Lecture 12

Database Security

Dr. Vandana Kushwaha

Department of Computer Science
Institute of Science, BHU, Varanasi

Types of Security

- **Database security** is a broad area that addresses **many issues**, including the following:
  - **Various legal and ethical issues** regarding the **right to access** certain information—for example, **some information** may be deemed to be **private** and cannot be accessed legally by unauthorized organizations or persons.
  - **Policy issues** at the **governmental, institutional, or corporate** level as to what kinds of information should not be made—for example, credit ratings and personal medical records. **publicly available**
  - **Security Levels:** The need in some organizations to identify **multiple security levels** and to **categorize** the **data** and **users** based on these **classifications**—for example: **Top secret, Secret, Confidential, and Unclassified.**

Threats to Databases

- **Threats to databases** can result in the **loss or degradation** of some or all of the following commonly accepted **security goals:**
  - *Integrity,*
  - *Availability,*
  - *Confidentiality.*

Threats to Databases

- **Loss of Integrity**
  - **Database integrity** refers to the **requirement** that **information be protected** from **improper modification.**
  - **Modification** of data includes **creation, insertion, updating, changing** the **status of data**, and **deletion.**
  - **Integrity is lost** if **unauthorized changes** are made to the **data** by either **intentional** or **accidental acts.**
  - If the **loss of data integrity** is **not corrected**, continued use of the **contaminated system** or **corrupted data** could result in **inaccuracy, fraud**, or **erroneous decisions.**

Threats to Databases

- **Loss of Availability**
  - **Database availability** refers to making objects **available** to a **human user** or a **program** to which they have a **legitimate right.**
- **Loss of Confidentiality**
  - **Database confidentiality** refers to the **protection of data** from **unauthorized disclosure.**
  - The **impact** of **unauthorized disclosure** of **confidential information** can range from **violation of the Data Privacy Act** to the jeopardization of **national security**.
  - Unauthorized disclosure could result in **loss of public confidence, embarrassment, or legal action against the organization.**

Control Measure

- To **protect databases** against these types of **threats,** it is common to implement **four kinds** of **Control measures:** ***Access control, Inference control, Flow control, and Encryption.***
- Access Control
  - A **security problem** common to **computer systems** is that of **preventing unauthorized persons from accessing the system itself**, either to **obtain information** or to **make malicious changes** in a portion of the **database.**
  - The **security mechanism** of a **DBMS** must include **provisions for restricting access** to the **database system** as a whole.
  - This function, called **Access control**, is **handled** by **creating user accounts** and **passwords** to **control** the **login process** by the **DBMS.**

Control Measure

- **Inference control**
  - **Statistical databases** are used to provide **statistical information** or **summaries** of values based on various criteria.
  - **Security** for **statistical databases** must ensure that **information about individuals cannot be accessed.**
  - It is **sometimes possible** to **deduce or infer certain facts concerning individuals** from **queries** that **involve only summary statistics** on **groups;** consequently, this **must not be permitted either.**

Control Measure

- **Flow Control**
  - **Flow control**, which **prevents information from flowing in such a way** that it **reaches unauthorized users.**
  - **Channels** that are **pathways for information to flow implicitly** in ways that **violate the security policy** of an **organization** are called **covert channels.**
- **Encryption**
  - A final **control measure** is **data encryption**, which is used to **protect sensitive data** (such as credit card numbers) that is **transmitted** via some type of **communications network.**
  - **Encryption** can be used to provide **additional protection** for **sensitive portions of a database** as well.

Database Security Mechanisms

- **Discretionary Security mechanisms**
  - These are used to **grant privileges** to **users**, including the capability to **access specific data files, records, or fields** in a **specified mode** (such as **read, insert, delete, or update**).
- **Mandatory Security mechanisms**
  - These are used to **enforce multilevel security** by **classifying the data** and **users** into **various security classes** (or **levels**) and then implementing the **appropriate security policy** of the **organization.**
  - An extension of this is **role-based security**, which **enforces policies** and **privileges** based on the **concept of organizational roles.**

Database Security and the DBA

- The **database administrator (DBA)** is the **central authority** for **managing** a **database system.**
- The **DBA's responsibilities** include:
  - **Granting privileges** to **users** who need to use the system and
  - **Classifying users** and **data in accordance** with the **policy of the organization.**
- The **DBA** has a **DBA account** in the **DBMS**, sometimes called a **system or super user account**, which provides **powerful capabilities** that are not made available to regular database accounts and users.
- **DBA** has the following **rights** for maintaining the **Database security:**
  - **1. Account creation**. This action creates a new **account** and **password** for a **user** or a **group of users** to enable **access** to the **DBMS.**

Database Security and the DBA

- **2. Privilege granting**. This action permits the **DBA** to **grant certain privileges** to certain accounts.
- **3. Privilege revocation**. This action permits the **DBA** to **revoke (cancel)** certain **privileges** that were previously given to certain accounts.
- **4. Security level assignment**. This action consists of **assigning user accounts** to the appropriate **security clearance level.**

- **Action 1(Account creation)** in the preceding list is used to **control access** to the **DBMS** as a whole.
- Whereas **Actions 2(Privilege granting**) and **Action 3(Privilege revocation)** are used to **control Discretionary database authorization**, and
- **Action 4**(**Security level assignment**) is used to **control Mandatory authorization.**

Discretionary Access Control

- The typical method of enforcing **discretionary access control** in a **database system** is based on the **granting** and **revoking** of privileges.

Types of Discretionary Privileges

- Informally, there are **two levels** for **assigning privileges** to use the database system:
  - **The Account level.**
  - At this level, the **DBA specifies** the **particular privileges** that **each account holds** independently of the relations in the database. **Ex.** Create Table, Create View, Alter Table, Delete Table etc.
  - **The Relation (or table) level.**
  - At this level, the **DBA can control the privilege** to **access** each **individual relation or view** in the database.

Discretionary Access Control

- **Specifying Privileges through the Use of Views**
  - The mechanism of **views** is an **important discretionary authorization mechanism** in its own right.
  - For **example,** if the **owner A** of a **relation R** wants another **account B** to be able to retrieve only **some fields of R(X,Y,Z)**, then **A** can create a **view V** of **R** that includes only those attributes as:
  - **Create View V** as Select **X,Y** from **R**
  - And then can assign the **Select privilege** to **account B** as:
  - **GRANT SELECT on V to B.**

Discretionary Access Control

- The same applies to **limiting B** to **retrieving only certain tuples** of **R**.
- A **view V’** can be created by **defining the view** by means of a **query that selects only those tuples from R** that **A wants to allow B to access:**
- **Create View** **V’** as **Select X,Y** from **R** where **Y>5000**
- And then can assign the **Select privilege** to **account B** as:
- **GRANT** **SELECT on V’ to B.**

Discretionary Access Control

- **Revoking of Privileges**
  - In some cases it is **desirable to grant a privilege** to a **user temporarily.**
  - For **example**, the **owner of a relation** may want to **grant the SELECT privilege to a user for a specific task** and then **revoke that privilege once the task is completed.**
  - Hence, a **mechanism for revoking privileges** is needed.
  - In **SQL** a **REVOKE command** is included for the purpose of **canceling privileges.**

Propagation of Privileges Using the GRANT OPTION

- Whenever the **owner A** of a relation **R grants** a **privilege** on **R** to another **account B**, the **privilege** can be given to **B with or without** the **GRANT OPTION.**
- If the **GRANT OPTION is given**, this means that **B can also grant that privilege on R to other accounts.**
- Suppose that **B is given the GRANT OPTION** by **A and that B then grants the privilege on R to a third account C**, also with the **GRANT OPTION.**
- In this way, **privileges** on **R** can **propagate** to other **accounts without the knowledge of the owner of R.**
- If **the owner account A** now **revokes** the **privilege granted** to **B**, **all the privileges** that **B propagated** based on that privilege should **automatically be revoked** by the system.

Example

- Suppose that the **DBA** creates **four accounts—A1, A2, A3**, and **A4**—and wants **only A1** to be able to create base relations**.**
- To do this, the **DBA must issue** the following **GRANT command in SQL:**
  - **GRANT CREATETAB TO A1;**
- Suppose that **account A1** wants to **grant to account A2** the **privilege to insert and delete tuples** in these relations.
- However, **A1 does not want A2 to be able to propagate these privileges to additional accounts**.
- **A1 can issue** the following **command:**
  - **GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;**

Example

- Suppose that **A1 wants to allow account A3** to **retrieve information from either of the two tables** and also to **be able to propagate the SELECT privilege to other accounts.**
- **A1 can issue** the following command:
  - **GRANT SELECT ON EMPLOYEE, DEPARTMENT TO A3 WITH GRANT OPTION**
- **A3** can grant the **SELECT privilege** on the **EMPLOYEE** relation to **A4** by issuing the following **command:**
  - **GRANT SELECT ON EMPLOYEE TO A4;**
- Notice that **A4 cannot propagate the SELECT privilege** to **other accounts** because the **GRANT OPTION** was **not given to A4.**

Example

- Now suppose that **A1 decides to revoke the SELECT privilege** on the **EMPLOYEE relation from A3; A1** then can issue this **command:**
  - **REVOKE SELECT ON EMPLOYEE FROM A3;**
- The **DBMS** must now **revoke the SELECT privilege on EMPLOYEE from A3**, and it must **also automatically revoke the SELECT privilege on EMPLOYEE from A4.**
- This is because **A3 granted that privilege to A4, but A3 does not have the privilege any more.**

Example

- Next, suppose that **A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation** and wants to **allow A3 to be able to propagate the privilege.**
- The limitation is to **retrieve only the Name, Bdate, and Address attributes** and **only for the tuples with Dno = 5.**
- **A1** then can **create** the **following view:**
  - **CREATE VIEW A3EMPLOYEE AS SELECT Name, Bdate, Address FROM EMPLOYEE WHERE Dno = 5;**
- After the **view** is created, **A1 can grant SELECT privilege on the view A3EMPLOYEE to A3** as follows:
  - **GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION**

Example

- Finally, suppose that **A1 wants to allow A4 to update only the Salary attribute** of **EMPLOYEE**.
- **A1 can then issue** the following **command:**
  - **GRANT UPDATE ON EMPLOYEE (Salary) TO A4;**
- The **UPDATE and INSERT privileges** can **specify particular attributes** that may be **updated or inserted in a relation.**
- **Other privileges (SELECT, DELETE)** are **not attribute specific**, because this specificity can easily be controlled by **creating the appropriate views** that include only the desired attributes and granting the corresponding privileges on the views.
- However, **because updating views is not always possible**, the **UPDATE and INSERT privileges** are given the option to specify the **particular attributes** of a **base relation** that **may be updated.**

Drawback of DAC

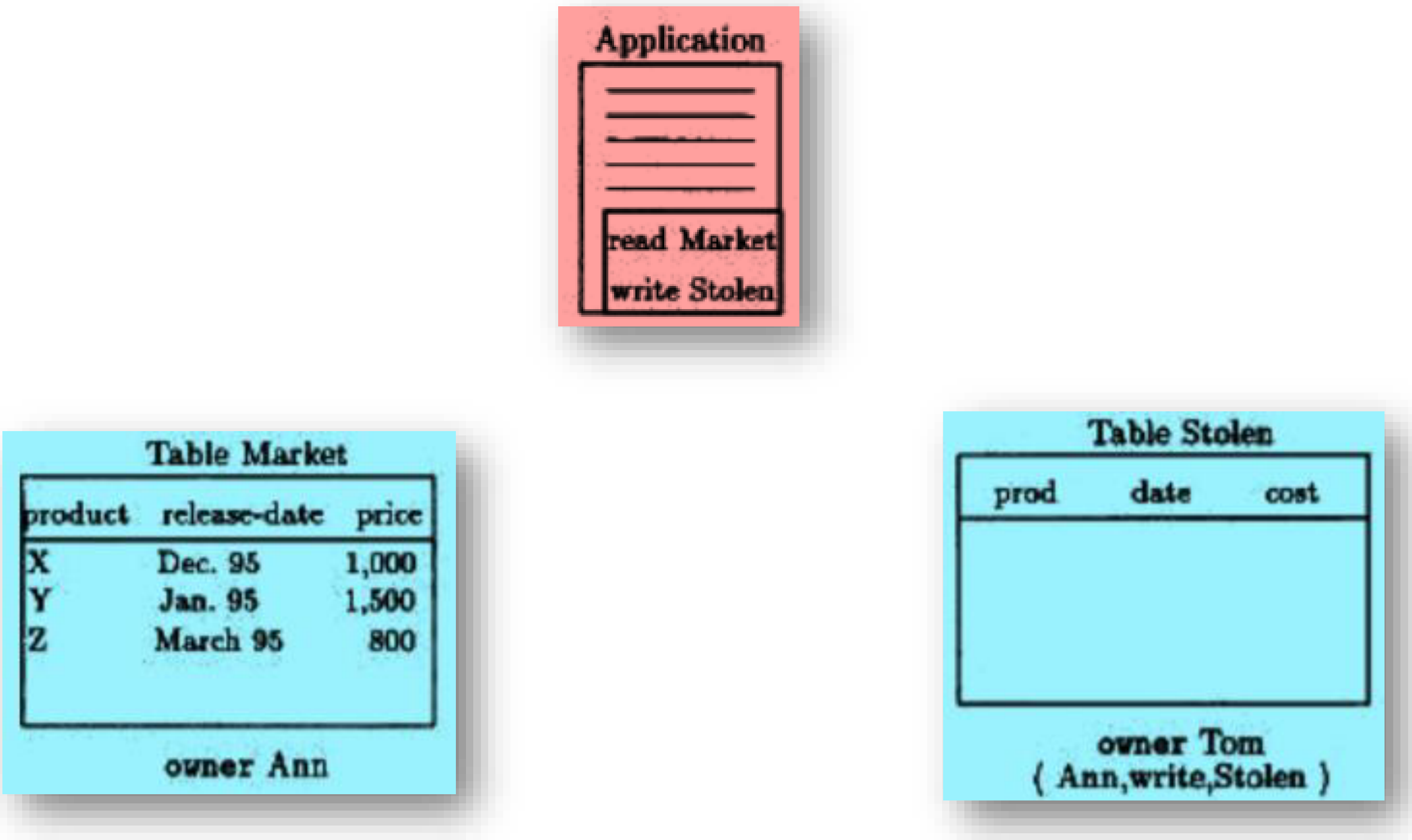
- The main **drawback** of **DAC** is that although each **access** is **controlled** and allowed only if authorized, it is **possible to bypass the access** restrictions.
- A **user** who is **able to read data** can **pass the data** to **other user not authorized** to **read the data** without the cognizance of the **data owner.**
- This weakness makes **DAC vulnerable** to **malicious attacks** such as **Trojan Horses.**
- A **Trojan horse** is a **computer program** with an **apparently** or actually **useful function**, which contains **additional hidden functions** that **secretly exploit** the **legitimate authorizations** of the invoking process.
- The **Trojan Horse Attacks** can be understood by the **example** of an **organization.**

Trojan Horse Attacks

- Suppose a **top-level manager** named **Ann** creates a **table market** containing **sensitive information.**
- **Tom,** one of **Anns subordinate**, who also **works secretly** for **another organization wants this information.**
- To achieve this, **Tom secretly creates a table stolen** and **gives write privilege** to **Ann** on this **table stolen.**
- **Ann** doesn't even know about the existence of this **table stolen** and **having privilege** on the **table stolen.**
- **Tom** also **secretly modifies worksheet application** to **include two hidden operations:**
  - A **read operation** on the **table market**.
  - A **write operation** on the **table stolen.**

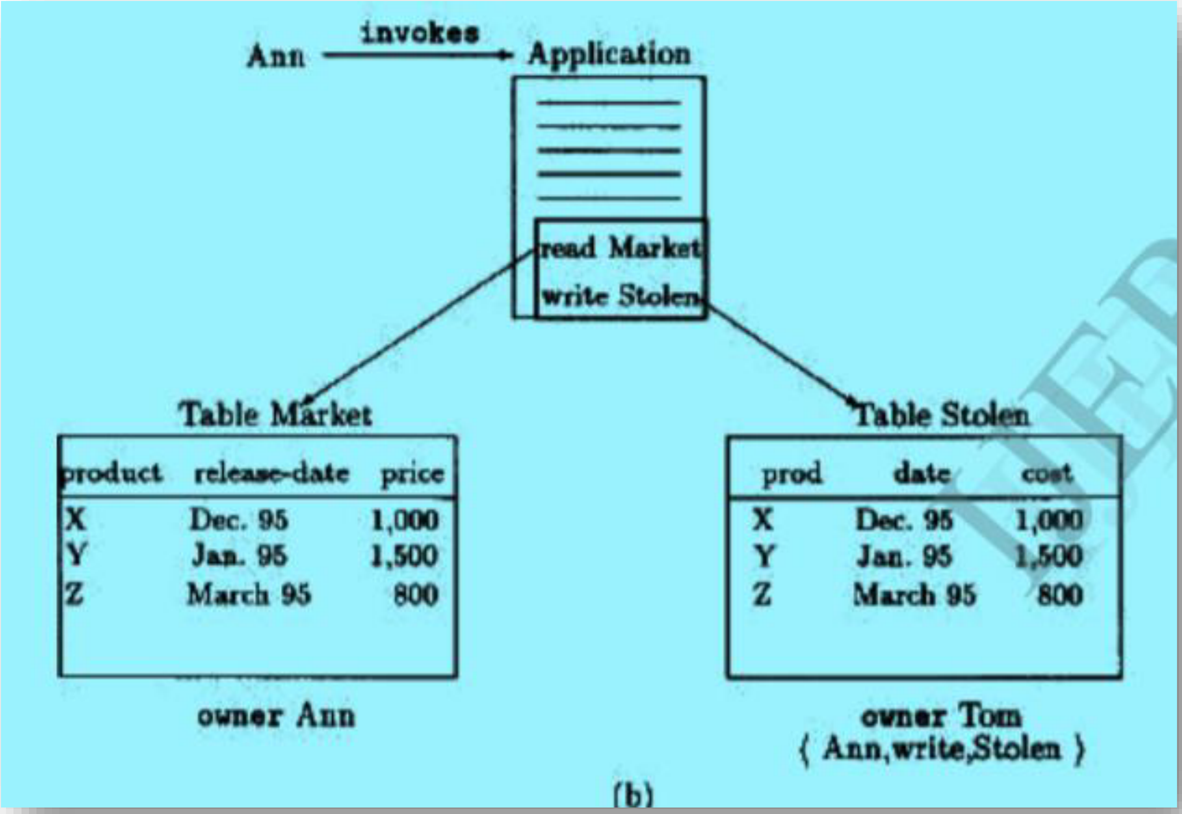
Trojan Horse Attacks

- As shown in fig below. **Tom** then **gives this new application** to his manager **Ann.**



Trojan Horse Attacks

- As a **result** during **execution**, **sensitive information** is **copied** from the **market table** to **stolen table** and becomes **available to dishonest employee Tom** who can now **misuse this information**.



Mandatory access control

- The **Discretionary access control technique** of granting and revoking privileges on relations has **traditionally been the main security mechanism** for **relational database systems.**
- This is an **all-or-nothing method:** A user either has or does not have a certain privilege.
- In **many applications**, an **additional security policy is needed** that **classifies data and users based on security classes.**
- This approach, known as **Mandatory Access Control (MAC),** would typically be **combined with the Discretionary access control mechanisms.**
- It is important to note that **most commercial DBMSs currently provide mechanisms only for discretionary access control.**

Mandatory access control

- However, the **need for multilevel security exists** in **government, military, and intelligence applications,** as well as in many industrial and corporate applications.
- Some DBMS vendors—for example, **Oracle**—have released special versions of their **RDBMSs** that **incorporate Mandatory access control** for **government use.**
- **Typical security classes** are **top secret (TS), secret (S), confidential (C), and unclassified (U),** where **TS is the highest level and U the lowest,** where
  - **TS ≥ S ≥ C ≥ U.**
- The commonly used **model for multilevel security**, known as the **Bell-LaPadula model**:
- It classifies each **subject (users account, program)** and **object (relation, tuple, column, view)** into one of the **security classifications TS, S, C, or U.**

The Bell-LaPadula Security Policy Model

- Proposed by **David Bell** and **Len Lapadula** in **1973,** in response to U.S. Air Force concerns over the security of time-sharing mainframe systems.
- This **model** is the **most widely** recognized **MLS model.**
- The model deal with **confidentiality** only.
- **Two properties**:
- **1. Simple security property:**
  - **Subject S** is allowed to **read** **object O** only if **class(O) ≤ class(S).**
  - The **first property** enforces the obvious rule that **no subject can read an object** whose **security classification** is **higher** than the **subject's security clearance.**

The Bell-LaPadula Security Policy Model

2. A **subject S** is **allowed to write an object O** only if **class(S) ≤ class(O).**

- This is known as the **star property (or *-property).**
- It **prohibits** a **subject** from **writing an object** at a **lower security classification** than the **subject's security clearance.**
- **Violation** of this **rule** would **allow information to flow from higher to lower classifications**, which violates a basic tenet of **multilevel security**.
- For **example**, a **user (subject)** with **TS clearance** may make a **copy** of an **object** with **classification TS** and then **write it back** as a **new object** with **classification U**, thus **making** it **visible throughout** the **system**.

The Bell-LaPadula Security Policy Model

Rule: Allow **information to flow Only** from **lower to higher classifications**.

- **Read Operation**: Subject ← Object
- **Write Operation:** Subject → Object

	Subject	Object	Database Operations
TS		X	**A** can't **read X** as **class(A) < class(O)**
S	A		**A** can **read O** as **class(A) > class(O)**
C		O	**A** can't **write O** as **class(A) > class(O)**
U	S		**S** can **write O** as **class(S) < class(O)**

Multilevel Security

- To incorporate **multilevel security** notions into the **relational database model**, it is common to consider **attribute values** and **tuples** as **data objects.**
- Hence, each **attribute value** in **a tuple** is **associated** with a corresponding **security classification.**
- In addition, a **tuple classification attribute TC** is added to **each tuple.**
- Hence, a **multilevel relation schema R** with **n attributes** would be represented as :
  - **R(A1 ,C1 ,A2 ,C2 , …, An ,Cn ,TC)**
- where each **Ci** represents the **classification attribute** associated with **attribute Ai .**
- The **value** of the **TC attribute** in each **tuple t** – which is the **highest of all attribute classification values** within **t** – provides a general classification for the tuple itself, whereas **each Ci** provides a **finer security classification** for **each attribute value** within the tuple.

Multilevel Security

- The **apparent key** of a **multilevel relation** is the **set of attributes** that would have formed the **primary key** in a **regular(single-level) relation.**
- A **multilevel relation** shown in Figure, where we display the classification attribute values next to each attribute's value.

EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Brown C	80000 S	Good C	S

- Assume that the **Name attribute** is the **apparent key**, and consider the **query** SELECT * FROM EMPLOYEE.

Multilevel Security

- Appearance of EMPLOYEE after filtering for **classification C users:**

EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	40000	C	Fair	S	S
Brown	C	80000	S	Good	C	S

→

EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	40000	C	NULL	C	C
Brown	C	NULL	C	Good	C	C

- Appearance of EMPLOYEE after filtering for **classification U users**.

EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	40000	C	Fair	S	S
Brown	C	80000	S	Good	C	S

→

EMPLOYEE

Name		Salary		JobPerformance		TC
Smith	U	NULL	U	NULL	U	U

SQL Injection attack

- In an **SQL Injection attack**, the **attacker injects** a **string input** through the application, which **changes or manipulates** the **SQL statement** to the **attacker's advantage.**
- An **SQL Injection attack** can **harm** the database in various ways, such as **unauthorized manipulation** of the database, or **retrieval** of **sensitive data**.
- It can also be used to **execute system level commands** that may cause the system to **deny service to the application.**
- For **example**, suppose that a simplistic authentication procedure issues the following query and checks to see if any rows were returned:
- *SELECT * FROM users WHERE username = 'jake' and PASSWORD = 'jakespasswd'.*

SQL Injection attack

- The **attacker** can try to **change (or manipulate)** the **SQL statement**, by changing it as follows:

  *SELECT * FROM users*

  WHERE username = ‘jake’ and (PASSWORD = ‘jakespasswd’ or ‘x’ = ‘x’)

- As a result, the **attacker** who knows that **‘jake’** is a valid **login** of some user is able to log into the database system as ‘jake’ without knowing his password and is able to do everything that ‘jake’ may be authorized to do to the database system.

Statistical Database Security

- **Statistical databases** are used mainly to **produce statistics** about various populations.
- The **database** may contain **confidential data** about **individuals,** which should be protected from user access.
- However, **users are permitted** to **retrieve statistical information** about the populations, such as *averages, sums, counts, maximums, minimums, and standard deviations*.
- **Statistical database security techniques** must **prohibit** the **retrieval of individual data**.
- This can be achieved by **prohibiting queries** that **retrieve attribute values** and by **allowing only queries** that involve **statistical aggregate functions** such as *COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION.*
- Such queries are sometimes called **statistical queries.**

Statistical Database Security

- In some cases **it is possible to infer the values of individual tuples** from a sequence of **statistical queries.**
- This is particularly **true** when the **conditions result** in a population consisting of a **small number of tuples**.
- As an **illustration**, consider the following **statistical queries:**
  - ***Q1:* *SELECT COUNT (*) FROM PERSON WHERE ;***
  - ***Q2:* *SELECT AVG (Income) FROM PERSON WHERE ;***
- Now suppose that we are interested in finding the **Salary** of **Jane Smith,** and we know that **she has** a **Ph.D. degree** and that **she lives** in the **city of Bellaire**, **Texas.**
- We issue the **statistical query Q1** with the following condition: *(Last_degree='Ph.D.' AND Gender='F' AND City='Bellaire' AND State='Texas')*

Statistical Database Security

- If we get a **result of 1** for this **query**, we can **issue Q2** with the **same condition** and **find the Salary** of **Jane Smith.**
- Even if the **result of Q1** on the preceding condition is **not 1** but is a **small number—say 2 or 3**—we can **issue statistical queries** using the **functions MAX, MIN,** and **AVERAGE** to identify the **possible range of values** for the **Salary** of **Jane Smith.**
- The *possibility of inferring* **individual information** from **statistical queries** is **reduced** if **no statistical queries are permitted** whenever **the number of tuples** in the **population specified** by the **selection condition** falls below **some threshold**.
- Another **technique** for prohibiting retrieval of individual information is to **prohibit sequences of querie**s that **refer repeatedly to the same population of tuples**.