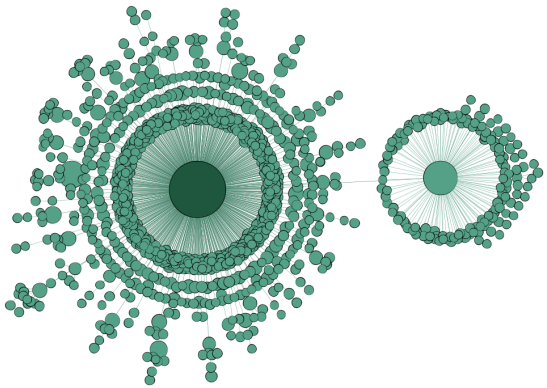
Week 9 Workshop - Database Security

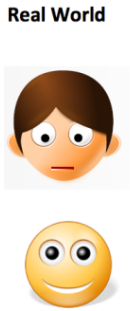
Qing Wang

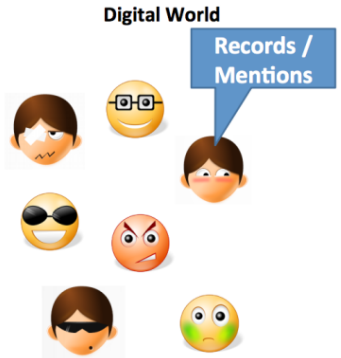
Website: *http://users.cecs.anu.edu.au/~u5170295/*
https://graphlabanu.github.io/website/

Zoom drop-in session: Tuesday 2pm-3pm (Week 10 to Week 12)

Research areas: Data management and analytics
Data mining
Deep learning on graphs
Graph algorithms.







House Keeping

- Lab 8 (Database programming) in Week 10 is optional - three options. A sign-up page is available on Wattle.
- Assignment 2 (Database Theory) is due at 23:59, Oct 12.

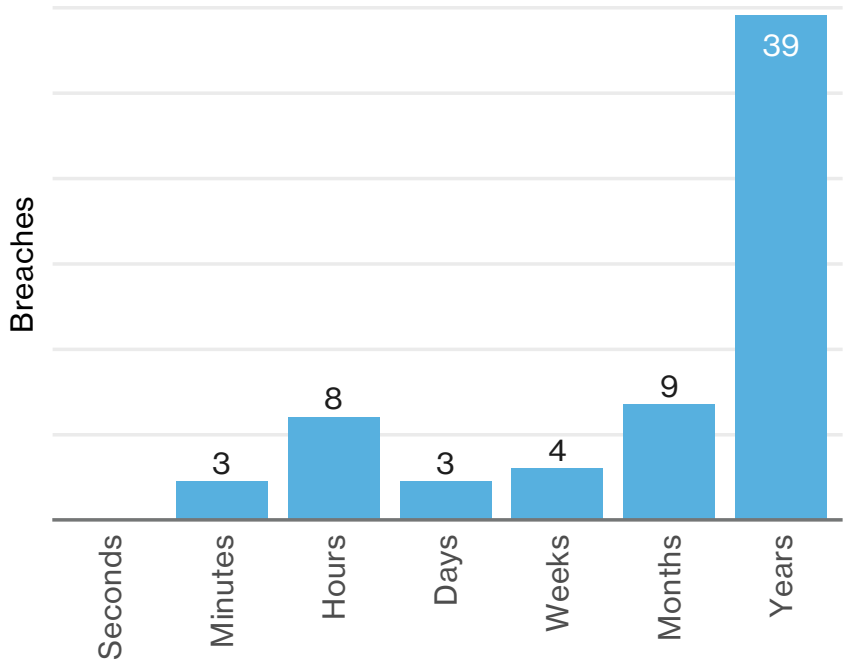
Week 9 Workshop - Database Security

"Hardware is easy to protect: lock it in a room, chain it to a desk, or buy a spare. Information poses more of a problem. It can exist in more than one place; be transported halfway across the planet in seconds; and be stolen without your knowledge."

– Bruce Schneier

Data Breaches

- In 80% of cases, attackers are able to compromise an organization within minutes. However, in almost 60% of cases, it takes years to learn that they have been breached.[1]



Time-to-discovery within Public breaches (n=66)

[1] Verizon 2016&2017 Data Breach Investigation Reports

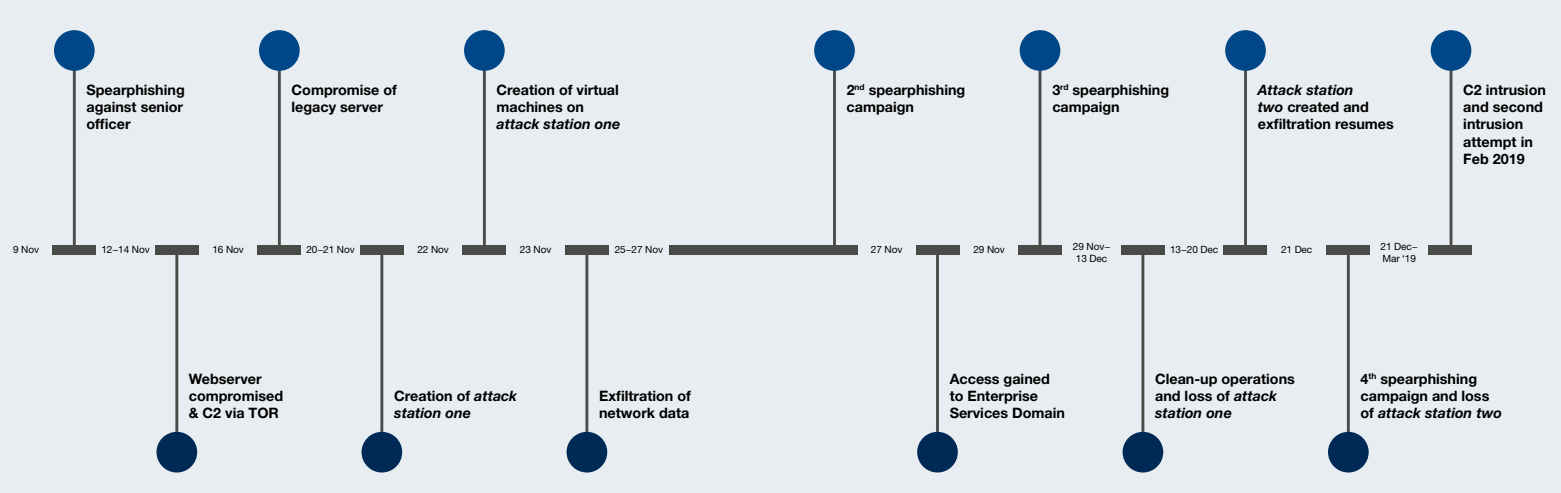
Data Breaches

- June 2019, ANU community was notified of a data breach.

Data Breaches

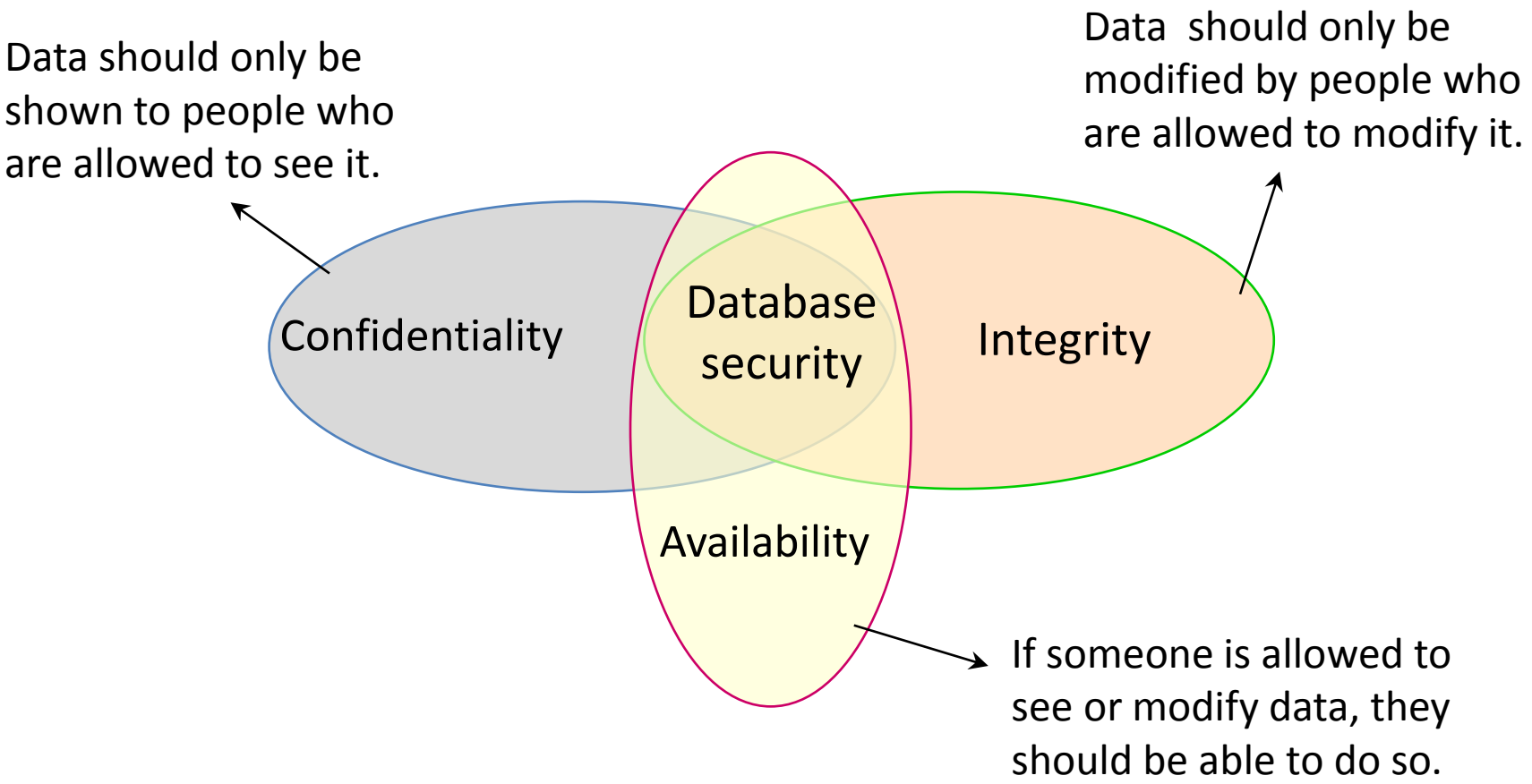
- June 2019, ANU community was notified of a data breach.

"It's shocking in its sophistication"



"While we cannot confirm exactly what data was taken, we know it was much less than the 19 years' worth we originally reported"

Objectives of Database Security



Database Security - Examples

1. *A health-care information system*

Database Security - Examples

1. *A health-care information system*
   - A patient's medical information should not be improperly disclosed.

Database Security - Examples

1. *A health-care information system*
   - A patient's medical information should not be improperly disclosed.
   - A patient's medical information should be correct.

Database Security - Examples

1. *A health-care information system*
   - A patient's medical information should not be improperly disclosed.
   - A patient's medical information should be correct.
   - A patient's medical information can be accessed when needed for treatment.

Database Security - Examples

1. *A health-care information system*
   - A patient's medical information should not be improperly disclosed.
   - A patient's medical information should be correct.
   - A patient's medical information can be accessed when needed for treatment.
2. *A military system*

Database Security - Examples

1. *A health-care information system*
   - A patient's medical information should not be improperly disclosed.
   - A patient's medical information should be correct.
   - A patient's medical information can be accessed when needed for treatment.

2. *A military system*
   - The target of a missile cannot be given to an unauthorized user.

Database Security - Examples

1. *A health-care information system*
   - A patient's medical information should not be improperly disclosed.
   - A patient's medical information should be correct.
   - A patient's medical information can be accessed when needed for treatment.

2. *A military system*
   - The target of a missile cannot be given to an unauthorized user.
   - The target of a missile cannot be arbitrarily modified.

Database Security - Examples

1. *A health-care information system*
   - A patient's medical information should not be improperly disclosed.
   - A patient's medical information should be correct.
   - A patient's medical information can be accessed when needed for treatment.

2. *A military system*
   - The target of a missile cannot be given to an unauthorized user.
   - The target of a missile cannot be arbitrarily modified.
   - The target of a missile can be accessed when needed.

Database Security - Core Services

- **Confidentiality**
- **Integrity**
- **Availability**

Database Security - Core Services

- **Confidentiality**
  - E.g. enforced by access control mechanisms
- **Integrity**
- **Availability**

Database Security - Core Services

- **Confidentiality**
  - E.g. enforced by access control mechanisms
- **Integrity**
  - E.g. enforced by access control mechanisms and integrity constraints specified on schemas
- **Availability**

Database Security - Core Services

- **Confidentiality**
  - E.g. enforced by access control mechanisms
- **Integrity**
  - E.g. enforced by access control mechanisms and integrity constraints specified on schemas
- **Availability**
  - E.g. enforced by recovery and concurrency control mechanisms

Database Security - Core Services

- **Confidentiality**
  - E.g. enforced by access control mechanisms
- **Integrity**
  - E.g. enforced by access control mechanisms and integrity constraints specified on schemas
- **Availability**
  - E.g. enforced by recovery and concurrency control mechanisms

Some further services

- **Encryption**: to protect data when being transmitted across systems and when being stored on secondary storage
- **Query authentication**: to ensure a query result is correct by using signature mechanisms and data structures
- ...

Access Control Mechanisms

Access Control Mechanisms

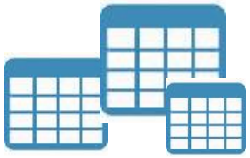
- Three types:
  1. Discretionary access control (DAC)
  2. Mandatory access control (MAC)
  3. Role-based access control (RBAC)

Database Security - DAC

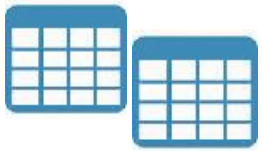
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



Granting/Revoking/Delegating Privileges

```
GRANT privileges ON object TO users [WITH GRANT OPTION]
```

Granting/Revoking/Delegating Privileges

```
GRANT privileges ON object TO users [WITH GRANT OPTION]
```

```
REVOKE [GRANT OPTION FOR] privileges ON object FROM users
       [RESTRICT|CASCADE]
```

Granting/Revoking/Delegating Privileges

```
GRANT privileges ON object TO users [WITH GRANT OPTION]
```

```
REVOKE [GRANT OPTION FOR] privileges ON object FROM users
       [RESTRICT|CASCADE]
```

- Possible privileges:
  - `SELECT`
  - `INSERT` and `INSERT`(column)
  - `UPDATE` and `UPDATE`(column)
  - `DELETE`
  - `REFERENCES`(column)
  - ...

Granting/Revoking/Delegating Privileges

- The privileges of an object can be given to a user with or without the `GRANT OPTION`

```
GRANT SELECT ON Supplier TO Bob;
GRANT SELECT ON Supplier TO Bob WITH GRANT OPTION;
```

Granting/Revoking/Delegating Privileges

- The privileges of an object can be given to a user with or without the GRANT OPTION

```
GRANT SELECT ON Supplier TO Bob;
GRANT SELECT ON Supplier TO Bob WITH GRANT OPTION;
```

- The privileges of an object can be taken away from a user. It is also possible to only revoke the GRANT OPTION on a privilege.

```
REVOKE SELECT ON Supplier FROM Bob;
REVOKE GRANT OPTION FOR SELECT ON Supplier FROM Bob;
```

Question

- In which situations a user can grant a privilege on an object to other users?
  - (1) The user is the owner of the object.
  - (2) The user has the privilege on the object.
  - (3) The user is a superuser of the database.
  - (4) The user has received the privilege with the `GRANT OPTION` from the owner of the object.

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

- Questions:
  1. What privilege(s) does Jane receive?

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

- Questions:
  1. What privilege(s) does Jane receive?
  2. What privilege(s) does Tom receive?

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

- Can these commands be executed?

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

- Can these commands be executed?

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

- Can these commands be executed?
  - The first three are fully executed.

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

- Can these commands be executed?
  - The first three are fully executed.
  - The fourth one is not executed, because Bob does not have the `UPDATE` privilege on the table.

Example - Granting Privileges

- Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT, INSERT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
(Alice):  GRANT INSERT ON Employee TO Jane;
  (Bob):  GRANT UPDATE ON Employee TO Tom WITH GRANT OPTION;
 (Jane):  GRANT SELECT, INSERT ON Employee TO Tom;
```

- Can these commands be executed?
  - The first three are fully executed.
  - The fourth one is not executed, because Bob does not have the `UPDATE` privilege on the table.
  - The fifth one is partially executed because Jane has the `SELECT` and `INSERT` privileges but no `GRANT OPTION` for `INSERT`. Therefore, Tom only receives the `SELECT` privilege.

Granting/Revoking/Delegating Privileges

- A user can only revoke privileges that the user has granted earlier, with two optional keywords in the `REVOKE` command:
  - `CASCADE`: revoking the privilege from a specified user also revokes the privileges from all users who received the privilege from that user.
  - `RESTRICT`: revoking the privilege only from a specified user.

    Possible implementations:

    (1) Causing an error message in some DBMS if the revoked privilege is still delegated;

    (2) Revoking the privilege from the specified user in any case.
- If a user receives a certain privilege from multiple sources, and the user would lose the privilege only after all sources revoke this privilege.

Example - Revoking Privileges

- Again, Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
  (Bob):  GRANT SELECT ON Employee TO Tom;
 (Jane):  GRANT SELECT ON Employee TO Tom;
  (Bob):  REVOKE SELECT ON Employee FROM Tom;
```

- Will Tom lose the SELECT privilege on EMPLOYEE?

Example - Revoking Privileges

- Again, Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
  (Bob):  GRANT SELECT ON Employee TO Tom;
 (Jane):  GRANT SELECT ON Employee TO Tom;
  (Bob):  REVOKE SELECT ON Employee FROM Tom;
```

- Will Tom lose the SELECT privilege on EMPLOYEE?
  - Tom will still hold the SELECT privilege on EMPLOYEE, since he has independently obtained such privilege from Jane.

Example - Revoking Privileges

- Again, Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
  (Bob):  GRANT SELECT ON Employee TO Tom;
 (Jane):  GRANT SELECT ON Employee TO Tom;
(Alice):  REVOKE SELECT ON Employee FROM Bob CASCADE;
```

- Will Tom lose the SELECT privilege on EMPLOYEE?

Example - Revoking Privileges

- Again, Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
  (Bob):  GRANT SELECT ON Employee TO Tom;
 (Jane): GRANT SELECT ON Employee TO Tom;
(Alice):  REVOKE SELECT ON Employee FROM Bob CASCADE;
```

- Will Tom lose the SELECT privilege on EMPLOYEE?

Example - Revoking Privileges

- Again, Alice owns table EMPLOYEE:

```
(Alice):  GRANT SELECT ON Employee TO Bob WITH GRANT OPTION;
(Alice):  GRANT SELECT ON Employee TO Jane WITH GRANT OPTION;
  (Bob):  GRANT SELECT ON Employee TO Tom;
 (Jane): GRANT SELECT ON Employee TO Tom;
(Alice):  REVOKE SELECT ON Employee FROM Bob CASCADE;
```

- Will Tom lose the SELECT privilege on EMPLOYEE?
  - Tom will lose the SELECT privilege on EMPLOYEE.

Delegating Privileges - Propagation

- There are techniques to limit the propagation of privileges.
  - Limiting **horizontal propagation**: limits that an account given the `GRANT OPTION` can grant the privilege to at most n other accounts;
  - Limiting **vertical propagation**: limits the depth of the granting privileges.
- How can we keep track of privilege propagation?

Privilege Propagation

- tuna owns:

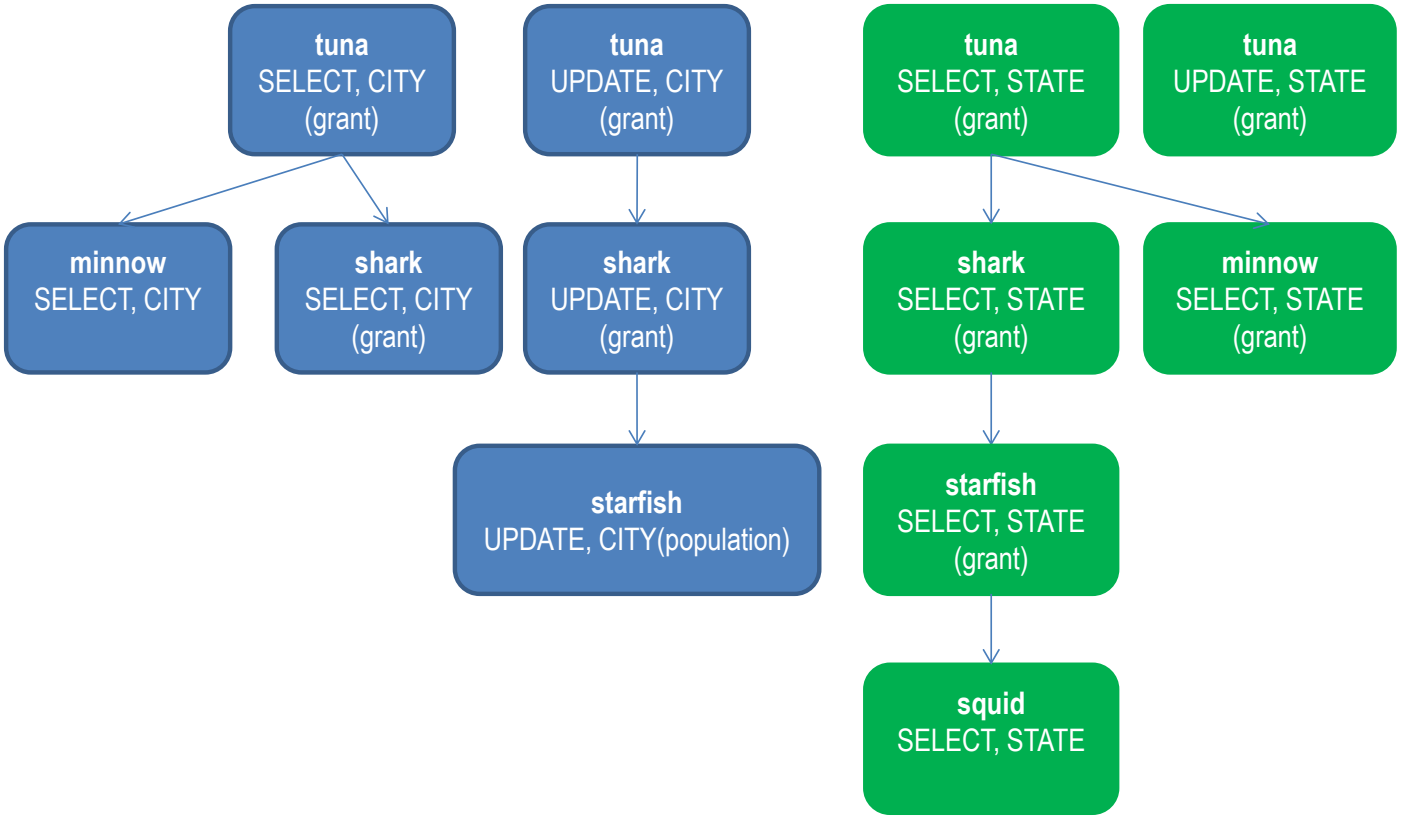
CITY(name, state, population)
STATE(name, abbreviation, capital, area, population)

- The following commands are executed in order:

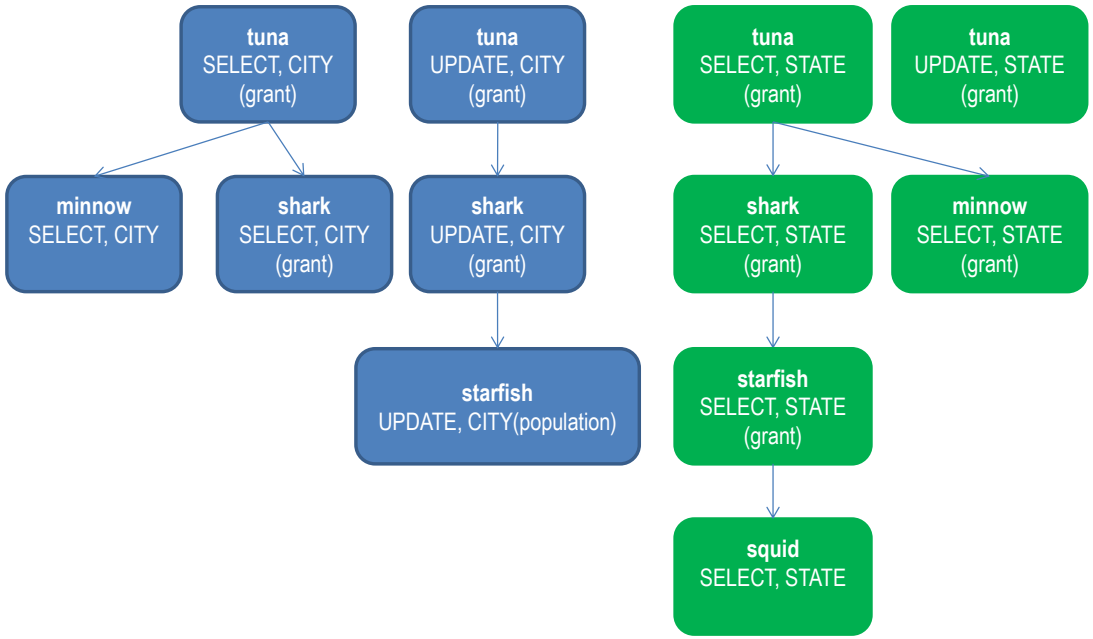
```
    (tuna):  GRANT SELECT, UPDATE ON CITY TO shark WITH GRANT OPTION;
    (tuna):  GRANT SELECT ON CITY TO minnow;
    (tuna):  GRANT SELECT ON STATE TO shark, minnow WITH GRANT OPTION;
   (shark):  GRANT SELECT ON STATE TO starfish WITH GRANT OPTION;
   (shark):  GRANT UPDATE (population) ON CITY TO starfish;
(starfish):  GRANT SELECT ON STATE TO squid;
   (shark):  ...
```

Privilege Propagation

- A **grant graph** can be used to keep track of privilege propagation.



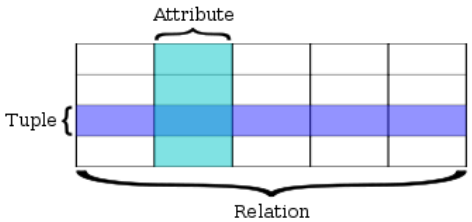
```
    (tuna):  GRANT SELECT, UPDATE ON City TO shark WITH GRANT OPTION;
    (tuna):  GRANT SELECT ON City TO minnow;
    (tuna):  GRANT SELECT ON State TO shark, minnow WITH GRANT OPTION;
   (shark):  GRANT SELECT ON State TO starfish WITH GRANT OPTION;
   (shark):  GRANT UPDATE (population) ON City TO starfish;
(starfish):  GRANT SELECT ON State TO squid;
```



Using Views

```
CREATE VIEW view_name AS
    SELECT attribute_list
      FROM table_list
      ...
```

- Views can be used to create a “window” on a collection of data that is appropriate for some users to access.



Some examples:

1. The owner A of a relation R wants to give a user B read access to some columns of R. A can create a view V_1 that includes only those columns.
2. The owner A of a relation R wants to give a user B read access to some rows of R. A can create view V_2 that selects only those rows from R.

Using Views

- EXAMS(CourseID, StudtID, Grade, Date)

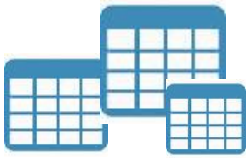
```
(Tom):  CREATE VIEW HardCourses AS
            SELECT CourseID, AVG(Grade) AS Difficulty FROM EXAMS
            GROUP BY CourseID
            Having AVG(Grade)≤50;

(Tom):  CREATE VIEW AllCourses AS
            SELECT CourseID, Grade FROM EXAMS;
```

Database Security - DAC

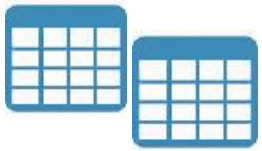
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice

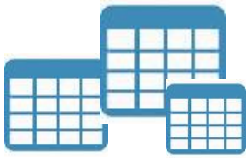


Database Security - DAC

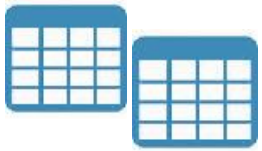
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



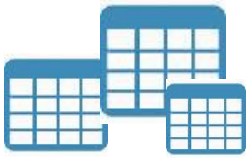
– `GRANT SELECT ON` tableB `TO` Alice;

Database Security - DAC

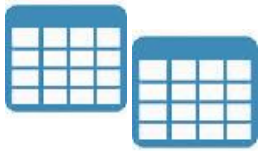
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



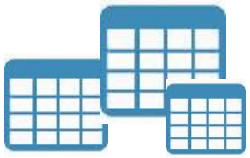
- `GRANT SELECT ON` tableB `TO` Alice;
- `REVOKE SELECT ON` tableB `FROM` Alice;

Database Security - DAC

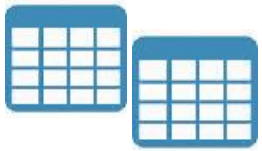
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



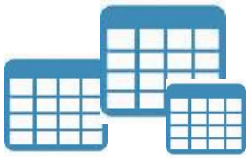
- `GRANT SELECT ON` tableB `TO` Alice;
- `REVOKE SELECT ON` tableB `FROM` Alice;
- `GARNT SELECT ON` tableB `TO` Alice
  `WITH GRANT OPTION`;

Database Security - DAC

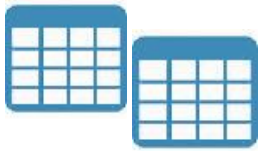
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



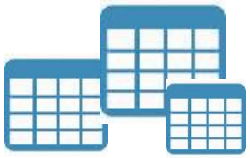
- `GRANT SELECT ON` tableB `TO` Alice;
- `REVOKE SELECT ON` tableB `FROM` Alice;
- `GARNT SELECT ON` tableB `TO` Alice
  `WITH GRANT OPTION`;
- `REVOKE GRANT OPTION FOR SELECT`
  `ON` tableB `FROM` Alice;

Database Security - DAC

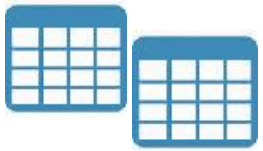
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



– `GRANT SELECT ON` tableB `TO` Alice;

– `REVOKE SELECT ON` tableB `FROM` Alice;

– `GARNT SELECT ON` tableB `TO` Alice
`WITH GRANT OPTION`;

– `REVOKE GRANT OPTION FOR SELECT`
`ON` tableB `FROM` Alice;

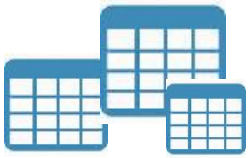
– `GRANT SELECT ON` tableA `TO` Bob;

Database Security - DAC

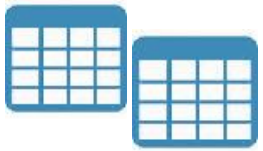
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



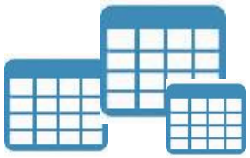
- `GRANT SELECT ON` tableB `TO` Alice;
- `REVOKE SELECT ON` tableB `FROM` Alice;
- `GARNT SELECT ON` tableB `TO` Alice `WITH GRANT OPTION`;
- `REVOKE GRANT OPTION FOR SELECT ON` tableB `FROM` Alice;
- `GRANT SELECT ON` tableA `TO` Bob;
- `REVOKE SELECT ON` tableA `FROM` Bob;

Database Security - DAC

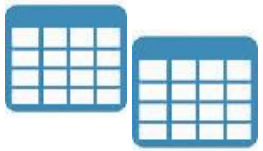
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob



Alice



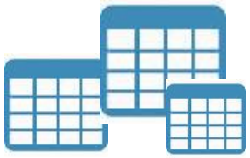
- `GRANT SELECT ON` tableB `TO` Alice;
- `REVOKE SELECT ON` tableB `FROM` Alice;
- `GARNT SELECT ON` tableB `TO` Alice `WITH GRANT OPTION`;
- `REVOKE GRANT OPTION FOR SELECT ON` tableB `FROM` Alice;
- `GRANT SELECT ON` tableA `TO` Bob;
- `REVOKE SELECT ON` tableA `FROM` Bob;
- `GRANT SELECT ON` tableB `TO` Tom;

Database Security - DAC

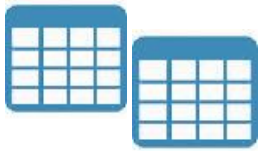
Your objects at your own discretion!

Grant privileges
Revoke privileges
Delegate privileges

Bob

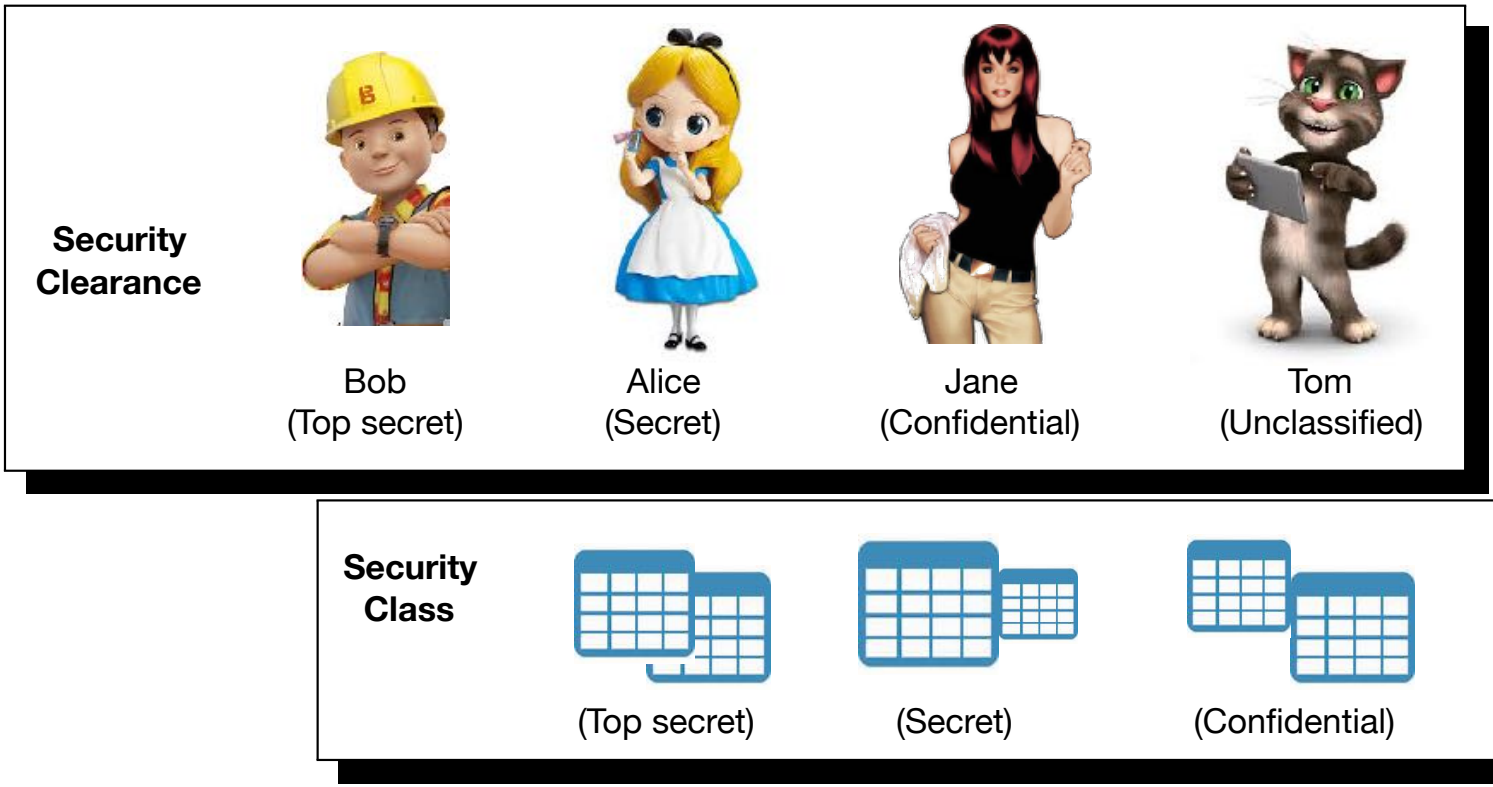


Alice



- `GRANT SELECT ON` tableB `TO` Alice;
- `REVOKE SELECT ON` tableB `FROM` Alice;
- `GARNT SELECT ON` tableB `TO` Alice `WITH GRANT OPTION`;
- `REVOKE GRANT OPTION FOR SELECT ON` tableB `FROM` Alice;
- `GRANT SELECT ON` tableA `TO` Bob;
- `REVOKE SELECT ON` tableA `FROM` Bob;
- `GRANT SELECT ON` tableB `TO` Tom;
- `REVOKE SELECT ON` tableB `FROM` Tom;

Database Security - MAC



System-wide policies govern controlled access to classified information.

Mandatory Access Control

- It is based the **Bell-LaPadula** model (originally developed for U.S. Department of Defense multilevel security policy).
  - Subjects (e.g. users) are assigned *security clearances*;
  - Objects (e.g. rows, tables, views) are assigned *security classes*.

$$TS \geq S \geq C \geq U$$

(TS: Top secret, S: Secret, C: Confidential, U: Unclassified)

Mandatory Access Control

- It is based the **Bell-LaPadula** model (originally developed for U.S. Department of Defense multilevel security policy).
  - Subjects (e.g. users) are assigned *security clearances*;
  - Objects (e.g. rows, tables, views) are assigned *security classes*.

$$TS \geq S \geq C \geq U$$

(TS: Top secret, S: Secret, C: Confidential, U: Unclassified)

- Two rules are enforced by the model:
  1. Subject X can read object Y only if $clearance(X) \geq class(Y)$.
     $\hookrightarrow$ **Read down**
  2. Subject X can write object Y only if $clearance(X) \leq class(Y)$.
     $\hookrightarrow$ **Write up**

Mandatory Access Control

- It is based the **Bell-LaPadula** model (originally developed for U.S. Department of Defense multilevel security policy).
  - Subjects (e.g. users) are assigned *security clearances*;
  - Objects (e.g. rows, tables, views) are assigned *security classes*.

$$TS \geq S \geq C \geq U$$

(TS: Top secret, S: Secret, C: Confidential, U: Unclassified)

- Two rules are enforced by the model:
  1. Subject X can read object Y only if $clearance(X) \geq class(Y)$.
     $\hookrightarrow$ **Read down**
  2. Subject X can write object Y only if $clearance(X) \leq class(Y)$.
     $\hookrightarrow$ **Write up**

- The key idea is "**preventing information in high level objects from flowing to low level subjects**".

Mandatory Access Control

- **Multilevel relations:** Assume that each row is assigned a security class. Then users with different security clearances see a different collection of rows when they access the same table.

city	rating	security class
Paris	4	secret (S)
Canberra	5	confidential (C)

  - Bob with C clearance can only access the second tuple.
  - Peter with S clearance can access both tuples.

Mandatory Access Control

- **Multilevel relations:** Assume that each row is assigned a security class. Then users with different security clearances see a different collection of rows when they access the same table.

city	rating	security class
Paris	4	secret (S)
Canberra	5	confidential (C)

  - Bob with C clearance can only access the second tuple.
  - Peter with S clearance can access both tuples.

- Suppose that city is the primary key, and Bob with C clearance wishes to add a row $(Paris, 4, confidential(C))$.

  1. What would happen?

Mandatory Access Control

- **Multilevel relations:** Assume that each row is assigned a security class. Then users with different security clearances see a different collection of rows when they access the same table.

city	rating	security class
Paris	4	secret (S)
Canberra	5	confidential (C)

  - Bob with C clearance can only access the second tuple.
  - Peter with S clearance can access both tuples.

- Suppose that city is the primary key, and Bob with C clearance wishes to add a row $(Paris, 4, confidential(C))$.

  1. What would happen? The first record may be (partial) inferred.

Mandatory Access Control

- **Multilevel relations:** Assume that each row is assigned a security class. Then users with different security clearances see a different collection of rows when they access the same table.

city	rating	security class
Paris	4	secret (S)
Canberra	5	confidential (C)

  - Bob with C clearance can only access the second tuple.
  - Peter with S clearance can access both tuples.

- Suppose that city is the primary key, and Bob with C clearance wishes to add a row $(Paris, 4, confidential(C))$.
  1. What would happen? The first record may be (partial) inferred.
  2. How to solve the potential security issues?

Mandatory Access Control

- **Multilevel relations:** Assume that each row is assigned a security class. Then users with different security clearances see a different collection of rows when they access the same table.

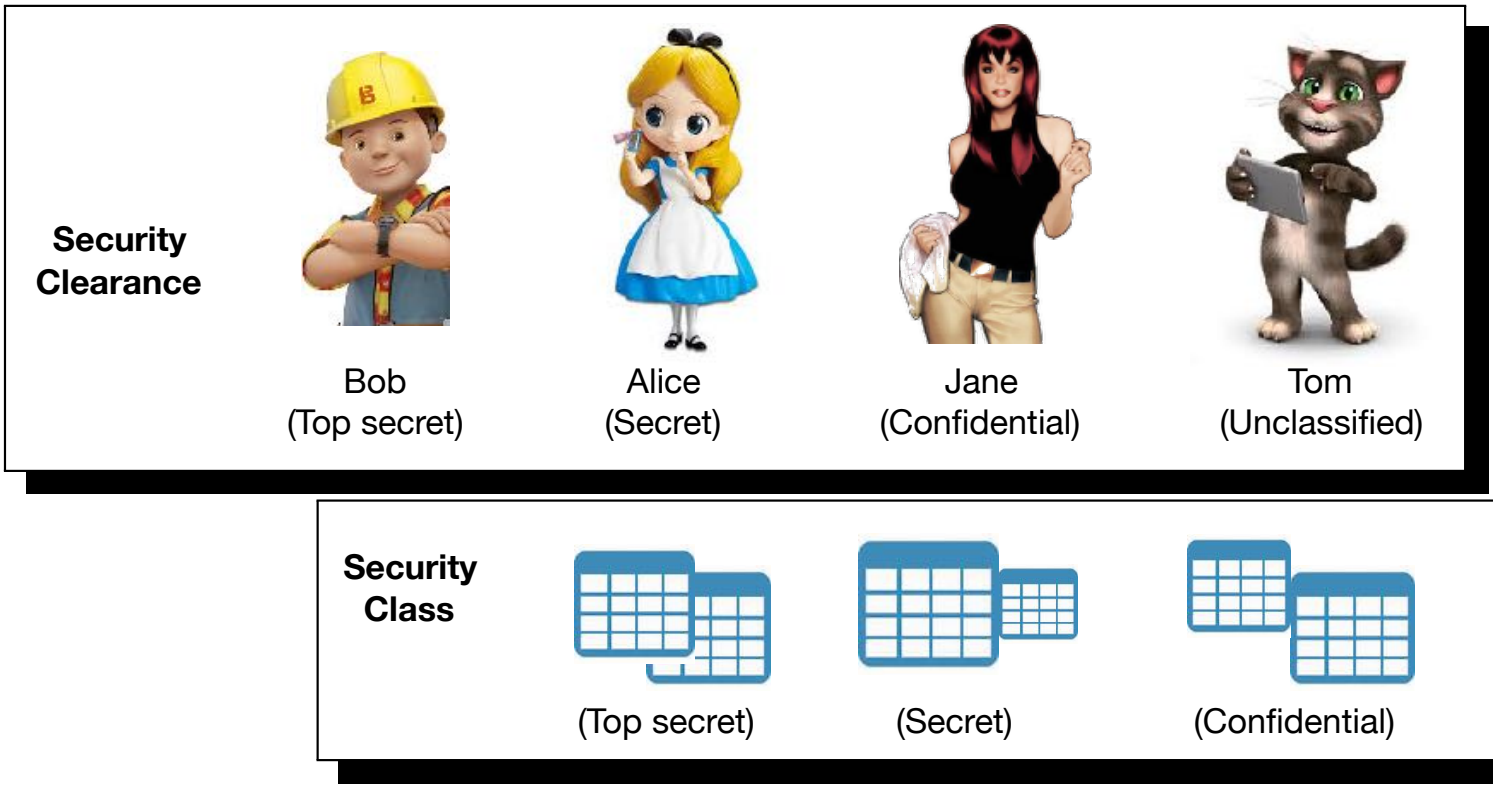
city	rating	security class
Paris	4	secret (S)
Canberra	5	confidential (C)

  - Bob with C clearance can only access the second tuple.
  - Peter with S clearance can access both tuples.

- Suppose that city is the primary key, and Bob with C clearance wishes to add a row $(Paris, 4, confidential(C))$.

  1. What would happen? The first record may be (partial) inferred.
  2. How to solve the potential security issues? whiteTreating security class as part of the primary key.

Database Security - MAC



- **Read down**: Subject X can read object Y only if clearance(X) $\geq$ class(Y).
- **Write up**: Subject X can write object Y only if clearance(X) $\leq$ class(Y).

DAC vs MAC

- How do DAC and MAC differ from each other?

DAC vs MAC

- How do DAC and MAC differ from each other?
  - **DAC is very flexible but complex.**
    - Owners decide how their data is shared.
    - A user may have different privileges on different objects.
    - Different users may have different privileges on the same object.
    - ...

DAC vs MAC

- How do DAC and MAC differ from each other?
  - **DAC is very flexible but complex.**
    - Owners decide how their data is shared.
    - A user may have different privileges on different objects.
    - Different users may have different privileges on the same object.
    - ...
  - **MAC is comparatively rigid.**
    - The system decides how data is shared.
    - Each object is given a security class, and each user is given a security clearance.
    - An object can then be accessed by users with the appropriate clearance.
    - ...

DAC - Limitations

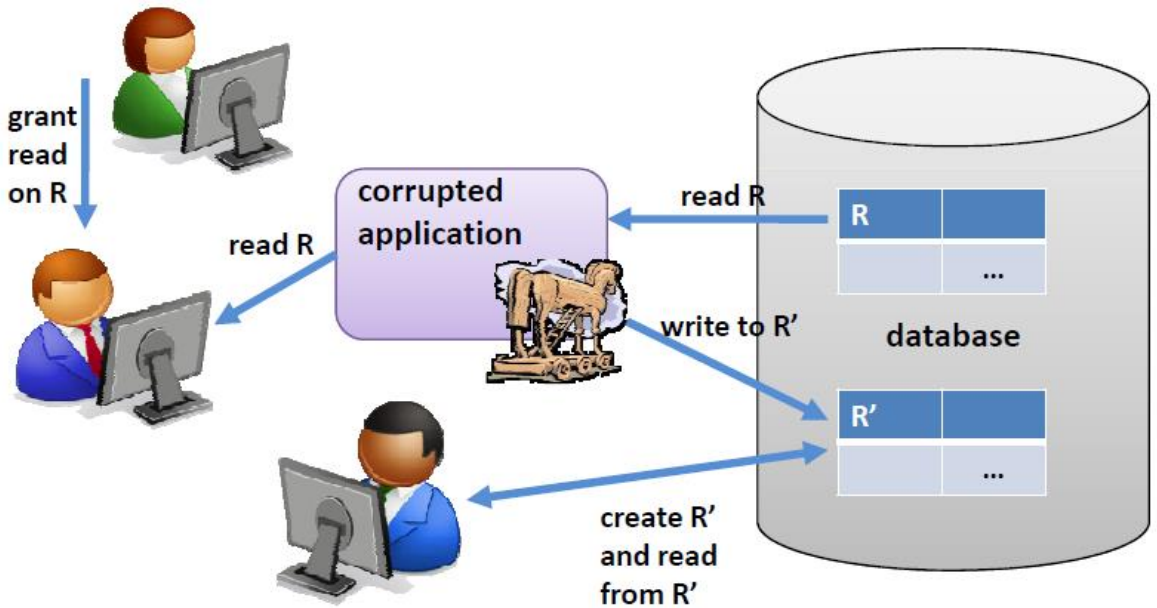
- Suppose that Alice owns a table *R*.
  - Alice gives Bob the `SELECT` privilege to read it, but not Steve. However, Steve may steal the information in *R* from Bob.
  - How?

DAC - Limitations

- Suppose that Alice owns a table *R*.
  - Alice gives Bob the `SELECT` privilege to read it, but not Steve. However, Steve may steal the information in *R* from Bob.
  - How? **Trojan Horse attacks**.

DAC - Limitations

- **Trojan Horse attacks**: If Steve tricks Bob into copying data from table *R* into table *R′*, then the access control on table *R* doesn't apply to the copy of the data in table *R′*.



- Can this problem occur in MAC?

DAC - Limitations

- DAC does not impose any restriction on the usage once data has been obtained by a user, i.e., the dissemination of data is not controlled.
- MAC prevents illegitimate flow of information by attaching security classes to objects and security clearances to subjects.

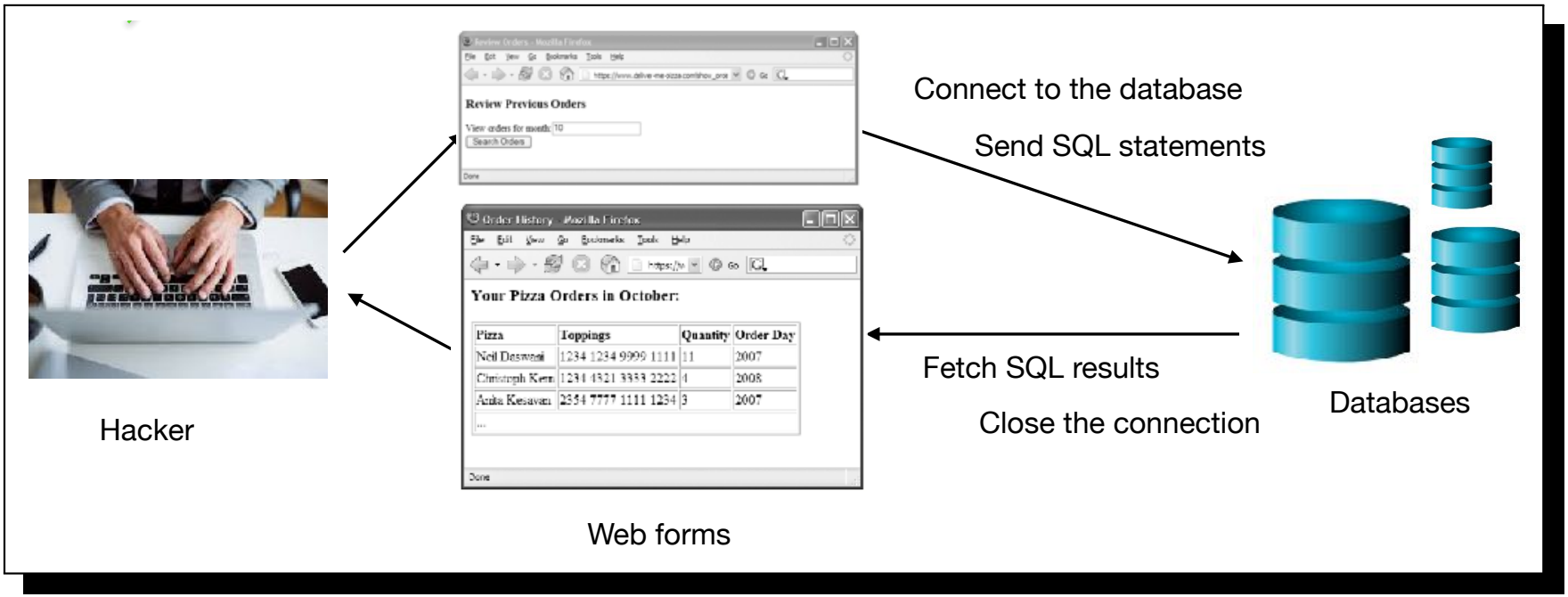
SQL Injection Attacks

SQL Injection Attacks

- SQL injection is one of the most basic and oldest tricks hackers use to get into websites and their backend databases.
- Web applications often access a database by
  1. Connect to the database;
  2. Send SQL statements to the database; ⟵ **SQL Injection!**
  3. Fetch the result and display data from the database;
  4. Close the connection.

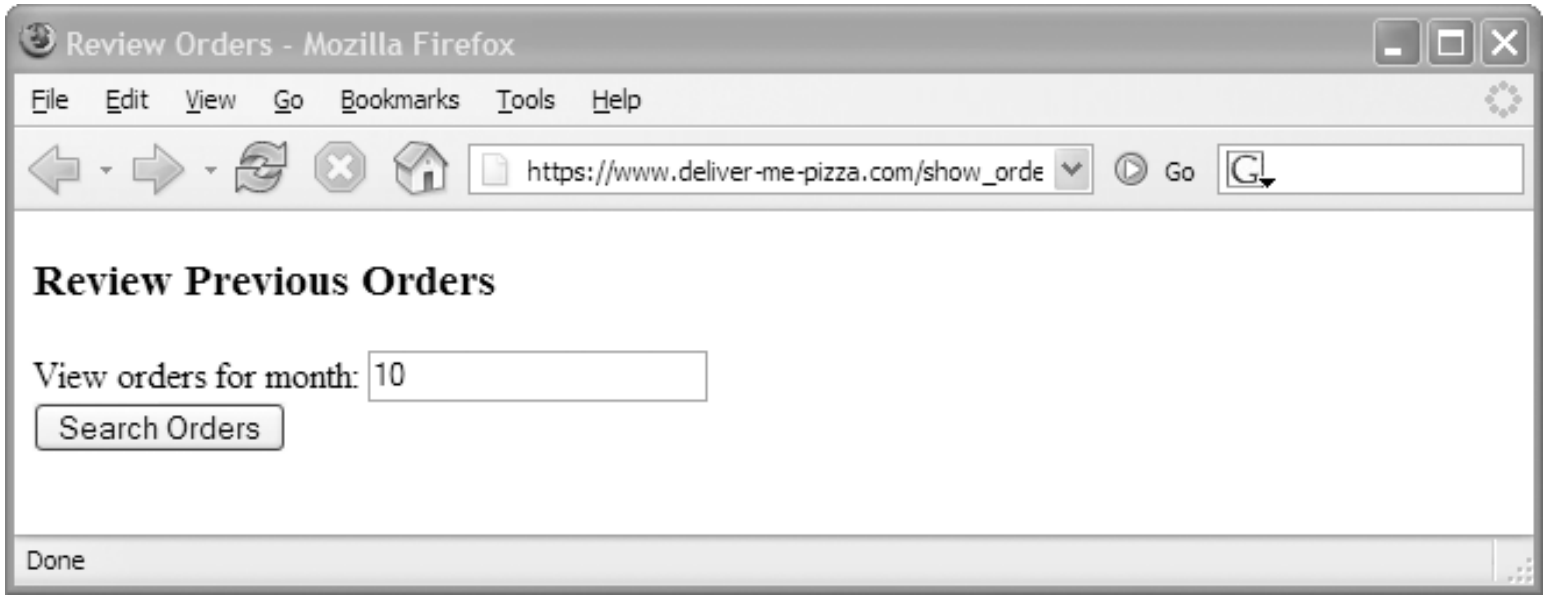
SQL Injection Attacks

- Many web applications take user input from a form.
- A user input is used in constructing a SQL query submitted to a database.
- A SQL injection attack involves manipulating queries through the user input.



SQL Injection - Example

- Consider a pizza-ordering application that allows users to review the orders they have made in a given month.



The pizza order review form

SQL Injection - Example

- When the form is submitted, it results in an HTTP request to the application:

  https://www.deliver-me-pizza.com/show_orders?month=10

- When receiving such a request, the application constructs an SQL query:

```
sql_query = "SELECT pizza, toppings, quantity, order_day "
          + "FROM orders "
          + "WHERE userid=" + session.getCurrentUserId() + " "
          + "AND order_month=" + request.getParameter("month");
```

- Assuming that the current user's userid is 1234, we have:

```
SELECT pizza, toppings, quantity, order_day
  FROM orders
 WHERE userid=1234 AND order_month=10
```

SQL Injection - Example

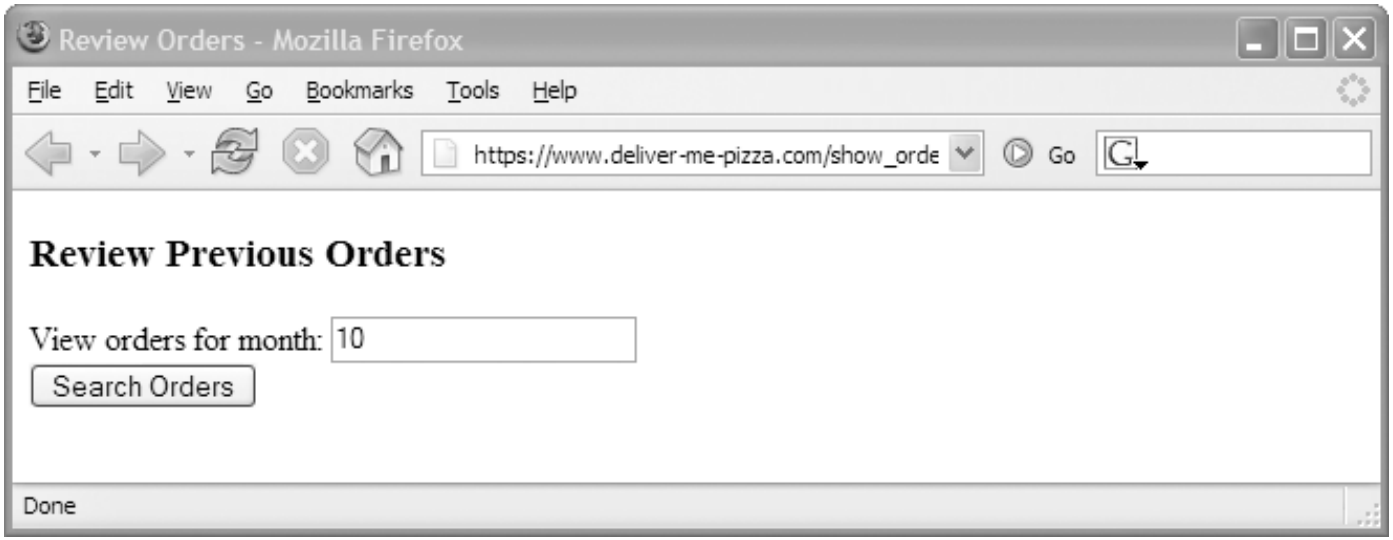
- The application then executes the query and retrieves the result set.

Pizza order history

- How can this application be attacked?

SQL Injection - Example

- What would happen if an attacker replaces '10' with '`0 OR 1=1`'?



- Alternatively, the attacker may modify the HTTP request, e.g.,

https://www.deliver-me-pizza.com/show_orders?month=0%20OR%201%3D1

Then request.getParameter("month") extracts '0%20*OR*%201%3*D*1' and returns the string '`0 OR 1=1`'.

SQL Injection - Example

- The SQL query that the application constructs and sends to the database now becomes:

```
SELECT pizza, toppings, quantity, order_day
  FROM orders
 WHERE userid=4123 AND order_month=0 OR 1=1
```

- Since the operator precedence of the `AND` operator is higher than that of `OR`, the `WHERE` condition is equivalent to

```
WHERE (userid=4123 AND order_month=0) OR 1=1
```

- What happened?

  The (malicious) user supplied a parameter that, once inserted into the SQL query string, actually altered the meaning of the query!

SQL Injection - Example

- However, the attacker might be able to do even more damage, e.g., making a request such that the request parameter month evaluates to:

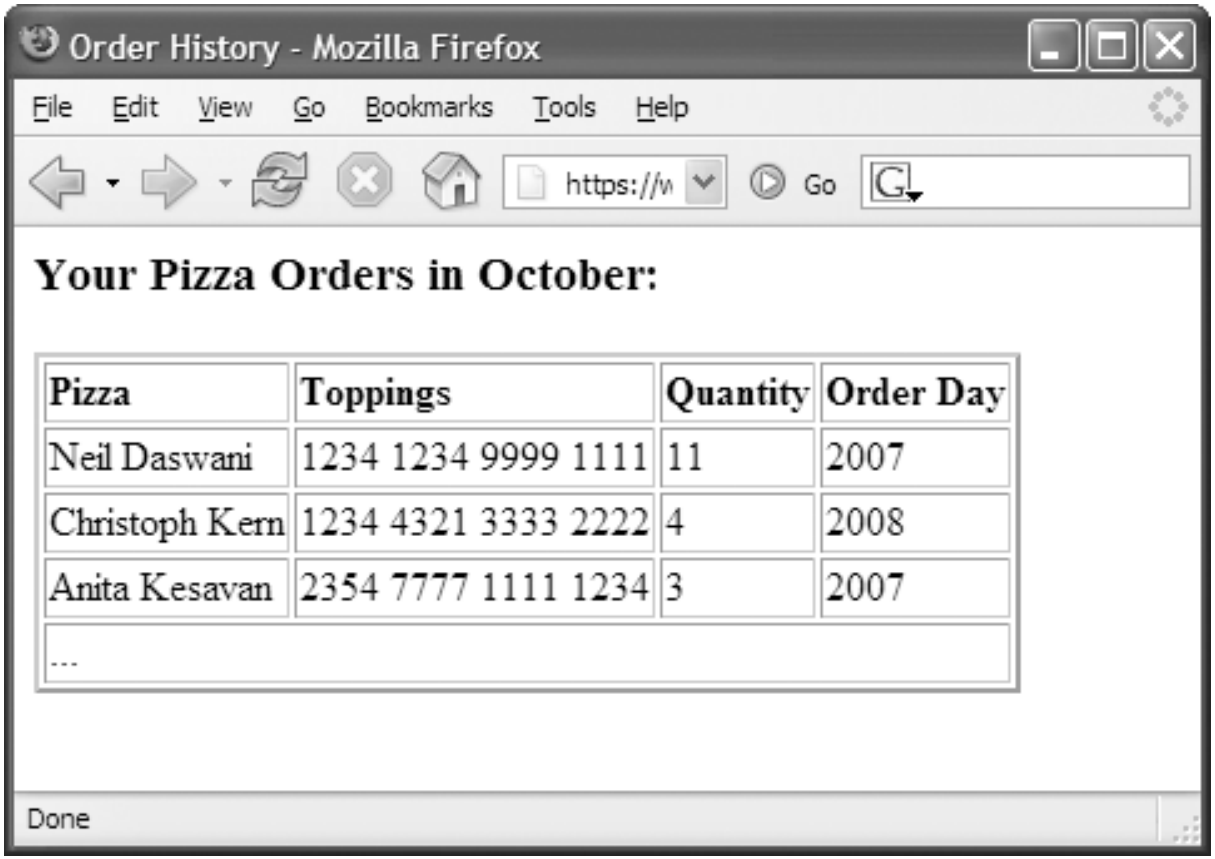
```
0 AND 1=0
UNION
SELECT cardholder, number, exp_month, exp_year
FROM creditcards
```

- Then, the SQL query that the application constructs and sends to the database becomes:

```
SELECT pizza, toppings, quantity, order_day
FROM orders
WHERE userid=4123 AND order_month=0 AND 1=0
UNION
SELECT cardholder, number, exp_month, exp_year
FROM creditcards
```

SQL Injection - Example

- As a result, the attacker receives an HTML page that contains the entire content of the creditcards table.



SQL Injection Attacks

- How can we prevent SQL injection attacks?
- Can SQL injection attacks be prevented by any of the following security solutions?

1. **Firewall**

   i.e., monitors and controls the incoming and outgoing network traffic based on predetermined security rules

2. **Intrusion detection system** (IDS)

   i.e., monitors a network or systems for malicious activity or policy violations

3. **Authentication**

   i.e., the process by which a system can identify users

SQL Injection Attacks - Protection Techniques

- Several techniques of input validation:
  - **Blacklisting**?

    i.e., blacklist quotes, semicolons, etc. from the input string. However, if you forget to blacklist just one type of dangerous character, it could give rise to a successful attack.
  - **Whitelisting**?

    i.e., explicitly test whether a given input is within a well-defined set of values that are known to be safe, e.g., the parameter month is a string that represents a non-negative integer.
  - **Escaping**?

    i.e., transform dangerous input characters to turn a potentially dangerous input string into a sanitized one, e.g., escape(o‘connor)= o“connor (the double quote is the escaped version of the single quote).

SQL Injection Attacks - Protection Techniques

- The recommended solution: **Parameterized Queries**

  Two steps:

  1. The statement is prepared (parsed and compiled), in which ? is used as place-holders for the actual parameters.
  2. The actual parameters are passed to the prepared statement for execution.

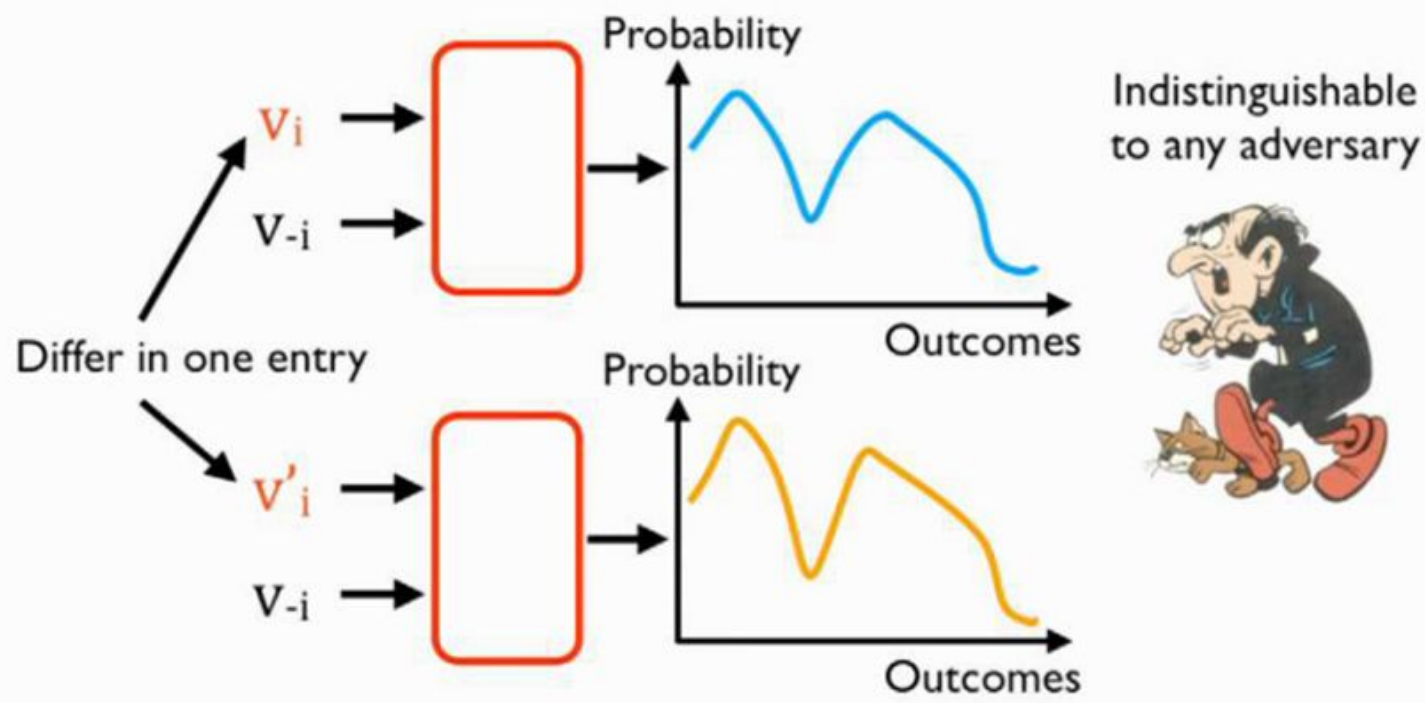
```
PreparedStatement stmt=conn.prepareStatement(
        "SELECT pizza, toppings, quantity, order_day "
        + "FROM orders WHERE userid=? AND order_month=?");
stmt.setInt(1, session.getCurrentUserId());
stmt.setInt(2, Integer.parseInt(request.getParamenter("month")));

ResultSet res = ps.executeQuery();
```

- The key idea is "**separation between control and data**"!

Research Topics

Research Topics

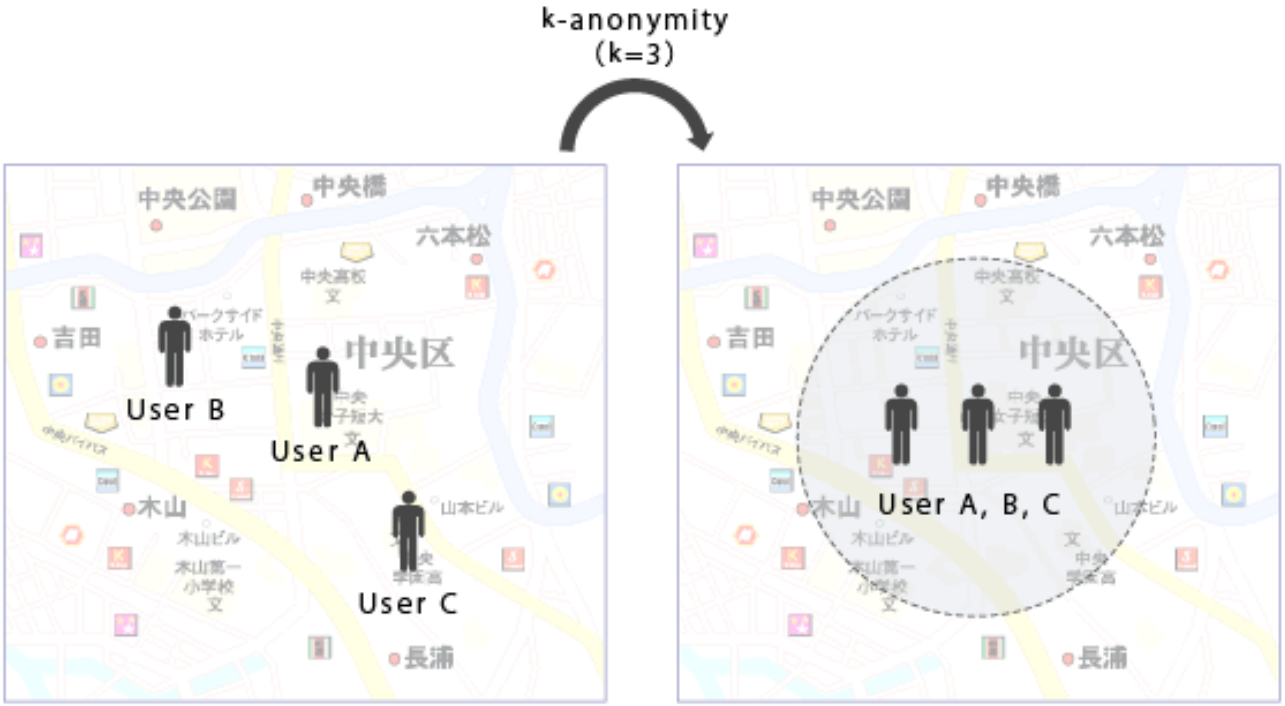
- Differential privacy



- What Apple's differential privacy means for your data and the future machine learning: *https://techcrunch.com/2016/06/14/differential-privacy/*
- Learning with privacy at scale: *https://machinelearning.apple.com/research/learning-with-privacy-at-scale*

Research Topics

- Data anonymization



- k-anonymity: *https://en.wikipedia.org/wiki/K-anonymity*