CHAPTER 8

Introduction to Database Security

TABLE OF CONTENTS

CHAPTER OBJECTIVES

In this chapter you will learn the following:

- The meaning of *database security*
- How security protects privacy and confidentiality
- Examples of accidental or deliberate threats to security
- Some database security measures
- The meaning of *user authentication*
- The meaning of *authorization*
- How access control can be represented
- How the view functions as a security device

- The purpose of the security log and audit trail
- How and why data encryption is performed
- How to protect databases against SQL injection
- How security is enforced in some systems
- How Internet security is implemented

8.1 Issues in Database Security

Database security involves protecting the database from unauthorized access, modification, or destruction. Since the database represents an essential corporate resource, database security is an important subcomponent of any organization's overall information systems security plan. In addition to the need to preserve and protect data for the smooth functioning of the organization, database designers have a responsibility to protect the privacy of individuals about whom data is kept. **Privacy** is the right of individuals to have some control over information about themselves. Many countries have laws designed to protect privacy, and every organization that collects and stores information about individuals is legally obliged to adopt policies that conform to local privacy legislation. The database design should reflect the organization's commitment to the protection of individual privacy rights by including only those items that the organization has a right to know and keeping them secure.

The security of information typically follows the **CIA** model, where CIA stands for *confidentiality*, *integrity*, and *availability*. **Confidentiality** requires that only authorized users have access to information in order to preserve the privacy of individuals, business intellectual property, and national security efforts. With the growth of social media and online business due to the Internet, maintaining confidentiality involves using appropriate encryption techniques as well as user authorization, identification, and authentication procedures. **Integrity** requires that only authorized users be allowed to modify data, thus maintaining data consistency and trustworthiness. If data is incorrect, it is no longer useful. Incorrect data can also be harmful to individuals (such as wrong data on a credit report) and organizations (such as invalid financial reports). **Availability** requires that information be accessible by authorized users when needed. Security attacks against an organization can cause business services to become unavailable, leading to violations of service level agreements that are critical to business operations.

Some of the laws and standards requiring controls on access, disclosure, and modification of sensitive data are:

- **The Federal Information Security Management Act (FISMA)**. FISMA requires federal agencies in the United States to develop and implement an agency-wide information security plan in support of federal operations.
- **The European General Data Protection Regulation (GDPR)**. The GDPR establishes data protection regulations for all foreign companies that process data of European Union residents.

- **The U.S. Health Insurance Portability and Accountability Act (HIPAA)**. HIPAA defines requirements for health care organizations for maintaining security and privacy of patient data.
- **The U.S. Sarbanes-Oxley (SOX) Act**. SOX defines strict regulations for financial reporting activities of publically traded companies.
- **The U.S. Gramm-Leach-Bliley Act (GLBA)**. GLBA establishes provisions to ensure the protection of consumers' financial information.
- **The Worldwide Payment Card Industry Data Security Standard (PCI DSS)**. PCI DSS defines a framework for secure processing of consumer credit card information.

Violation of these practices and regulations can lead to fraud, financial losses, and severe penalties.

Security threats are events or situations that could harm the system by compromising privacy or confidentiality, or by damaging the database itself. A vulnerability is a weakness in a system, such as inappropriate access control or loopholes in firewall protection, that allows a threat to occur. Security threats can occur either **accidentally** or **deliberately**. Putting a database security plan in place should include a risk assessment process that identifies threats and vulnerabilities and establishes appropriate controls in the context of the CIA model.

8.1.1 Accidental Security Threats

Some examples of accidental security violations are the following:

- The user may unintentionally request an object or an operation for which he or she should not be authorized, and the request could be granted because of an oversight in authorization procedures or because of an error in the database management system or operating system.
- A person may accidentally be sent a message that should be directed to another user, resulting in unauthorized disclosure of database contents.
- A communications system error might connect a user to a session that belongs to another user with different access privileges.
- The operating system might accidentally overwrite files and destroy part of the database, fetch the wrong files, and then inadvertently send them to the user, or it might fail to erase files that should be destroyed.

8.1.2 Deliberate Security Threats

Deliberate security violations occur when a user intentionally gains unauthorized access and/or performs unauthorized operations on the database. A disgruntled employee who is familiar with the organization's computer system poses a tremendous threat to security. Industrial spies seeking information for competitors also threaten security. Privileged users such as DBAs who access end-user data that they should not be permitted to see threaten security. There are many ways deliberate security breaches can be accomplished, including:

- Wiretapping of communication lines to intercept messages to and from the database
- Electronic eavesdropping, to pick up signals from workstations, printers, or other devices within a building
- Reading display screens and reading or copying printouts left unsupervised by authorized users
- Impersonating an authorized user, or a user with greater access, by using his or her log-in and password
- Writing systems programs with illegal code to bypass the database management system and its authorization mechanism, and to access database data directly through the operating system
- Writing applications programs with code that performs unauthorized operations
- Deriving information about hidden data by clever querying of the database
- Modifying database queries through SQL injection to gain unauthorized access to data or to maliciously modify or delete data
- Removing physical storage devices from the computer facility
- Making physical copies of stored files without going through the database management system, thereby bypassing its security mechanisms
- Bribing, blackmailing, or otherwise influencing authorized users in order to use them as agents in obtaining information or damaging the database
- Using system privileges to grant oneself access to confidential user data

8.2 Fundamentals of Access Control

In any organization, access control methods should be defined to restrict access to company resources as well as employee and client data. Access control is a fundamental component in the support of confidentiality and integrity. Access control must be addressed in the context of physical security as well as information system access control. To protect the information system, the database administrator is responsible for the following major tasks:

- Installing the database management system and configuring it securely
- Creating and securing user accounts and developing appropriate access controls for users
- Developing and enforcing standards for applications programs that access the database
- Encrypting sensitive data
- Ensuring that network connections to the data are secure
- Establishing appropriate audit mechanisms for the database
- Protecting the database against intruders by identifying and guarding against security threats and applying security controls and security updates as needed

8.2.1 Physical Security

An access control plan should begin with physical security measures for the building itself, with special precautions for the computer facilities. Designing a physically secure building is clearly outside the domain of the database designer. However, the DBA or data administrator should be able to suggest measures that would control access to database facilities. Often these begin at the front door, where all employees must be identified visually by guards or by using badges, handprints, sign-ins, or other mechanisms. Additional identification should be required to enter the computer facilities. Physical security measures should be extended to cover any location where offline data, such as backups, are stored as well.

8.2.2 Information System Access Control

Development of information system access control is a process that involves *authorization*, *identification*, *authentication*, and *accountability*.

Authorization requires defining who has access to the system and the specific data they are allowed to access. Most database management systems designed for multiple users have their own security subsystems. These subsystems provide for **user authorization**, a method by which users are assigned rights to use database objects. Most multiple-user systems have a **data control language**, also called an **authorization language**, that is part of the data sublanguage. For example, SQL provides standard authorization commands to grant privileges to users, as discussed in Section 8.7. The DBA uses the authorization language to specify users' rights by means of **authorization rules**, statements that specify which users have access to what information, and what operations they are permitted to use on what data. The authorization mechanism is designed to protect the database by preventing individuals from unauthorized reading, updating, or destruction of database contents. These restrictions are added to the security mechanisms provided by the operating system. However, in a surprisingly large number of cases, database security subsystems are minimal or are not fully utilized. Recognizing that data is a valuable corporate resource, the designer should include available security mechanisms as an important factor in evaluating alternative database management systems, and should develop effective security policies utilizing whatever controls are available with the chosen system.

Identification refers to the way in which users are identified. A user ID is a common form of identification. In addition to a computer system user ID, users may also have a specific database ID, which forms the basis for defining access rules using the database authorization sublanguage. In conjunction with physical security, users may have other forms of identity, such as smart cards that are swiped through an electronic card reader to gain access to parking lots, buildings, and rooms that house database facilities as well as other general workspaces. **Biometrics** can provide a more secure form of identification, especially in highly confidential applications. Biometrics can include fingerprints, handprints, face recognition, voice recognition, and retina scans.

Authentication is the process of verifying the identity of a user—checking to ensure that the actual user is who he or she claims to be. Authentication is initially implemented at the operating system level. When the user signs on, he or she enters a user ID, which is checked for validity. The system has a user profile for that ID, giving information about the user. The profile normally includes a password, which should be known only to the user. Passwords should be kept secret and changed frequently. A simple security precaution is for the system to require length and special character requirements for a password and that passwords be changed frequently. The system should never display passwords at log-in, and the stored profiles should be kept secure, in encrypted form. Another security precaution is to lock a user out of an account after several invalid log-in attempts. The lockout policy prevents hackers from a brute-force attempt at guessing a user's password.

Although passwords are the most widely used authentication method, they are not very secure, since users sometimes write them down, choose words that are easy to guess, or share them with others. In some organizations, a multifactor approach to authentication is used, where users must provide two or more forms of authentication. In the multifactor approach, a user might provide a user ID and password as well as a smartcard, badge, token, or some form of biometrics. An authentication procedure might also consist of answering a series of questions that would take longer and be more difficult to reproduce than a single password. Although authentication may be done only at the operating system level, it is desirable to require it again at the database level. At the very least, the user should be required to produce an additional password to access the database.

The final component of information system access control is accountability. **Accountability** refers to the need to capture and maintain log files that can be used for traceability when security incidents occur. For example, operating systems maintain login information about users as well as the directories, files, and databases that they access. Log files are also maintained about network traffic that can be used to trace remote access to a system. Database systems maintain log files as part of the database recovery system, recording user access information as well as the inserts, updates, and deletes that occur. Log files can provide important information about user access to specific data items when conducting forensic activity after a security breach.

8.3 Database Access Control

Database access control is the process of making sure that data or other resources are accessed only in authorized ways. In planning access, the DBA might use an **access control matrix** for the database, as shown in FIGURE 8.1. The column headings represent database objects, which may be the names of tables, views, data items, objects, modules, or other categories, depending on the database model and management system used. The row labels represent individuals, roles, groups of users, or applications. The cell entries specify the type of access permitted. Values of entries will also depend on the particular system used, but the choices usually include `READ`, `INSERT`, `UPDATE`, `DELETE`, `EXECUTE`, `CREATE`, and others. Once the access control matrix is complete, the DBA must use the appropriate authorization language to implement it. The DBA, of course, is permitted to create and change the structure of the database, and to use the authorization language to grant data access to others or to revoke access. Some systems allow the

FIGURE 8.1
Access Control Matrix

OBJECT

SUBJECT	Student table	StuView1	WrapUp Procedure	Faculty table	Enroll table	CREATE TABLE
User U101	READ, UPDATE	READ	EXECUTE	READ		YES
User U102		READ				NO
Advisor Role	READ	READ			READ, INSERT, UPDATE, DELETE	NO
. . .	. . .	. . .	. . .	. . .	. . .	. . .

DBA to delegate some of this power, granting users the power to authorize other users to perform operations on the database. However, having many such "authorizers" can be extremely dangerous. Since authorizers can create other authorizers, the situation can get out of hand very quickly, making it difficult for the DBA to revoke authorizations.

8.4 Using Views for Access Control

The view is a widely used method for implementing access control in database applications. The view mechanism has a twofold purpose. It is a facility for the user, simplifying and customizing the external model through which the user deals with the database, freeing the user from the complexities of the underlying model. It is also a security device, hiding structures and data that the user should not see. In the relational and object-relational models, a user's external model can consist entirely of views, or some combination of base tables and views. By specifying restrictions in the `WHERE` line of the `SELECT` statement used to create views, the view can be made **value-dependent**. FIGURE 8.2(A) gives an example of a view created from the `Student` table by including only data about students whose major is CSC. **Value-independent** views are created by specifying columns of base tables and omitting the `WHERE` line of the `SELECT` statement. FIGURE 8.2(B) gives an example of a view of the `Student` table showing only columns `stuId`, `lastName`, `firstName`, and `major`.

```
CREATE VIEW CSCMAJ AS
        SELECT stuId, lastName, firstName, credits
        FROM Student
        WHERE major = 'CSC';
```

FIGURE 8.2(A)
Value-dependent View

```
CREATE VIEW StuView1 AS
        SELECT stuId, lastName, firstName, major
        FROM Student;
```

FIGURE 8.2(B)
Value-independent View

8.5 Security Logs and Audit Trails

Another important security tool is the **security log**, which is a journal that keeps a record of all attempted security violations. The violation can be simply recorded in the log, or it can trigger an immediate message to the system operator or to the DBA. Knowing about the existence of the log can be a deterrent in itself. If the DBA suspects that data is being compromised without triggering security log entries, it is possible to set up an **audit trail**. Such an auditing system records all access to the database, keeping information about the user who requested the access, the operation performed, the workstation used, the exact time of occurrence, the data item, its old value, and its new value, if any. The audit trail can therefore uncover the sources of suspicious operations on the database, even if they are performed by authorized users, such as disgruntled employees. **Triggers** can also be used to set up an audit trail for a table, recording all changes, the time they were made, and the identity of the user who made them. For example, in Oracle, if we wish to monitor changes to `grade` in the `Enroll` table, we could first set up a table to hold the audit records. The schema for that table might be:

```
EnrollAudit(dateandTimeOfUpdate, userId, oldStuId,
oldClassNo, oldGrade, newGrade)
```

The trigger should insert a record in the `EnrollAudit` table when a user tries to update a grade in the `Enroll` table. The code to do this is shown in FIGURE 8.3. It uses `SYSDATE` and `USER`, which are referred to as

FIGURE 8.3
Audit Trail Using Trigger

```
CREATE OR REPLACE TRIGGER EnrollAuditTrail
    BEFORE UPDATE OF grade ON Enroll
    FOR EACH ROW
    BEGIN
        INSERT INTO EnrollAudit
            VALUES(SYSDATE, USER, :OLD.stuId, :OLD.classNumber, :OLD.grade,
                   :NEW.grade);
END;
```

pseudocolumns in Oracle. Both act as functions that return appropriate values. `SYSDATE` returns the current date and time, while `USER` returns the `ID` of the current user. Oracle itself has built-in auditing that can be used to set up various types of audit trails as well as other security measures.

8.6 Encryption

To counter the possibility of having files accessed directly through the operating system or having files stolen, data can be stored in the database in encrypted form. Only the database management system can unscramble the data, so that anyone who obtains data by any other means will receive jumbled data. When authorized users access the information properly, the DBMS retrieves the data and decodes it automatically. Encryption should also be used whenever data is communicated to other sites, so that wire tappers will also receive scrambled data. Encryption requires a **cipher system**, which consists of the following components:

- An **encrypting algorithm**, which takes the normal text (**plaintext**) as input, performs some operations on it, and produces the encrypted text (**ciphertext**) as output
- An **encryption key**, which is part of the input for the encrypting algorithm and is chosen from a very large set of possible keys
- A **decrypting algorithm**, which operates on the ciphertext as input and produces the plaintext as output
- A **decryption key**, which is part of the input for the decrypting algorithm and is chosen from a very large set of possible keys

8.6.1 Symmetric Key Encryption

Symmetric key encryption is a form of encryption where the decryption key is the same as the encryption key, and the decrypting algorithm is the inverse of the encrypting algorithm. One widely used symmetric key encryption scheme was the **Data Encryption Standard (DES)**, devised by IBM for the U.S. National Bureau of Standards and adopted in 1977. In the DES scheme, the algorithm itself is public, while the key is private. FIGURE 8.4 gives an overview of the DES process. The DES algorithm uses a 56-bit key on 64-bit blocks of plaintext, producing 64-bit blocks of ciphertext. When data is encoded, it is split up into 64-bit blocks. Within each block, characters are substituted and rearranged according to the value of the key. The decoding algorithm uses the same key to put back the original characters and to restore them to their original positions in each block.

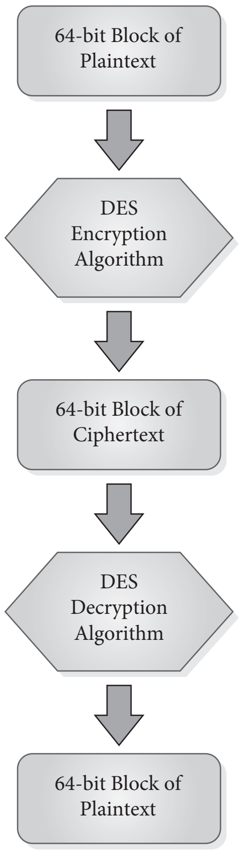

64-bit Block of Plaintext
DES Encryption Algorithm
64-bit Block of Ciphertext
DES Decryption Algorithm
64-bit Block of Plaintext

FIGURE 8.4
Overview of DES Encryption

Two major challenges with the DES system involve key security and the ease of cracking the code. The key must be kept secure or the encryption is worthless, since anyone with the key has access to the data. Therefore, the security depends on the secrecy of the key, but all authorized users must be told the key. The more people who know the key, the more likely it is that the key will be disclosed to unauthorized users. Also, it is necessary to distribute the key to receivers of encrypted messages. If telecommunications lines are used, transmitting the key in plaintext would allow wire tappers easy access to encrypted messages. Often, more secure lines are used for key distribution, or the key is distributed by mail or messenger. DES is not a very secure scheme, since it can be cracked in a reasonable amount of time due to the shortness of the keys. As a result of several famous cases where DES keys were cracked, a more secure version, called **Triple DES** or **3DES**, was recommended in 1999 by the U.S. National Institute of Standards and Technology, the successor to the National Bureau of Standards. Triple DES is now widely used commercially and is still permitted for some government agency use. The triple DES system uses three keys and essentially performs the DES encryption three times, once with each key.

In 2001, an improved encryption scheme called the **Advanced Encryption Standard (AES)** was developed. AES was the result of a five-year worldwide competition, with the winning design coming from two Belgian cryptographers, Daemen and Rijmen, who proposed a scheme they called **Rijndael**. It was adopted as a standard for U.S. government agency use in 2002, and it is widely used commercially. It uses a symmetric scheme that is more sophisticated than the DES scheme, and it supports three possible key sizes of 128 bits, 192 bits, or 256 bits, depending on the level of security needed. The data itself is broken into 128-bit blocks and is subjected to four rounds of transformations, each with several steps whose exact nature is determined by the key. Because of the larger key sizes, cracking the scheme is more challenging.

8.6.2 Public-Key Encryption

An alternative approach to encryption is **public-key encryption**, which is also known as **asymmetric encryption**. Public-key encryption uses two separate keys, where one key is a public key and the other key is a private key. FIGURE 8.5 provides an overview of public-key encryption. For each user, a pair of large prime numbers, (p, q), is chosen as the user's **private key**, and the product of the pair, $p*q$, becomes the user's **public key**. Public keys are shared freely, so that anyone wishing to send a message to a user can find his or her public key easily. The public key is then used as input to an encryption algorithm, which produces the ciphertext for that user. When the

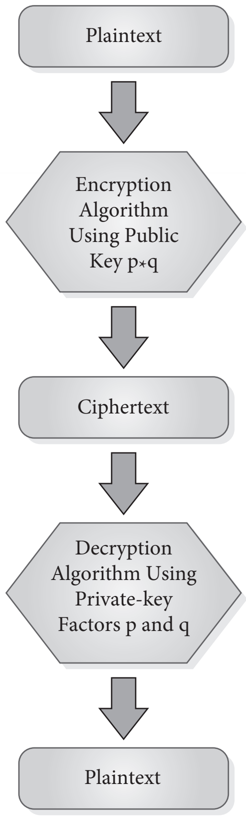

Plaintext
Encryption Algorithm Using Public Key p*q
Ciphertext
Decryption Algorithm Using Private-key Factors p and q
Plaintext

FIGURE 8.5 Overview of Public-Key Encryption

user receives an encrypted message, he or she must produce the prime factors of the public key to decode it. Since there is no quick method of finding the prime factors of a large number, it is difficult for an intruder to find these factors. However, an intruder who is determined to break the key can do so, provided he or she is willing to commit substantial resources to the task. This method is only as secure as the private key, so users must be given their private keys in some secure fashion and must protect the private keys against disclosure. One well-known method of public-key encryption is **RSA**, named for its developers, Rivest, Shamir, and Adleman.

8.7 SQL Data Control Language

SQL has an authorization sublanguage, Data Control Language, that includes statements to grant privileges to and revoke privileges from users.

A **privilege** is an action, such as creating, executing, reading, updating, or deleting, that a user is permitted to perform on database objects. In standard SQL, the creator of a schema is given all privileges on all the objects (tables, views, roles, applications) in it, and can pass those privileges on to others. Ordinarily, only the creator of the schema can modify the schema itself (adding tables, columns, and so on). The statement for granting privileges has the following form:

```
GRANT {ALL PRIVILEGES | privilege-list }
ON {object-name}
TO {PUBLIC |user-list|role-list } [WITH GRANT OPTION];
```

The possible privileges for base tables are `SELECT`, `DELETE`, `INSERT`, `UPDATE`, or `REFERENCES(col-name)`. If a table is named in the `ON` clause, then `ALL PRIVILEGES` includes all of these operations. If a view is named in the `ON` clause, and the view was constructed in such a way that it is updatable, the `SELECT`, `DELETE`, `INSERT`, and `UPDATE` privileges can be granted on that view. For views that are not updatable, only the `SELECT` can be granted. The `UPDATE` privilege can be made more restrictive by specifying a column list in parentheses after the word `UPDATE`, restricting the user to updating only certain columns, as in:

```
GRANT UPDATE ON Student(major) TO U101;
```

The `REFERENCES` privilege is applied to columns that may be used as foreign keys. This privilege allows the user to refer to those columns in creating foreign key integrity constraints. For example, to allow a user who can update the `Enroll` table to be able to reference `stuId` in the `Student` table in order to match its values for the `Enroll` table, we might write:

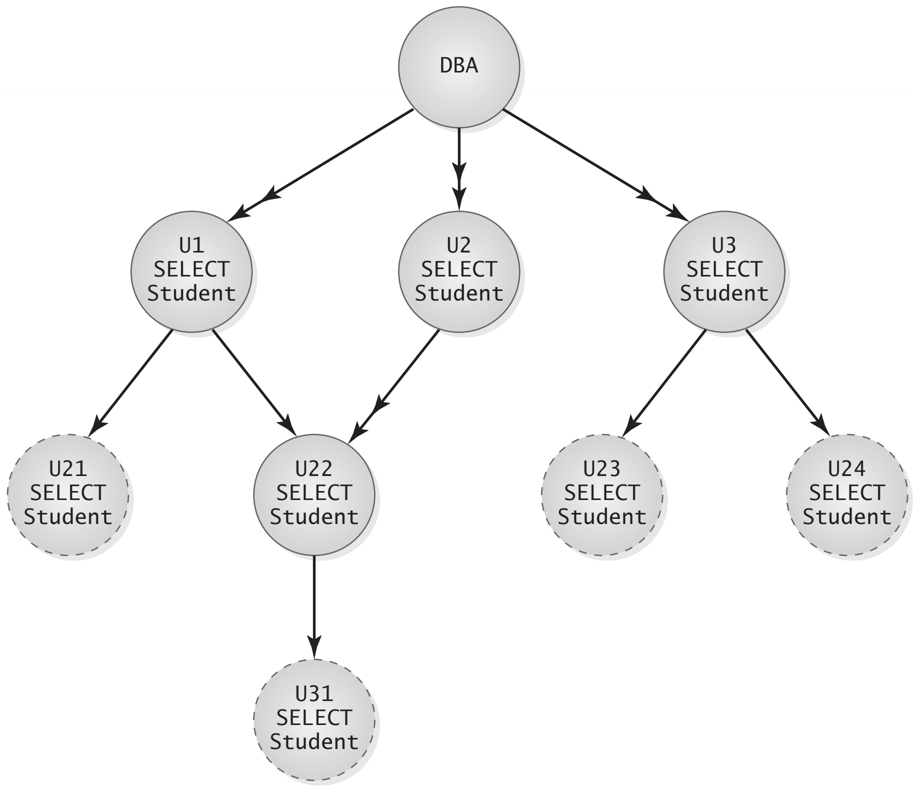
```
GRANT REFERENCES (stuId) ON Student TO U101;
```

The user list in the `TO` clause can include a single user, several users, or all users (the public). The optional `WITH GRANT OPTION` clause gives the newly authorized user(s) permission to pass the same privileges to others. For example, we could write:

```
GRANT SELECT, INSERT, UPDATE ON Student TO U101,
U102, U103 WITH GRANT OPTION;
```

Users `U101`, `U102`, and `U103` would then be permitted to write SQL `SELECT`, `INSERT`, and `UPDATE` statements for the `Student` table, and to pass that permission on to other users. Because of the ability of users with the grant option to authorize other users, the system must keep track of authorizations using a **grant diagram**, also called an **authorization graph**. FIGURE 8.6 shows an authorization graph. Here, the DBA, who (we assume) is the creator

FIGURE 8.6
An Authorization Graph



of the schema, gave a specific privilege (for example, to use SELECT on the `Student` table) `WITH GRANT OPTION` to users `U1`, `U2`, and `U3`. We will use a double arrowhead to mean granting with grant option, and a single arrowhead to mean without it. A solid outline for a node will mean that the node has received the grant option, and a dashed outline will mean it has not. `U1` passed along the privilege to `U21` and `U22`, both without the grant option. `U2` also passed the privilege to `U22`, this time with the grant option, and `U22` passed the privilege to `U31`, without the grant option. `U3` authorized `U23` and `U24`, both without the grant option. Note that if we give a different privilege to one of these users, we will need a new node to represent the new privilege. Each node on the graph represents a combination of a privilege and a user.

SQL DCL includes the capability to create user roles. A role can be thought of as a set of operations that should be performed by an individual or a group of individuals as part of a job. For example, in a university, advisors may need to be able to read student transcripts of selected students, so there may be an `Advisor` role to permit that. Depending on the policies of the university, the `Advisor` role might also include the privilege of inserting enrollment records for students at registration time. Students may be permitted to perform `SELECT` but not `UPDATE` operations on their personal data, so there may be a `Student` role that permits such access. Once the DBA has identified a role, a set of privileges is granted for the role, and then user accounts can be assigned the role. Some user accounts may have several roles.

To create a role, we write a statement such as:

```
CREATE ROLE AdvisorRole;
CREATE ROLE FacultyRole;
```

We then grant privileges to the role just as we would to individuals, by writing statements such as:

```
GRANT SELECT ON Student TO AdvisorRole;
GRANT SELECT, UPDATE ON Enroll TO AdvisorRole;
GRANT SELECT ON Enroll TO FacultyRole;
```

To assign a role to a user, we write a statement such as:

```
GRANT AdvisorRole TO U999;
```

We can even assign a role to another role by writing, for example:

```
GRANT FacultyRole TO AdvisorRole;
```

This provides a means of inheriting privileges through roles.

The SQL DCL statement to remove privileges has this form:

```
REVOKE {ALL PRIVILEGES | privilege-list }
ON object-list
FROM {PUBLIC | user-list | role-list };
[CASCADE | RESTRICT];
```

For example, for `U101`, to whom we previously granted `SELECT`, `INSERT`, and `UPDATE` on `Student` with the grant option, we could remove some privileges by writing this:

```
REVOKE INSERT ON Student FROM U101;
```

This revokes `U101`'s ability both to insert `Student` records and to authorize others to insert `Student` records. We can revoke just the grant option, without revoking the insert, by writing this:

```
REVOKE GRANT OPTION FOR INSERT ON Student FROM U101;
```

If an individual has the grant option for a certain privilege and the privilege or the grant option on it is later revoked, all users who have received the privilege from that individual have their privilege revoked as well. In this way, revocations **cascade**, or trigger other revocations. If a user obtained the same privilege from two authorizers, one of whom has authorization revoked, the user still retains the privilege from the other authorizer. Thus, if the DBA revoked the authorization of user `U1` in Figure 8.6, `U21` would lose all privileges, but `U22` would retain whatever privileges were received from `U2`. Since `U22` has the grant option, user `U21` could regain privileges from `U22`. In this way, unscrupulous users could conspire to retain privileges despite attempts by the DBA to revoke them. For this reason, the DBA should be very careful about passing the grant option to others. If the `RESTRICT` option is specified, the system checks to see if there are any cascading revocations and returns an error if they exist, without executing the revoke statement. `CASCADE` is the default. When a privilege is revoked, the authorization graph is modified by removing the node(s) that lose their privileges.

8.8 Security in Oracle

Oracle provides robust security that goes far beyond the SQL authorization language commands. There are many different ways to set up and manage the security of an Oracle database installation besides the methods discussed here.

8.8.1 Security Features

Security features include facilities for all the following activities:

- **Management of user accounts**. User accounts can be created, user rights defined, and password and profile policies set up in several ways. Strong passwords can be enforced. User views, user privileges, and roles can be used to limit user access to data.
- **Authentication of users** can be performed for the database from the operating system level and from a network.
- **Application security** policies can be set for all applications that access the database.
- **Privilege analysis** allows the DBA to identify privileges that are being used, track the source of the privileges, and identify privileges that are not being used. This information can be used to tighten security.
- **User session information for applications**. Information such as the user name and location can be gathered automatically and used to control the user's access through an application.

› **Virtual Private Database (VPD)** is an additional level of security that can be used to control access on the row and column level.

› **Data redaction** is a method of masking data at run time, when queries are executed. Some or all of the characters are hidden or replaced in the results set. For example, only the last four digits of a Social Security number or a credit card number may be displayed. Redaction is often done to comply with regulations such as PCI DSS or SOX.

› **Transparent sensitive data protection** can be used as a method of identifying and protecting all columns that hold sensitive data, even across several databases. Once identified, the columns may be protected using VPD or data redaction.

› **Network data encryption** can be performed automatically or manually using the DBMS_CRYPTO PL/SQL package. Oracle Net Services can be configured to provide data encryption and integrity on servers and clients. Thin Java Database Connectivity (JDBC) clients can be configured for secure connections to databases.

› **Strong authentication.** Available industry-standard authentication methods include centralized authentication and single sign-on, Secure Sockets Layer (SSL), Remote Authentication Dial-In User Service (RADIUS), and Kerberos.

8.8.2 Administrative Accounts

On installation, Oracle provides several predefined administrative super accounts, including `SYS`, `SYSTEM`, and `DBSNMP`, as well as some sample schemas. `DBSNMP` is used for administration tasks in Oracle Enterprise Manager, and the management agent can manage and monitor the database using the `DBSNMP` account. The `SYS` account stores data dictionary information for base tables and views, and should be used only by the DBMS itself, not by users. The `SYSTEM` account stores other tables and tools used by Oracle and tables for administration. None of these accounts should be used to create user tables, and access to them should be strictly controlled. On installation, these three administrator accounts are open, and the system prompts for passwords for all three of them, although default passwords are provided. Since these passwords are widely known, it is strongly recommended that new passwords be created to protect the database from attack. The accounts automatically have the DBA role, which permits a user to create roles and users; to grant privileges to other users; and to create, modify, and delete schemas and objects. Oracle suggests that administrative tasks are best performed using more targeted accounts that are authorized for specific jobs, a concept called **separation of duties**. To this end, there are six additional administrator accounts that should be opened and assigned to administrators to be used for specific tasks. They are `SYSDBA`, `SYSOPER`, `SYSASM`, `SYSBACKUP`, `SYSDG`, and `SYSKM`.

Privileges granted to database users can be *object privileges* or *system privileges*. An **object privilege** is the right to perform an action using DML commands on a table, view, procedure, function, sequence, or package. The creator of a schema automatically has all object privileges on all objects in the schema and can grant the same object privileges to other users. For tables, the privileges include `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `REFERENCES`, as described in Section 8.7, but also `ALTER` (the right to use the `ALTER TABLE` command) and `INDEX` (the right to use the `CREATE INDEX` command). For updatable views, privileges are `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. **System privileges** include the right to perform actions using DDL commands on database data, schemas, tablespaces, or other Oracle resources, as well as the right to create user accounts.

8.8.3 Security Tools

Some of the tools the DBA can use to secure the database installation are Oracle Database Configuration Assistant, Oracle Enterprise Manager, SQL*Plus, and Oracle Net Manager. Oracle Enterprise Manager is an online tool found on the Oracle Database Home page. On a Windows installation, SQL*Plus can be found in the `Application Tools` subdirectory of the `Ora` home directory, and the two other tools within the `Configuration and Migration Tools` subdirectory.

- **Oracle Database Configuration Assistant** has options to create, configure, or delete databases and other operations, including setting an audit policy.
- **Oracle Enterprise Manager** is a Web-based facility that offers options for granting and revoking privileges. The DBA has to log in initially using a privileged account such as `SYSTEM` to the Oracle Database home page to access the Enterprise Manager. To create user accounts from there, the DBA can choose the `Administration` icon, then `Users`, then `Create`. The DBA fills in the new user name and password, enters a temporary password, and can choose to have the password expire immediately. This will cause the new user to be prompted for a new password the first time he or she uses the account. The account status should be set to `Unlocked`. The user role and privileges should be chosen from the list provided, and the `CREATE` button should be clicked to finish the operation. It is recommended that the first account created be one that will be used

by a security administrator, and that this account be given all rights related to security, to separate those tasks from other administrative responsibilities. That account should then be used for managing security.

- **SQL*Plus** can also be used to create users and roles. After signing in to a privileged account such as `SYSTEM`, or having been authorized to create users, the DBA writes a `CREATE USER` command, which has this form:

```
CREATE USER username IDENTIFIED BY password;
```

For example:

```
CREATE USER U999 IDENTIFIED BY SESAME;
```

However, this command does not give any privileges to the user, so `U999` will not be able to establish a session unless the DBA also writes the following:

```
GRANT CREATE SESSION TO U999;
```

To require the user to change his or her password at the first actual log-in, the DBA uses this command:

```
ALTER USER username

PASSWORD EXPIRE;
```

When the user tries to connect, he or she will be given a message saying the password has expired and prompting for a new one before being connected. Once connected, the user can also change his or her own password at any time by writing the following in SQL*Plus:

```
ALTER USER username

IDENTIFIED BY newpassword;
```

Although the user will be connected, he or she will not be able to access any data, since the only privilege given is the one to create a session. To actually use Oracle's facilities, the user needs to be given additional privileges, which can be either object privileges or system privileges as described earlier.

The syntax for granting object privileges is the same as the standard SQL DCL syntax shown in Section 8.7. For example, the DBA might give `U999` wide privileges on the `Student` table by writing as follows:

```
GRANT ALL PRIVILEGES ON Student TO U999 WITH GRANT OPTION;
```

If there is a stored procedure called `WrapUp`, the DBA can give `U999` permission to run the procedure by writing this command:

```
GRANT EXECUTE ON WrapUp TO U999;
```

There are 236 different system privileges possible. A list of system privileges can be seen by writing the following SQL command:

```
SELECT name
FROM SYSTEM_PRIVILEGE_MAP;
```

System privileges can be given through SQL*Plus using a `GRANT` command of this form:

```
GRANT systemprivilege
TO username
[WITH ADMIN OPTION];
```

For example, we could allow `U999` to create tables by writing:

```
GRANT CREATE TABLE TO U999 WITH ADMIN OPTION;
```

Additionally, privileges that are object privileges on single tables can be extended to become system privileges that extend to any table by using the keyword `ANY`, as in:

```
GRANT SELECT ANY TABLE TO U999;
```

The `WITH ADMIN OPTION` clause allows the user to pass the privilege on to others.

As in the SQL standard, Oracle allows privileges to be given to a role as well as to individuals or groups of users. A role consists of a group of privileges. Any number of roles can be granted to a user. Roles can also be granted to other roles, allowing inheritance of privileges. Roles can be created in SQL*Plus using the DCL commands discussed in Section 8.7.

› **Oracle Net Manager**. During installation, Oracle creates an initial network configuration, including a default listener. Changes can be made to the configuration by using the Net Configuration Assistant, which is found in the `Configuration and Migration Tools` subdirectory of the `Ora` home directory. After configuration, the `Oracle Net Manager` in the same subdirectory can be used to manage the networks. The DBA can set profiles, choose encryption for the server and client, provide an encryption seed, and choose one or more of several encryption methods.

8.9 Statistical Database Security

Statistical databases are designed to provide data to support statistical analysis on populations. The data itself may contain facts about individuals, but the data is not meant to be retrieved on an individual basis. Users are granted permission to access statistical information such as totals, counts, or averages, but not information about individuals. For example, if a user is permitted statistical access to an employee database, he or she is able to write queries such as:

```
SELECT SUM(Salary)
FROM Employee
WHERE Dept = 10;
```

but not:

```
SELECT Salary
FROM Employee
WHERE empId = 'E101';
```

Special precautions must be taken when users are permitted access to statistical data, to ensure that they are not able to deduce data about individuals. For the preceding example, if there are no restrictions in place except that all queries must involve `COUNT`, `SUM`, or `AVERAGE`, a user who wishes to find the employee of `E101` can do so by adding conditions to the `WHERE` line to narrow the population down to that one individual, as in:

```
SELECT SUM(Salary)
FROM Employee
WHERE Dept = 10 AND jobTitle = 'Programmer' AND
dateHired > '01-Jan-2015';
```

The system can be modified to refuse to answer any query for which only one record satisfies the predicate. However, this restriction is easily overcome, since the user can ask for total salaries for the department and then ask for the total salary without that of `E101`. Neither of these queries is limited to one record, but the user can easily deduce the salary of employee `E101` from them. To prevent users from deducing information about individuals, the system can restrict queries by requiring that the number of records satisfying the predicate must be above some threshold and that the number of records satisfying a pair of queries simultaneously cannot exceed some limit. It can also disallow sets of queries that repeatedly involve the same records.

8.10 SQL Injection

Database applications must take security precautions to protect a database against a form of attack known as **SQL injection**. The term *injection* refers to the fact that user input from a client through the application interface can be designed to take advantage of vulnerabilities associated with the dynamic construction of SQL queries. Using SQL injection, an attacker can insert (or *inject*) code into a query that can be used to retrieve information that the attacker is not authorized to see, maliciously delete or modify data, or insert data that would give an attacker unauthorized access to the database. SQL injection was first discovered around 1998 and is now ranked as a top software security concern by the **Open Web Application Security Project** and by the **Common Weakness Enumeration/SANS Top 25 Most Dangerous Software Errors**. SQL injection poses threats to confidentiality, integrity, availability, authentication, and authorization.

8.10.1 Examples of SQL Injection

SQL injection takes advantage of the fact that SQL queries can be dynamically constructed in application code. As an example, consider a Web form that allows a student to enter his or her identifier and password into the variables `userID` and `password`. The website then uses this information to retrieve the student's confidential information from the `Student` table. Inside of the application code, the query can be constructed dynamically through the following statement:

```
studentInfoQuery = "SELECT * FROM student WHERE
userID = '" + userID + "' AND password = '" +
password + "';"
```

The + character represents the string concatenation operator. If `userID` contains the value `John` and `password` contains the value `x1y2z3`, then `studentInfoQuery` would contain the following `SELECT` statement:

```
SELECT *
FROM student
WHERE userID = 'John' AND password = 'x1y2z3';
```

The `studentInfoQuery` would then be submitted to the database for retrieval of the information. This query only works as intended if the input value for `userID` and/or `password` does not contain a single quote character. For example, if the user enters `x1y'z3` by mistake, the query becomes:

```
SELECT *
FROM student
WHERE userID = 'John' AND password = 'x1y'z3';
```

The extraneous single quote in the password will cause SQL to generate an error message since `'x1y'z3';` is invalid syntax in the SQL parser. An attacker will sometimes use this technique to initially discover that a database is vulnerable to an SQL injection attack, where the syntax error message gives an attacker a clue that the query is being dynamically constructed without any input validation. A subsequent malicious query can then be constructed that will give the attacker access to John's information as well as the information of all other students.

As an example, suppose an attacker does not know any valid user IDs or passwords and that the attacker enters the value `X` as the `userID` and `Y' or 'a'='a` as the password. The query then becomes:

```
SELECT *
FROM student
WHERE userID = 'X' AND password = 'Y' OR 'a'='a';
```

Even with an incorrect `userID` and `password`, this query will always evaluate to true because the `or` condition will always be satisfied. Furthermore, the query will essentially evaluate as the query below, which returns information about all students:

```
SELECT *
FROM student
WHERE 'a'='a';
```

Since some database products allow the execution of multiple SQL statements separated by semicolons within a single query string, attackers can also take advantage of this vulnerability together with single quotes in input values to enter additional malicious statements. To illustrate this type of attack, assume the attacker enters the value `X` as the `userID` and the value `Y' or 'a'='a'; DELETE * FROM student; --` as the password. In this case, the query becomes:

```
SELECT *
FROM student
WHERE userID = 'X' AND password = 'Y' OR 'a'='a';
DELETE * FROM student;
--';
```

The query will retrieve all student information and then delete all of the information in the `Student` table. Notice that the "`--`" comment characters are used at the end of the password string so that any extraneous characters will be commented out of the query execution to avoid a syntax error.

8.10.2 Mitigation of SQL Injection

As illustrated in the previous subsection, SQL injection can be used to cause serious harm to a database application. In addition to the examples described earlier that violate confidentiality and integrity, other SQL statements can be injected into a query that give an attacker access to the database, such as creating a new unauthorized user ID and password with special security privileges. In some cases, it is even possible to invoke certain operating system commands from an SQL query.

Fortunately, there are several actions that can be taken to mitigate SQL injection attacks. The most basic vulnerability lies in the dynamic construction of the SQL query as a string. This approach to building a query in application code is not considered a safe programming practice and should be avoided. An alternative approach is to use parameterized statements, as in prepared statements of the JDBC API. Using parameterized queries forces the values of variables that are used to construct a query to conform to a specific type value instead of an arbitrary string that can contain malicious SQL statements. JBDC and parameterized queries are covered in more detail in Chapter 5. The use of stored procedures as described in Chapter 5 can also be used to avoid SQL injection attacks as long as SQL queries are not dynamically constructed in the stored procedure.

Another mitigation technique is to always validate user input to make sure the input conforms to valid types and patterns before the input is used to construct an SQL query. Database permissions should also be limited to a *need to know* basis to protect against SQL injection attacks. Some database products, such as Oracle, help to mitigate SQL injection by not allowing query strings that contain multiple SQL statements separated by semicolons.

8.11 Database Security and the Internet

Unless security software is used, all messages sent over the Internet are transmitted in plaintext and can be detected by intruders using *packet sniffing* software. Both senders and receivers need to be confident that their communications are kept private. Obviously, customers who wish to purchase products need to have assurance that their credit card information is secure when they send it over the Internet. Companies that allow Web connections to their internal networks for access to their database need to be able to protect it from attack. Receivers of messages need to have ways to be sure that those messages are genuine and trustworthy and have not been tampered with. Senders of messages should not be able to repudiate them, denying that they sent them. Web users who download executable content such as Java applets, ActiveX, or VBScript need to have ways to assure that the code will not corrupt their databases or otherwise harm their systems. Several techniques are used to address these issues.

8.11.1 Proxy Servers

A **proxy server** is a computer or program that acts as an intermediary between a client and another server, handling messages in both directions. When the client requests a service such as a connection or Web page, the proxy evaluates it and determines whether it can fulfill the request itself. If not, it filters the request, perhaps altering it, and requests the service from the server or other resource. It may cache (store a copy of) the server's response so that a subsequent request can be fulfilled from the stored content without using the server again. The proxy server can be used for several purposes, including to maintain security by hiding the actual IP address of the server, to improve performance by caching, to prevent access to sites that an organization wishes to block from its members, to protect the server from malware, and to protect data by scanning outbound messages for data leaks.

8.11.2 Firewalls

A **firewall** is a hardware and/or software barrier that is used to protect an organization's internal network (intranet) from unauthorized access. Various techniques are used to ensure that messages entering or leaving the intranet comply with the organization's standards. For example, a proxy server can be used to hide the actual network address. Another technique is a **packet filter**, which examines each packet of information before it enters or leaves the intranet, making sure it complies with a set of rules. Various gateway techniques can apply security mechanisms to applications or connections.

8.11.3 Digital Signatures

Digital signatures use a double form of public-key encryption to create secure two-way communications that cannot be repudiated. A digital signature allows a user to verify the authenticity of the person they are communicating with, and provides a means to prove that a message must have come from that person and that it has not been tampered with in transmission. One method of using digital signatures is for the sender to encode a message first with his or her own private key, and then with the public key of the receiver. The receiver decrypts the message first using his or her private key, and then uses the sender's public key. The double encryption ensures that both parties are authentic, since neither one could have encoded or decoded the message without his or her private key. It also ensures that the message is intact, since tampering would invalidate the signature, making it impossible to decode the message.

8.11.4 Certification Authorities

Customers who wish to purchase goods from an e-commerce website need to feel confident that the site they are communicating with is genuine and that their ordering information is transmitted privately. A widely used method of verifying that a site is genuine is by means of **certification authorities** such as Verisign. The process uses public-key encryption. The site begins the certification process by generating a public key and a private key and sending a request to Verisign, along with the site's public key. Verisign issues an encrypted certificate to the site, which stores it for future use. When a customer wishes to place an order using a secure connection to the site, his or her browser asks the site for its Verisign certificate, which it receives in encrypted form. The browser decrypts the certificate using Verisign's public key, and verifies that this is indeed a Verisign certificate and that the site's URL is the correct one. The certificate also contains the site's public key. The browser creates a session key—which it encrypts using the site's public key from the certificate—and sends the session key to the site. Since the session key is encrypted with the site's public key, only the actual site can decrypt it using its private key. Since both the browser and the site are the sole holders of the session key, they can now exchange messages encrypted with the session key, using a simpler protocol such as 3DES or AES. The process described here is the one used in the **Secure Sockets Layer (SSL)** protocol and is typically used for messages to and from a customer during an order process. An additional measure of security is usually used for transmission of credit card numbers. While the user's browser sends the seller site most of the order information encoded with its public key, when the customer is ready to transmit credit card information at the end of the order process, that information, along with the amount to be charged, is sent directly to the card company site for authorization and approval.

The **SET (Secure Electronic Transactions)** protocol, which was used for this process, has been superseded by newer protocols such as Visa's Verified by Visa, which provides both authentication and approval of the purchase. It uses an XML-based protocol called 3-D Secure.

Kerberos is an authentication protocol for networks that allows mutual authentication, in which both client and server can verify identity. A trusted Kerberos server is used as a certification authority. It has a key distribution center that stores the secret keys of each client and server on the network, and it uses these as input to generate time-stamped tickets when the client requests service. A ticket is then used to demonstrate to the server that the client is approved for service. Messages can be encrypted using either symmetric key or public-key protocols. Both the protocol and the free software implementing it were developed at the Massachusetts Institute of Technology. It is used by both Oracle and Caché, as well as many other vendors.

8.12 Chapter Summary

Database security means protecting the database from unauthorized access, modification, or destruction. **Privacy** is the right of individuals to have some control over information about themselves, and is protected by law in many countries. **Confidentiality** refers to the need to keep certain information from being known. Both privacy and confidentiality can be protected by database security. Security violations can be accidental or deliberate, and security breaches can be accomplished in a variety of ways. A security control plan should begin with physical security measures for the building and especially for the computer facilities. Security control of workstations involves user **authentication**, verifying the identity of users. The operating system normally has some means of establishing a user's identity, using user profiles, user IDs, passwords, authentication procedures, badges, keys, or physical characteristics of the user. Additional authentication can be required to access the database.

Most database management systems designed for multiple users have a security subsystem. These subsystems provide for **authorization**, by which users are assigned rights to use database objects. Most have an **authorization language** that allows the DBA to write **authorization rules** specifying which users have what type of access to database objects. **Access control** covers the mechanisms for implementing authorizations. An **access control matrix** can be used to identify what types of operations different users are permitted to perform on various database objects. The DBA can sometimes delegate authorization powers to others.

Views can be used as a simple method for implementing access control. A **security log** is a journal for storing records of attempted security violations. An **audit trail** records all access to the database, keeping information about the requester, the operation performed, the workstation used, and the time, data items, and values involved. **Triggers** can be used to set up an audit trail. **Encryption** uses a **cipher system** that consists of an **encrypting algorithm** that converts **plaintext** into **ciphertext**, an **encryption key**, a **decrypting algorithm** that reproduces plaintext from ciphertext, and a **decryption key**. Widely used schemes for encryption are the **Triple Data Encryption Standard (3DES)**, the **Advanced Encryption Standard (AES)**, and **public-key encryption**. DES/AES uses a standard algorithm, which is often hardware implemented. Public-key encryption uses a product of primes as a public key and the prime factors of the product as a private key.

SQL has a **Data Control Language**, an **authorization language** to provide security. The GRANT statement is used for authorization, and the

REVOKE statement is used to retract authorization. Privileges can be given to individuals or to a role, and then the role is given to individuals.

In Oracle, there are many ways to secure the database and assign privileges. Initially, super administrative accounts are open, and they can be used to create other accounts, roles, and users. Privileges include **object privileges** and **system privileges**. They can be granted using the DCL language in SQL*Plus or through the Oracle Enterprise Manager. Sensitive data can be identified with transparent sensitive data protection and secured with Virtual Private Database or data redaction. Network data can also be secured with Network Data Manager.

Statistical databases must take special precautions to make sure that queries are not used to deduce confidential information. **SQL injection** also poses a significant threat to database applications by taking advantage of vulnerabilities associated with the dynamic construction of queries with user input that has not been validated. Database developers can avoid SQL injection attacks by following more secure software development techniques for the dynamic construction of queries.

When the database is accessible through the Internet, special security techniques are needed. These include **firewalls**, **certification authorities** such as Verisign that issue **digital certificates** using **SSL** or **S-HTTP**, **Kerberos** or similar protocols for user authentication, stronger protocols for financial information, and **digital signatures**.

Exercises

8.1 For each of the following, write SQL statements to create views where needed and to grant the indicated privileges for the University database with this schema:

```
Student(stuId, lastName, firstName, major, credits)
Faculty(facId, name, department, rank)
Class(classNumber, facId, schedule, room)
Enroll(stuId, classNumber, grade)
```

a. Give permission to read the tables `Student` and `Class` to user 201.

b. Create a view of `Enroll` that does not include the `grade` attribute, and give user `201` permission to read and update the view.

c. Create a role that includes reading `Student`, `Class`, and the view created in (b). Give that role to all clerks in the dean's office, which includes users `202`, `203`, `204`, and `205`.
d. Give permission to user `206`, an assistant dean, to read and modify (insert, delete, update) the `Faculty` and `Class` tables. This user can authorize others to read and modify `Class` but not `Faculty`.
e. User `206` authorizes user `300` to read `Class`. Write the command to do this.
f. Create an authorization graph showing all the privileges given so far. You will need a separate node for each combination of privilege and user.
g. Revoke the authorization privilege that the assistant dean was given in (d), but keep his or her own reading and modification privileges. How would you show this change on the authorization graph?
h. Give permission to the Registrar, user `500`, to read and modify `Student`, `Class`, and `Enroll`, and to grant those rights to others.
i. For all academic advisors, give permission to read all `Class` records. For the advisor in the Math department, give permission to read the `Student` records of students majoring in Math and to modify `Enroll` records for these students.

8.2 Assume you have a statistical database with the following schema. The only legal queries are those involving `COUNT`, `SUM`, and `AVERAGE`.

```
newFaculty(facId, lastName, firstName, department,
salary, rank, dateHired)
```

a. Write a legal SQL query to find the salary of the only faculty member who is an instructor in the Art department.
b. Assume the system will refuse to answer queries for which only one record satisfies the predicate as in (a). Write a legal set of queries that allows the user to deduce the salary of the Art instructor.
c. Assume that there are 10 faculty members in the Art department. The system refuses to answer queries where the number of records satisfying the query is less than six. It will also refuse to answer pairs of queries where the number of records satisfying them simultaneously exceeds three. Would these restrictions make your query for (a) or (b) illegal? If so, is there another legal

set of queries that will allow you to deduce the salary of the Art instructor?

8.3 a. Using the University schema shown in Exercise 8.1, write an SQL statement to create a value-dependent view of `Student` that includes only seniors.

b. Write an SQL statement for a value-independent view of `Faculty`. Do not include the whole table.

c. Write a statement to authorize user `125` to read both views.

8.4 Write a trigger to create an audit trail that will track all updates to the salary field of the `newFaculty` table shown in Exercise 8.2.

8.5 Log on to an e-commerce website, such as that of a large bookseller. Locate and read the information provided about security of online transactions. Determine whether SSL or some other secure protocol is used. If possible, display and print the information about the Verisign certificate for the site. You may find this in the options in your browser.

8.6 Examine the security features in Microsoft Access by reading the online Help on the topic. Then do the following:

a. Print the list of trusted publishers, locations, and documents for your computer.

b. Open an Access database you created and encrypt the database with a password.

c. Sign your database and package it for distribution.

8.7 SQL injection is often used to exploit a database. It refers to the process of using SQL injection to read sensitive data, modify a database, or execute administrative operations on a database. In preparation for an SQL injection exploitation, attackers often use SQL injection to discover information about the database. Investigate how SQL injection can be used to discover information about a database. In particular, how can SQL injection be used to discover information such as field names, table names, or even email addresses?

8.8 Building on Exercise 8.7, assume you have discovered that the `Student` table contains the fields `studentID`, `email`, `userID`, `password`, `firstName`, and `lastName`. Using the `SELECT` statement from Section 8.10.1, which selects students based on their `userID` and `password`:

a. Show how SQL injection can be used to insert a new student into the `Student` table. What values have to be input for the `userID` and password to get the `INSERT` statement to execute?

b. What are some reasons why the SQL injection attack to insert a new user might fail?
c. If you know a student's email address, show how SQL injection can be used to change the email address to your email address.
d. Assuming you are successful at changing the student's email address, how can you then get access to the student's password? HINT: On most Web pages, how do you get your own password when you don't remember it?