an individual user may use a variety of applications to perform a variety of tasks, which requires its on set of privileges
  - It would be poor administrative practice to simply grant users all of the access rights they require for all the tasks they perform
  - eases admin burden, improves security

- Categories of Database Users:
  - Application Owner (end user who owns database objects i.e. tables, column, rows as a part of an application)
  - End User (who operates on database objects, does not own)
  - Administrator (who has administrative responsibility for part of or all of the DB)

- DB RBAC must manage roles and their users

Role-Based Access Control (RBAC)

- An application has associated with it a number of tasks, with each task requiring specific access rights to portions of the database
- For each task, one or more roles can be defined that specify the needed access rights
- The application owner may assign roles to end users

- A database RBAC facility needs to provide the following capabilities:
  - 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 (fixed, preconfigured with specific access rights)
  - Database roles (fixed, preconfigured with specific access rights)
  - User-defined roles (standard role, application role)

RBAC: Microsoft SQL Server

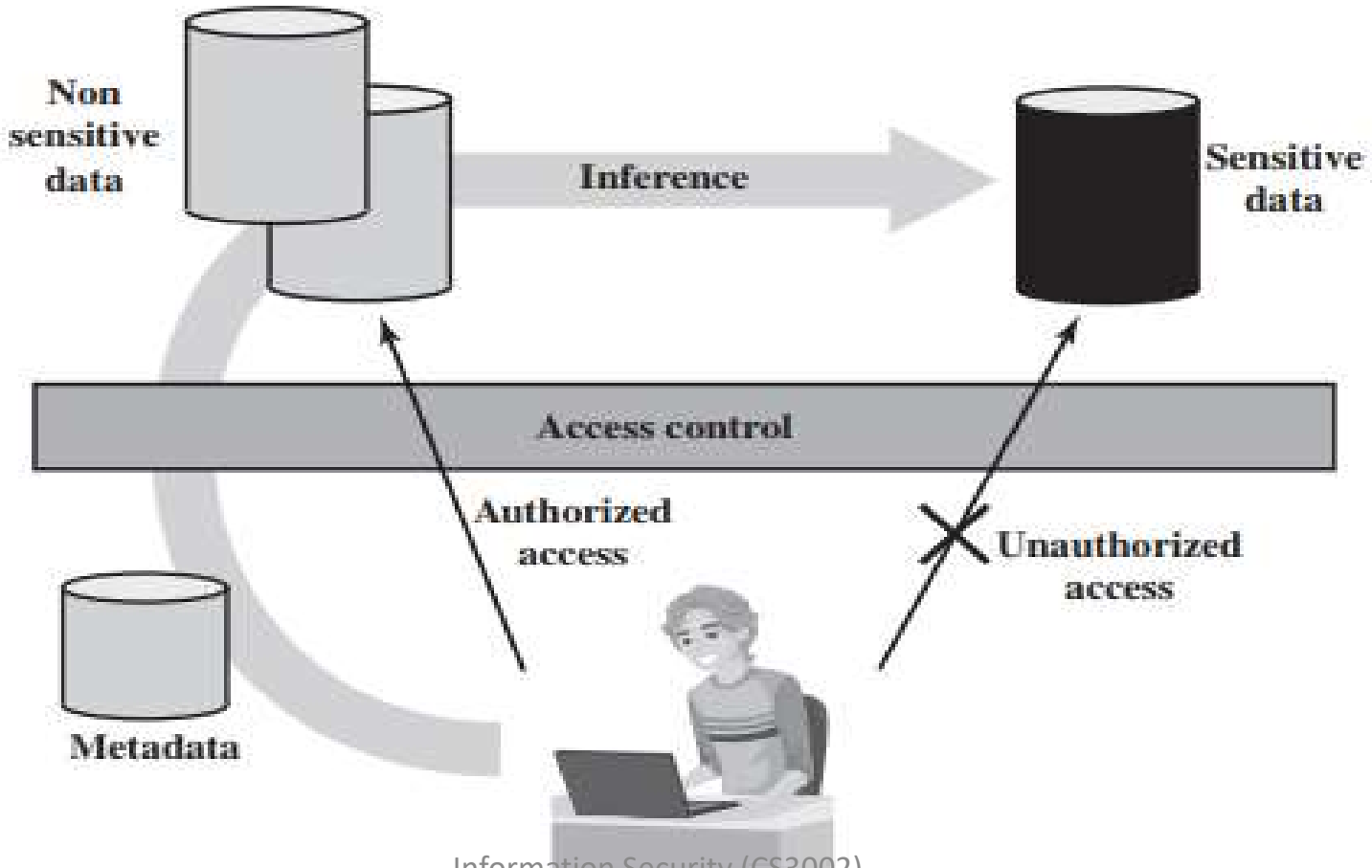
Table 5.2 **Fixed Roles in Microsoft SQL Server**

Role	Permissions
Fixed Server Roles	
sysadmin	Can perform any activity in SQL Server and have complete control over all database functions
serveradmin	Can set server-wide configuration options and shut down the server
setupadmin	Can manage linked servers and startup procedures
securityadmin	Can manage logins and CREATE DATABASE permissions, also read error logs and change passwords
processadmin	Can manage processes running in SQL Server
Dbcreator	Can create, alter, and drop databases
diskadmin	Can manage disk files
bulkadmin	Can execute BULK INSERT statements
Fixed Database Roles	
db_owner	Has all permissions in the database
db_accessadmin	Can add or remove user IDs
db_datareader	Can select all data from any user table in the database
db_datawriter	Can modify any data in any user table in the database
db_ddladmin	Can issue all data definition language statements
db_securityadmin	Can manage all permissions, object ownerships, roles and role memberships
db_backupoperator	Can issue DBCC, CHECKPOINT, and BACKUP statements
db_denydatareader	Can deny permission to select data in the database
db_denydatawriter	Can deny permission to change data in the database

2. Inference

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



Inference Example

```
CREATE view V1 AS
SELECT Availability, Cost
FROM Inventory
WHERE Department = "hardware"
```

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware
Cake pan	online only	12.99	housewares
Shower/tub cleaner	in-store/online	11.99	housewares
Rolling pin	in-store/online	10.99	housewares

(a) Inventory table

```
CREATE view V2 AS
SELECT Item, Department
FROM Inventory
WHERE Department = "hardware"
```

Availability	Cost ($)
in-store/online	7.99
online only	5.49
in-store/online	104.99

Item	Department
Shelf support	hardware
Lid support	hardware
Decorative chain	hardware

(b) Two views

Item	Availability	Cost ($)	Department
Shelf support	in-store/online	7.99	hardware
Lid support	online only	5.49	hardware
Decorative chain	in-store/online	104.99	hardware

(c) Table derived from combining query answers

Inference example

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

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

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

Inference Countermeasures

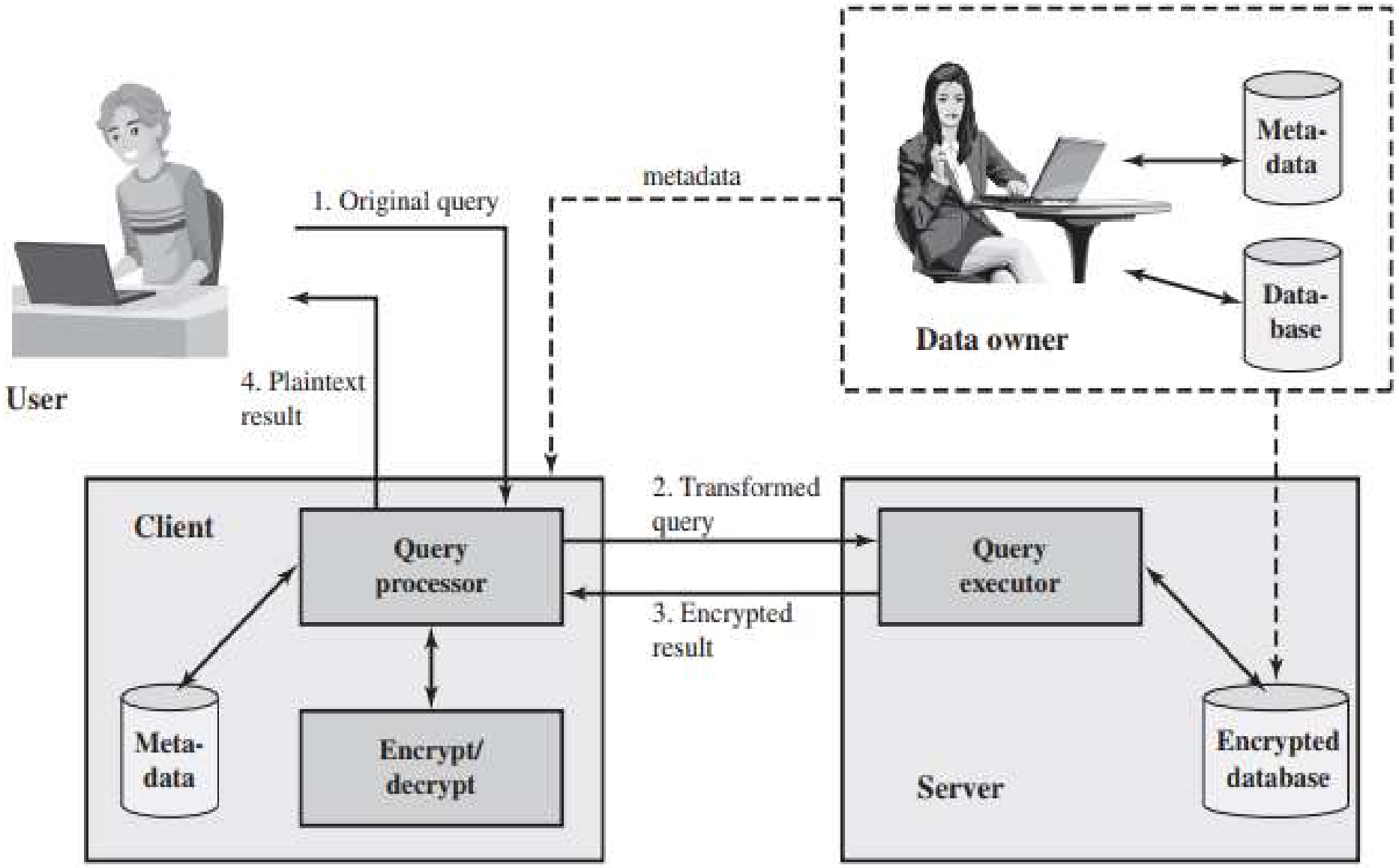
- Inference Detection at Database Design
  - Alter database structure or access controls regime to prevent Inference
  - Remove data dependencies
    - Splitting a table into multiple tables
    - More fine-grained access control roles in RBAC scheme
- Inference Detection at Query Time
  - This approach 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.

3. Database Encryption

Database Encryption

- Databases typically a valuable information resource
  - protected by multiple layers of security: firewalls, authentication, O/S access control systems, DB access control systems
  - In addition, for particularly sensitive data, database encryption is warranted and often implemented
- Can encrypt
  - entire database - very inflexible (record searching) and inefficient (key management)
  - individual fields - simple but inflexible
  - records (rows) or columns (attributes) - best
    - also need attribute indexes to help data retrieval
- Two approaches for DB security
  - Outsource DBMS to service provider (concern regarding confidentiality of data)
  - Encrypt the database (inflexible regarding searching and decrypting)
- Flexible, if possible to work with database in its encrypted form

Database Encryption



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

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.

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

(a) Employee 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

(b) Encrypted Employee Table with Indexes

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