CHAPTER

12 Database Security

Most modern organizations rely heavily on the information stored in their database systems. From sales transactions to human resources records, mission-critical, sensitive data is tracked within these systems. From the standpoint of security, it is very important that business and systems administrators take the proper precautions to ensure that these systems and applications are as secure as possible. You wouldn't want a junior-level database administrator to be able to access information that only the executive team should see; but you also wouldn't want to prevent your staff from doing their jobs. As with all security implementations, the key is to find a balance between security and usability.

All of the security-related best practices that have been laid out throughout this book apply to securing databases. These include network-level security, physical security, and using server-related best practices. However, there are additional considerations that should be taken into account when securing databases. In this chapter, we'll look at some of these special concepts and techniques. Specifically, we'll begin by taking a look at some general information about what makes databases special. Then, we'll look at the various levels of permissions that must be implemented and managed before a database can be considered secure. We'll also look at database auditing.

NOTE The focus of this chapter is on general database security best practices. Most of this information will apply to all modern databases. Particular terminology, tools, and techniques, however, do vary between products. Be sure to consult documentation for your database platforms to discover any special considerations that might apply to your installations.

General Database Security Concepts

Modern databases must meet different goals. They must be reliable, provide for quick access to information, and provide advanced features for data storage and analysis. Furthermore, they must be flexible enough to adapt to many different scenarios and types of usage. Many organizations rely on databases to serve as the "back end" for purchased applications or

custom-developed applications. The "front end" of these systems are generally client applications or web user interfaces.

Architecturally, relational databases function in a client-server manner (although they can certainly be used as part of multitier applications). That is, a client computer, application, or user can only communicate directly with the database services that are running. They cannot directly access the database files, as can be done with "desktop" database systems, such as Microsoft Access. This is an important point, since it allows security configuration and management to occur at the database level, instead of leaving that responsibility to users and applications.

Databases can be used in various capacities, including:

- **Application support** Ranging from simple employee lists to enterprise-level tracking software, relational databases are the most commonly used method for storing data. Through the use of modern databases, users and developers can rely on security, scalability, and recoverability features.
- **Secure storage of sensitive information** Relational databases offer one of the most secure methods of centrally storing important data. As we'll see throughout this chapter, there are many ways in which access to data can be defined and enforced. These methods can be used to meet legislative requirements in regulated industries (for example, the HIPAA standard for storing and transferring healthcare-related information) and generally for storing important data.
- **Online transaction processing (OLTP)** OLTP services are often the most common functions of databases in many organizations. These systems are responsible for receiving and storing information that is accessed by client applications and other servers. OLTP databases are characterized by having a high level of data modification (inserting, updating, and deleting rows). Therefore, they are optimized to support dynamically changing data. Generally, they store large volumes of information that can balloon very quickly if not managed properly.
- **Data warehousing** Many organizations go to great lengths to collect and store as much information as possible. But what good is this information if it can't easily be analyzed? The primary business reason for storing many types of information is to use this data eventually to help make business decisions. Although reports can be generated against OLTP databases, there are several potential problems: Reports might take a long time to run, and thus tax system resources. If reports are run against a production OLTP server, overall system performance can be significantly decreased. OLTP servers are not optimized for the types of queries used in reporting, thus making the problem worse. Reporting requirements are very different. In reporting systems, the main type of activity is data analysis. OLTP systems get bogged down when the amount of data in the databases gets very large. Therefore, production OLTP data must be often archived to other media or stored in another data repository. Relational database platforms can serve as a repository for information collected from many different data sources within an organization. This database can then be used for centralized reporting and by "decision support" systems.

Because of the heavy reliance that modern organizations place on their data storage systems, it's very important to understand, implement, and manage database security. Throughout this chapter, we'll look at various methods for doing just that. Let's start by looking at an overview of various layers of database security, and how they interact.

Understanding Database Security Layers

Since relational databases can support a wide array of different types of applications and usage patents, they generally utilize security at multiple layers. Each layer of security is designed for a specific purpose and can be used to provide authorization rules. In order to get access to your most trusted information, users must have appropriate permissions at one or more of these layers. As a database or systems administrator, your job is to ensure that the hurdles are of the proper height—that is, your security model takes into account both security and usability. In this section, we'll take an in-depth look at each level of permissions and how they interact. Let's start at the level of the server.

Server-Level Security

A database application is only as secure as the server it is running on. Therefore, it's important to start considering security settings at the level of the physical server or servers on which your databases will be hosted. In smaller, simple configurations, you might need to secure only a single machine. Larger organizations will likely have to make accommodations for many servers. These servers may be geographically distributed and even arranged in complex clustered configurations.

One of the first steps you should take in order to secure a server is to determine which users and applications should have access to it. Modern database platforms are generally accessible over a network, and most database administration tasks can be performed remotely. Therefore, other than for purposes of physically maintaining database hardware, there's little need for anyone to have direct physical access to a database. It's also very important to physically protect databases in order to prevent unauthorized users from accessing database files and data backups. If an unauthorized user can get physical access to your servers, it's much more difficult to protect against further breaches.

Network-Level Security

As mentioned previously, databases work with their respective operating system platforms to serve users with the data they need. Therefore, general operating system and network-level security also applies to databases. If the underlying platform is not secure, this can create significant vulnerabilities for the database. Since they are designed as network applications, you must take reasonable steps to ensure that only specific clients can access these machines.

Some standard "best practices" for securing databases include limiting the networks and/or network addresses that have direct access to the computer. For example, you might implement routing rules and packet filtering to ensure that only specific users on your internal network will even be able to communicate with a server.

As an example, Microsoft's SQL Server database platform uses a default TCP port of 1433 for communications between clients and the database. If you know for certain that there is no need for users on certain subnets of your network to be able to access this server directly, it would be advisable to block network access to this TCP port. Doing so can also prevent

malicious users and code (such as viruses) from attacking this machine over the network. Another security practice involves changing the default port on which the server listens. This can be done quite simply by using the Server Network Utility shown in Figure 12-1.

Of course, few real-world databases work alone. Generally, these systems are accessed directly by users, and often by mission-critical applications. Later in this chapter, we'll look at some methods for mitigating risks related to Internet-accessible applications.

Data Encryption

Another method for ensuring the safety of database information is to use encryption. Most modern databases support encrypted connections between the client and the server. Although these protocols can sometimes add significant processing and data transfer overhead (especially for large result sets or very busy servers), the added security may be required in some situations. Additionally, through the use of virtual private networks (VPNs), systems administrators can ensure that sensitive data remains protected during transit. Depending on the implementation, VPN solutions can provide the added benefit of allowing network administrators to implement security without requiring client or server reconfiguration.

Data encryption is also an important security feature in areas outside of the network layer. Often, database administrators will make backups of their data and store them on file servers. These file servers may not be as hardened as the sensitive databases that host the "live" copies of the data. It's very important to keep in mind that, by default, most relational database systems do not provide very strong security for backups. Because, in most cases, database backups are every bit as valuable as the live databases themselves, encryption, properly administered file system permissions, and related best practices should be followed.

Finally, data encryption can be effectively used *within* a database. Many types of systems store sensitive data, such as credit card numbers and passwords (which users might use for

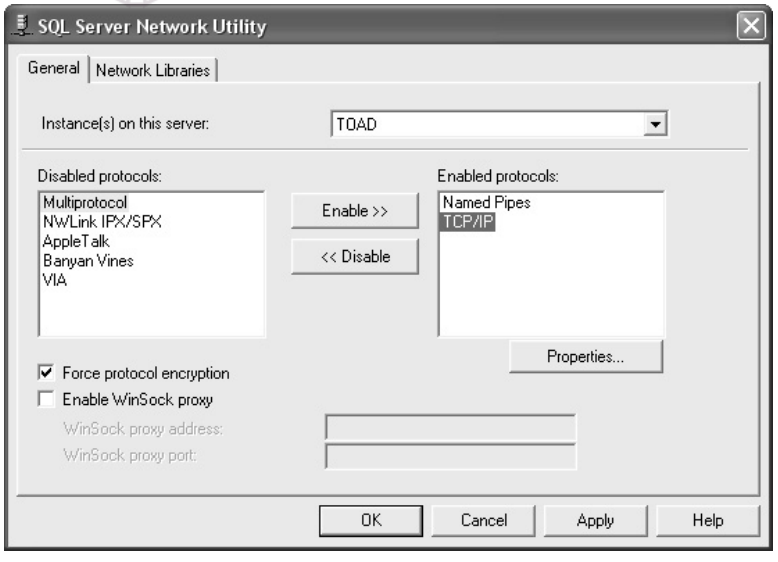

Figure 12-1 Using the Server Network Utility to configure network protocol settings for an installation of Microsoft SQL Server

several different applications). A potential problem lies in the fact that database developers and administrators often require full permissions on these tables in order to do their jobs. One way to obscure this data is to encrypt values that are stored in database tables. In this way, authorized users will be able to access and modify data, if needed, but only the calling application will be able to decipher it and make it usable. With some database vendors, like Oracle, the encryption is stored outside of the database, and in the event of key loss the data within the table/column will also be lost.

Operating System Security

On most platforms, database security goes hand in hand with operating system security. Network configuration settings, file system permissions, authentication mechanisms, and operating system encryption features can all play a role in ensuring that databases remain secure. For example, on Windows-based operating systems, only the NTFS file system offers any level of file system security (FAT and FAT32 partitions do not provide any file system security at all). In environments that use a centralized directory services infrastructure, it's important for systems administrators to keep permissions settings up to date and to ensure that unnecessary accounts are deactivated as soon as possible. Fortunately, many modern relational database platforms can leverage the strengths of the operating systems that they run on. Let's look at this in more detail.

Managing Database Logins

Most database systems require users to enter some authentication information before they can access a database. This first level of database security can be based on a standard username and password combination. Or, for improved manageability and single sign-on purposes, the database systems can be integrated with an organization's existing authentication system.

For example, many relational database products that operate on Microsoft's Windows operating system platform can utilize the security features of a domain-based security model. Based on an individual's user account and group membership, he or she can perform a seamless "pass-through authentication" that does not require rekeying a username or password. Among the many benefits of this method is the ability to centrally administer user accounts. When a user account is disabled at the level of the organization's directory service, no further steps need to be taken to prevent the user from accessing database systems. In addition, organizations are increasingly turning to biometric-based authentication (authentication through the use of fingerprint identification, retinal scans, and related methods), as well as smart-card and token-based authentication. Database administrators can take advantage of these mechanisms by relying on the operating system for identifying users. Therefore, integrated security is highly recommended, both for ease of use and for ease of management.

NOTE An important part of implementing a new database is to change the default passwords (and account names, if possible) during or immediately after installation. Many database administrators decide that they'll "get to this task later," but that usually means that it's overlooked. Using default usernames and passwords can give malicious users just the edge that they need to compromise your servers. Be sure to take a couple of minutes to close this potential vulnerability as soon as you install a new server.

Server logins can be granted permissions directly. For example, a user may be given the permission to shut down or restart a database or the ability to create a new database on the server. Login-level permissions generally apply to the server as a whole and can be used to perform tasks related to backup and recovery, performance monitoring, and the creation and deletion of databases. In some cases, users with server login permissions may be able to grant these permissions to other users. Therefore, it's very important to fully understand the security architecture of the database platform you're depending on to keep your information safe.

Another important consideration to keep in mind is that most relational database platforms allow operating system administrators to have many implicit permissions on the database. For example, systems administrators can start and stop the services and can move or delete database files. Additionally, some database platforms automatically grant to the systems administrator a database login that allows full permissions. Although this is probably desirable in some cases, it's something that must be kept in mind when trying to enforce overall security. In some situations, it's important that not all systems administrators have permissions to access sensitive data that is stored on these servers. Configuring systems in this way can be a challenge, and the exact method of implementation will be based on the operating system and database platform you're running.

Most often, a server login only allows a user to connect to a database. It does not implicitly allow the user to perform any specific actions within databases. In the next section, we'll take a look at how database-level security can be used to assign granular permissions to database logins.

Understanding Database-Level Security

Databases are commonly used to host many different databases and applications, and users should have different types of permissions based on their job functions. Once a user has been allowed to connect to a server (through the use of a server login), the user will be given only the permissions that are granted to that login. This process of determining permissions is generally known as authorization. Let's take a look at some standard types of database-level permissions.

NOTE Although the focus of this chapter is on providing technical best practices that will apply to most modern relational database platforms, we will use some examples from Microsoft's SQL Server platform to help illustrate concepts. Most of these concepts apply equally to other platforms, including Oracle's databases and IBM's DB2 platform.

The first type of database-level security is generally used to determine to which database(s) a user has access. Database administrators can specify whether or not certain databases can be accessed by a user login. For example, one login may be granted permissions to access only the Human Resources database and not any system databases or databases used by other applications.

NOTE In this section, the term "database" is used in a general sense. In these examples, a single server can host multiple, independent databases. Keep in mind that this terminology does differ in various database platforms, and the term "database" may have a slightly different meaning.

Once a user has been granted permissions to access a database, further permissions must be assigned to determine which actions he or she can take within the database. Let's look at those permissions next.

Database Administration Security

One important task related to working with a relational database is maintenance of the server itself. Important tasks include creating databases, removing unneeded databases, managing disk space allocation, monitoring performance, and performing backup and recovery operations. Database platforms allow the default systems administrator account to delegate permissions to other users, allowing them to perform these important operations.

As an example, Microsoft's SQL Server platform provides built-in server-level roles, including Database Creators, Disk Administrators, Server Administrators, Security Administrators, and many others. Figure 12-2 shows the user interface that allows the assignment of database administration permissions.

Of course, the majority of database users will not require server-level permissions. Instead, they'll need permissions that are assigned at the level of the database.

Database Roles and Permissions

As mentioned earlier in this chapter, having a valid server login only allows a user the permission to connect to a server. In order to actually access a database, the user's login must be authorized to use it. Figure 12-3 provides an example of granting database access in SQL Server.

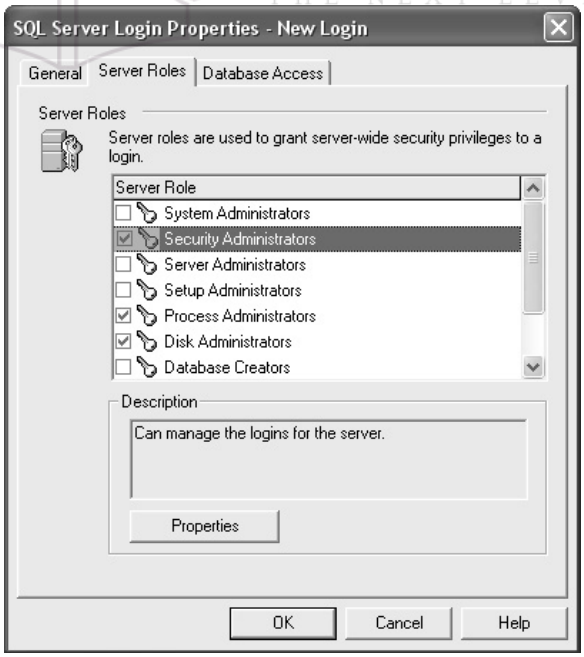

Figure 12-2 Granting database administration permissions to a user account

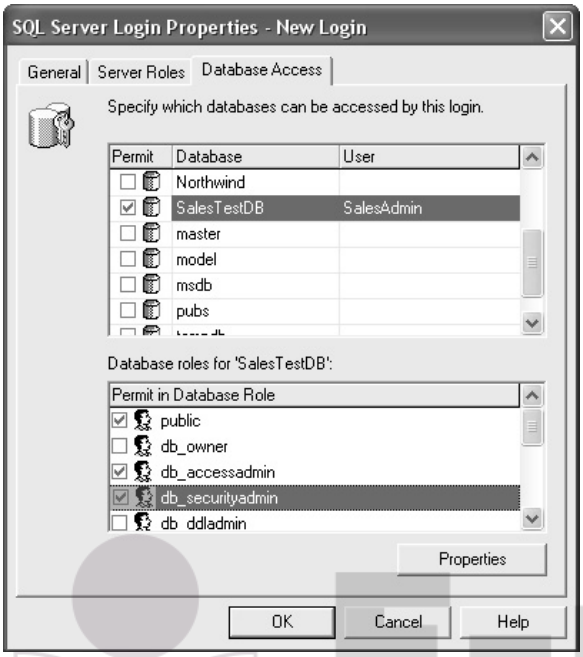

Figure 12-3 Granting database access permissions to a server login in SQL Server

The general process begins with specifying to which database(s) a login may connect. Then, permissions must be assigned within the database. The details here do vary between types of relational database platforms, but the overall concepts are the same. Generally, database administrators will create "groups" or "roles," and each of these will contain users. Specific permissions (which we'll look at in the next section) are assigned to the roles. This process is quite similar to the best practices that are suggested for most modern network operating systems. Additionally, some relational database platforms allow groups to be nested, thereby allowing you to create a hierarchy of permissions.

For example, a database administrator might create a role that allows Sales Staff to insert and update data in a specific table. Users of this role might also be able to call certain stored procedures, views, and other database objects. Another role might be created for Sales Managers. This role may be provided with the ability to delete sales-related data and make other changes within the database. Through the use of roles, database administrators can easily control which users have which permissions. Note, however, that it is very important to properly design security based on the needs of database users. Again, the principal of providing the least required permissions should be kept in mind. This is especially important since, through the use of the SQL language, well-meaning users can accidentally delete or modify data when their permissions are too lax.

Now that we've discussed database roles, let's look at the actual types of permissions that can be granted to them.

Object-Level Security

Relational databases support many different types of objects. Tables, however, are the fundamental unit of data storage. Each table is generally designed to refer to some type of entity (such as an Employee, a Customer, or an Order). Columns within these tables store details about each of these items (FirstName or CustomerNumber are common examples).

Permissions are granted to execute one or more of the most commonly used SQL commands. These commands are

- **SELECT** Retrieves information from databases. SELECT statements can obtain and combine data from many different tables, and can also be used for performing complex aggregate calculations.
- **INSERT** Adds a new row to a table.
- **UPDATE** Changes the values in an existing row or rows.
- **DELETE** Deletes rows from a table.

The ANSI Standard SQL language provides for the ability to use three commands for administering permissions to tables and other database objects:

- **GRANT** Specifies that a particular user or role will have access to perform a specific action.
- **REVOKE** Removes any current permissions settings for the specified users or roles.
- **DENY** Prevents a user or role from performing a specific action.

A typical command might look as follows:

```
Grant SELECT on EmployeeTable to HumanResourcesUser1
```

NOTE The SQL language is case insensitive (although some platforms allow case sensitivity for object and usernames). In this case, mixed case is being used for readability. Keep in mind that it's very likely that you'll need to modify this sample for your particular database platform. Consult the product's documentation for details.

Additionally, modern relational databases offer graphical methods for administering security. Figure 12-4 provides an example of setting high-level permissions on specific database objects in SQL Server. Note that these permissions are based on database tables and other objects.

Permissions can also be granted at a more granular level. In the case of specifying permissions on tables, database administrators can define permissions at the column level, as shown in Figure 12-5.

By now, you might be thinking that managing all of these levels of database security can cause significant work for a database administrator. Unfortunately, you're right. It can take a lot of time and effort initially to implement database security based on business and technical requirements, and it can take even more time and effort to ensure that database permissions reflect changes in the needs of your users. Fortunately, there are some ways to

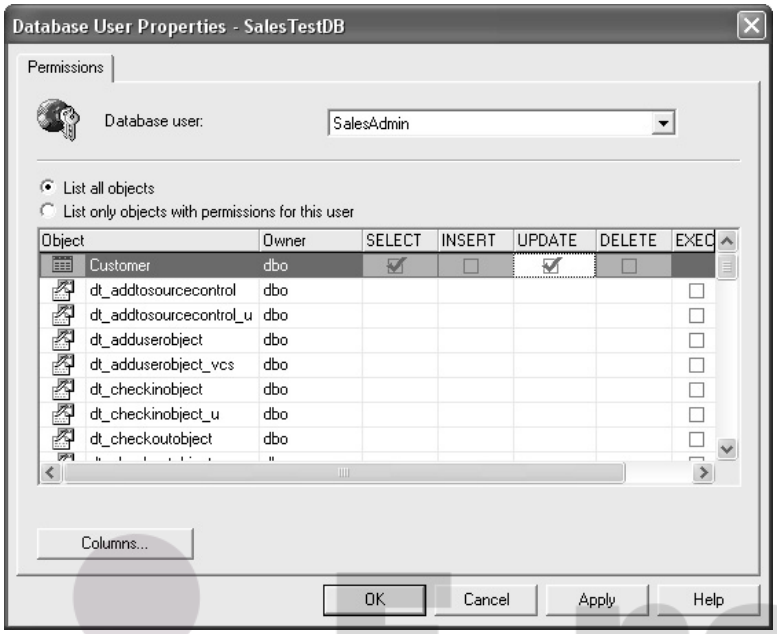

Figure 12-4 Setting object-level permission in a SQL Server database

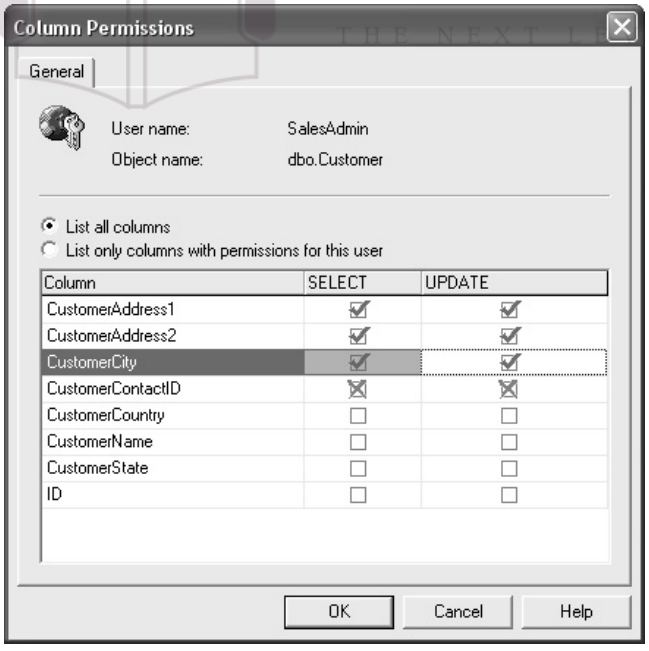

Figure 12-5 Setting column-level permission in a SQL Server database

make the management of database permissions easier. Later in this chapter, we'll talk about using application-level security. But first, let's take a look at some ways in which you can take advantage of other types of database objects for implementing and managing permissions.

NOTE Although the proper implementation of security settings is important, it's just as valuable to perform regular security settings reviews. Although the process can be tedious and time-consuming, many potential security problems can be detected before they're exploited. A good practice is to schedule and perform regular security reviews.

Using Other Database Objects for Security

In all but the simplest of databases, you will store data in many tables. And, each of these tables might have millions of rows of data. It doesn't take much imagination to see how this can lead to a lot of management effort. Fortunately, relational databases offer many other types of objects that can be used to better manage data and control access to information.

Because of the complexity and room for error, a good general recommendation is to avoid granting permissions directly on database tables. Instead, you should grant permissions to users on other database objects which, in turn, will allow them to access the data they need. In this section, we'll take a high-level look at the three commonly used database objects and how they can be used to better manage security settings.

Views

Perhaps the most commonly used method of controlling data access is views. A *view* is a logical relational database object that actually refers to one or more underlying database tables. Views are generally defined simply as the result of a SELECT query. This query, in turn, can pull information from many different tables and can also perform common calculations on the data. Figure 12-6 provides a conceptual diagram of how a view works.

Although views provide many advantages to database developers, they can also be very valuable from a security standpoint. First, views provide database administrators with a method to define granular permissions settings that would not otherwise be possible. For

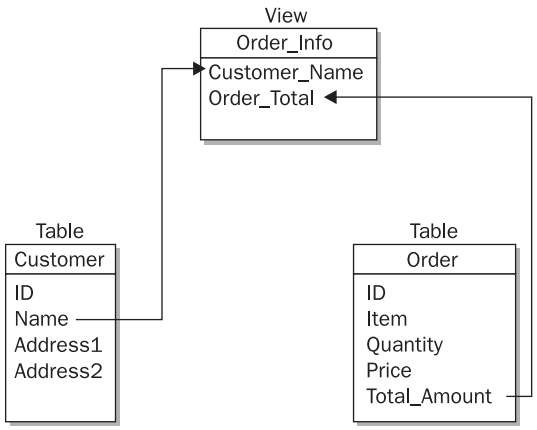

Figure 12-6 A conceptual diagram of a database view

example, you can create a view that shows basic information about employees but excludes sensitive data like their salaries and Social Security numbers. Or, you could define a view that allows users to see data for only particular employees within the company (for example, only the employees they manage).

Once a view has been defined, you can assign object-level permissions to the view. Users of the database can then use the view to access whatever information they require. Should security changes be required (if you added a "favorite color" column, for example), you can simply change the definition of the view itself, and all authorized users will be able to see this value in their result set. Furthermore, views can query other views, thereby creating a chain of objects based on business rules. When portions of the logic change, only some of the views may be affected. And, if business or technical requirements change, you can make corresponding changes in the view.

Views are generally used to return sets of data to users. Database developers can allow users to modify data through the use of views, but there are many important limitations to this method. That's where another type of database object can be helpful.

Stored Procedures

Database logic can become significantly complex, and common operations often must be performed by many different users. Thankfully, databases offer developers the ability to create and reuse SQL code through the use of objects called *stored procedures*. Stored procedures can be used to perform any function that is possible through the use of standard SQL commands. Additionally, they can take arguments (much like functions and subroutines in other programming languages), making them very flexible.

For example, a stored procedure might be used to automatically perform common operations on a set of customer-related database tables. When a customer record changes, corresponding changes can be easily made by calling the stored procedure. Related to security, and like views, instead of giving direct access to modify data stored in base tables (which in some cases might be too liberal, or your users may not completely understand how to modify the data), you can give access to stored procedures. This provides a layer of abstraction between the underlying database tables that might be affected and allows for encapsulating many of users' most common operations in manageable code modules.

Triggers

Triggers are designed to automatically be "fired" whenever specification actions take place within a database. For example, you might create a trigger on the SalesOrder table that will automatically create a corresponding row in the Invoice table. Or, you might create a trigger that performs complex data validation. (A common example would be one that checks for rules related to BeginDate and EndDate values.)

From a security standpoint, triggers can be used in different ways. First, you can use triggers to perform detailed auditing (see the section "Database Auditing and Monitoring," later in this chapter). For example, whenever a change is made to certain information in an EmployeeSalary table, you might want to notify a high-level manager, or you might write a row logging this action to another table. Another use of triggers is to enforce complex database-related rules. If your marketing staff is only allowed to add information to a table in a specific format, or if you want to ensure that a series of actions is always taken when data changes are made, you can write the appropriate trigger to do so.

Using Application Security

So far, we've looked at many general features that are available in modern relational database systems. You can define database-level permissions at levels ranging from a server login to a specific column in a specific table. In some cases, this level of security is very important. If users and administrators are granted permissions to directly access a database, the operations and data they can access must be limited. However, in the modern world, it can be tedious, at best, to have to manage database-level permissions for hundreds or thousands of users; and the problem is amplified when you're trying to support the entire world through the use of an Internet-based application.

Still, some database systems require very granular permissions. For example, users might be able to access certain information only at particular times during the day, or perhaps a complex set of logic might have to be used to determine users' effective security permissions based on other data in the database. Although it is certainly possible to implement this type of functionality using database-level permissions settings, in the real world, this process can be difficult to implement and maintain.

For these reasons, many modern database systems implement what is generally known as *application-level security*. In this method, a single database account is used by an application. This account provides the application with access to all of the databases, information, and operations that might be required by any users of the application. The application, in turn, is then made responsible for enforcing all user-level security rules. Figure 12-7 provides a simplified example of how application security works. Application security allows you to limit the number of database accounts and thus, by limiting the number of actual accounts that have database access, limit your exposure to external hacking attempts.

Large and complex database applications often enforce their own security based on business rules that are stored and enforced within the application itself. For example, an accounting package might enforce security permissions that allow a specific user to update a database only during specific hours. The application itself will use a single login and password that has access permissions to obtain and modify any data within a database. In order to secure the data, program logic within the application itself is used to determine which users can see which information.

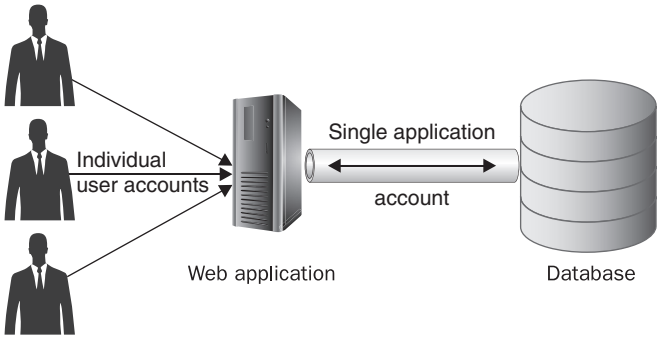

Figure 12-7 Application-level security for a database application

Another example of this might be a common ordering system for an online bookstore. Let's assume that this store records all of its information in a relational database system. Special databases contain important information such as inventory information, a book catalog, and user information. Through the front-end web servers, the online store is accessible to anyone in the world at any time. However, there are various groups of users that require different permissions. Unregistered users can only view information about specific books, while registered users have much more access. Furthermore, the online bookstore's own staff might have access to view and modify information about book costs and selling prices.

In this scenario, it would be difficult to implement and maintain all of the required security permissions at the database level. Instead, a commonly used approach would be to implement security-related rules within the web application logic. The web servers themselves would use only a single or relatively few database logins to access information stored in the database. From the viewpoint of the database, all data retrieval and modification requests coming from the web servers will be honored.

Some relational database platforms allow for implementing additional security for applications. For example, instead of simply allowing an application full access to an entire database, you might be able to control more granularly which permissions are allowed. Furthermore, you can use features such as code signing to prevent unauthorized users from creating or modifying their own application to access the database. For details, see the documentation included with your relational database platform.

Another common situation is for multiple web applications to access one or more relational databases. It's important to keep in mind that each application that requires access to your database should have a separate login. Apart from reducing the "sharing" of database authentication information, this will also allow you to better implement auditing functionality.

NOTE For highly secure applications, some implementers may want to take advantage of both application- and database-level security. This provides the added advantage of protecting against the failure or misconfiguration of one or the other type of security. It comes at a price, however, as administrators may have to make changes in two places, and the initial implementation requires significantly more effort.

Of course, application-level security is not a perfect solution. Let's take a look at some of the potential drawbacks.

Limitations of Application-Level Security

There are some important considerations to keep in mind when you implement application-level security. The first is that, by granting the "keys to the kingdom" to an application, you implicitly trust that application to manage all security for your entire system. Therefore, it is first and foremost important that you trust the application and its authors. However, you should also keep in mind that any defects or vulnerabilities in the application could easily translate into a security breach—users could access and modify without proper authorization. For this reason, it's important that applications that maintain their own security permissions are thoroughly tested.

The second major concern related to application-level security is that it does not provide any type of protection for users that can bypass the application. For example, database developers and other users might be given permissions to directly access a database. A common example is a high-level manager that must be given appropriate permissions to generate ad hoc reports based on real-time data. In this case, the user will be bypassing any current application-level logic. In some situations, this might be acceptable. For example, database developers might have "all-or-nothing" access to a database, in which the need to set granular permissions would not be important. In other situations, however, it might be necessary to provide direct database access but also maintain acceptable levels of security.

To mitigate these potential risks, real-world database applications can use a hybrid approach involving both database- and application-level permissions. For example, the majority of users of a web-based application would handle security at the level of the application. Users with special requirements—such as database developers, systems administrators, and those that require direct, real-time database access—would be given explicit permissions at the level of the database. Although this method clearly requires more effort up front, it is a good way to take advantage of database- and application-level security features.

Supporting Internet Applications

Many data-driven web sites rely on information stored in relational databases. Even relatively simple sites might store information such as registered users' e-mail addresses and passwords within database tables. Other sites and web applications might store sensitive information about users, including credit card numbers and other personal information. Internet-based applications cause an added challenge for security administrators. On one hand, it's usually important for any user in the world to be able to access a web server. On the other hand, you want to ensure that only users that are planning to use your site as it is intended are able to access it. The key requirement, therefore, is to find a secure configuration that balances accessibility and security.

A common network configuration for Internet-based applications is to prevent direct access to the databases from all but the most trusted servers (or, sometimes, networks). Figure 12-8 shows a commonly used network arrangement topology for a web-based application that is accessible via the Internet. The "front-end" web servers are accessible to all users on the Internet. The back-end databases, however, are much more protected. Note that the databases do not have a direct connection to the public Internet and that most users can only access information through the front-end web servers.

A potential point of weakness in this setup is that the overall strength of the security is dependent on the safety of the web servers. To begin with, organizations should take appropriate precautions to ensure that sensitive data is not stored on these machines. In the event that a web server is compromised, it might become possible for unauthorized users to access information stored on the databases. There are some ways to mitigate these risks.

First, web- and standard-client applications often use a "connection string" to store authentication information. For administration purposes, this information is often stored in configuration files that can be modified, as needed. It's important to ensure that these files are properly protected (through the use of encryption and file system permissions) to prevent the usability of this information in the case that it is compromised. Remember, if

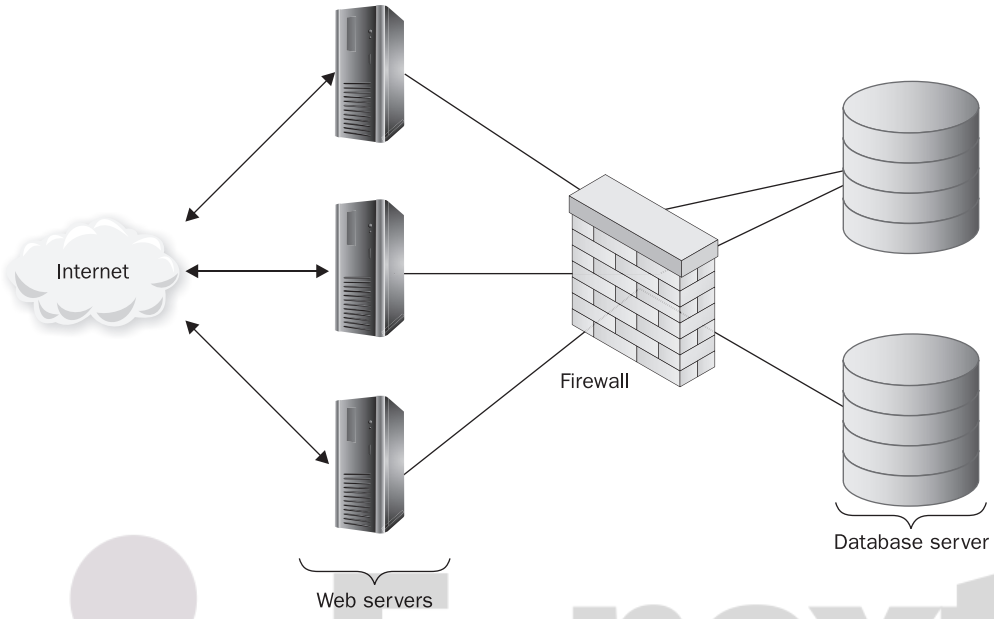

Figure 12-8 Securing Internet-accessible database applications

someone has a database connection string, they will generally be able to use it to gain full access to your databases. Better yet, the use of authentication mechanisms that are integrated with the operating system (such as Windows Authentication in the Microsoft world) can help reduce or eliminate this potential problem.

Another important mechanism for preventing errors, data corruption, or system crashes is to perform data validation in multiple places. For example, you might want to start by verifying data formats, implementing string length checks, and performing other basic data validation on the web server or client application. However, it's important not to trust this. It's relatively simple for even a novice web developer to create their own web page that circumvents these checks. Be sure also to check for data validity in any middle-tier application logic, as well as at the database level. This additional data verification can help prevent data changes from malicious users and can also help identify any missing logic in application code.

NOTE A common method of compromising applications and databases that do not perform strong data validation is known as *SQL injection*. This method involves the input of unintended code statements in data input that might actually be executed. For example, if a database query simply searches directly on the input entered by the user, the user might embed additional commands within the user input field to gain access to more or different data. Through the use of strong data validation (that is, ensuring that the input data is simple text) and the use of properly designed queries, this potential security problem can be avoided.

Database Backup and Recovery

An integral part of any overall database security strategy should be providing for database backup and recovery. Backups serve many different purposes. Most often, it seems that systems administrators perform backups to protect information in the case of server hardware failures. Although this is a very real danger in most environments, it's often not the most likely. Data can be lost due to accidental human errors, flawed application logic, defects in the database or operating system platform, and, of course, malicious users who are able to circumvent security measures. In the event that data is incorrectly modified or destroyed altogether, the only real method to recover information is from backups.

Since all relational database systems provide some method for performing database backups while a server is still running, there isn't much of an excuse for not implementing backups. The real challenge is in determining what backup strategies apply to your own environment. You'll need to find out what your working limitations are. This won't be an easy task, even in the best-managed organizations. It involves finding information from many different individuals and departments within your organization. You'll have to work hard to find existing data, and make best guesses and estimates for areas in which data isn't available.

To further complicate issues, there are many constraints in the real world that can affect the implementation of backup processes. First, resources such as storage space, network bandwidth, processing time, and local disk I/O bandwidth are almost always limited. Additionally, human resources—especially knowledgeable and experienced database administrators—may be difficult to find. And, performance requirements, user load, and other factors can prevent you from taking all the time you need to implement an ideal backup solution.

So, how do you decide what to protect? One method is to classify the importance of the relative types of information you need to protect. For example, your sales databases might be of "mission critical" importance, whereas a small decision-support system might rank "low priority" on the scale (since the data can relatively easily be re-created, if necessary). It's also important to keep in mind that business managers may have a very different idea of the importance of data when compared to other users who actually deal with this information frequently. Keep in mind that determining how to protect information must be a *team* effort if it is to be accurate and successful. An example of high-level data protection requirements is shown in Table 12-1.

Resource	Importance	Notes
OLTP server	Critical	Information can't be easily re-created, and data loss will lead to inaccurate or misleading reports.
E-mail server	High	Recovering lost messages and user mailboxes is very difficult.
Decision-support server (data warehouse)	Medium	Information can be regenerated from other sources.
Intranet web server	Medium	Content is important, but is replicated among multiple machines as part of development processes.

Table 12-1 A Sample Categorization of Data Based on Importance

Determining Backup Constraints

Once you have a reasonable idea of what your organization needs to back up, it's time to think about ways in which you can implement a data protection strategy. It is of critical importance that you define your business requirements before you look at the technical requirements for any kind of data protection solution. Table 12-2 provides an example of a requirements worksheet that summarizes data protection needs.

In addition to these requirements, you might also have a preliminary budget limit that can serve as a guideline for evaluating solutions. You should also begin thinking about personnel and the types of expertise you'll need to have available to implement a solution.

Determining Recovery Requirements

It's important to keep in mind that the purpose of data protection is not to create backups. The real purpose is to provide the ability to recover information, in case it is lost. To that end, a good practice is to begin designing a backup solution based on your recovery requirements. You should take into account the cost of downtime, the value of the data, and the amount of acceptable data loss in a worst-case scenario. Also, keep in mind the likelihood of certain types of disasters.

When planners are evaluating business needs, they may forget to factor in the potential time for recovering information. The question they should ask is the following: "If we lose data due to failure or corruption, how long will it take to get it back?" In some cases, the answer will be based on the technical limitations of the hardware you select. For example, if you back up 13GB of data to tape media and then the database becomes corrupted, the recovery time might be two hours. But what if that's not fast enough? Suppose your systems must be available within half that time—one hour. In that case, you'll need to make some important decisions. An obvious choice is to find suitable backup hardware to meet these constraints. If budgetary considerations don't allow that, however, you'll need to find

Machine	Amount of Data (est.)	Backup Window	Acceptable Downtime	Acceptable Data Loss	Other Requirements
Server 1 (file/ print services)	14GB	>12 hours	1 day	1 day	General file/ print server
Server 2 (file services)	>17GB	>6 hours	3 hours	4 hours	Engineering file server
SQL Server 1 (sales OLTP)	>6GB	>12 hours	30 minutes	1 hour	Sales order entry; must support point-in-time recovery
Shipping server	>17.5GB	>2 hours	5 minutes	None	Must remain online at all times; transactions cannot be lost

Table 12-2 Sample Data Protection Requirements Worksheet Based on Business Requirements

another way. In Chapter 32, we'll look at several technical solutions. For now, consider how long your business can *realistically* tolerate having certain information unavailable.

Now that we have some of the planning information out of the way, let's look at some technical information related to the performance impact of database backups.

Types of Database Backups

In an ideal world, you would have all of the resources you need to back up all of your data almost instantly. However, in the real world, large databases and performance requirements can often constrain the operations that can be performed (and when they can be performed). Therefore, you'll need to make some compromises. For example, instead of backing up all of your data hourly, you might have to resort to doing full backups once per week and smaller backups on other days.

Although the terminology and features vary greatly between relational database platforms, the following types of backups are possible on most systems:

- **Full backups** This type of backup consists of making a complete copy of all of the data in a database. Generally, the process can be performed while a database is up and running. On modern hardware, the performance impact of full backups may be almost negligible. Of course, it's recommended that database administrators test the performance impact of backups before implementing an overall schedule. Full backups are the basis for all other types of backups. If disk space constraints allow it, it is recommended to perform full backups frequently.
- **Differential backups** This type of backup consists of copying all of the data that has changed since the last full backup. Since differential backups contain only changes, the recovery process involves first restoring the latest full backup and then restoring the latest differential backup. Although the recovery process involves more steps (and is more time-consuming), the use of differential backups can greatly reduce the amount of disk storage space and backup time required to protect large databases.
- **Transaction log backups** Relational database systems are designed to support multiple concurrent updates to data. In order to manage contention and to ensure that all users see data that is consistent to a specific point in time, data modifications are first written to a transaction log file. Periodically, the transactions that have been logged are then committed to the actual database. Database administrators can choose to perform transaction log backups fairly frequently, since they only contain information about transactions that have occurred since the last backup. The major drawback to implementing transaction log backups is that, in order to recover a database, the last full (or differential) backup must be restored. Then, the unbroken chain of sequential transaction log files must be applied. Depending on the frequency of full backups, this might take a significant amount of time. However, transaction log backups also provide one extremely important feature that other backup types do not: point-in-time recovery. What this means is that, provided that backups have been implemented properly, database administrators can roll a database back to a specific point in time. For example, if you learn that an incorrect or unauthorized database transaction was performed at 3:00 P.M. on Friday, you will be able to restore the database to a point in time just before that transaction occurred. The end result is minimal data loss.

The various backup types that are available can be combined in order to provide flexible methods of backing up large or very busy databases. For example, you might choose to implement weekly full backups, daily differential backups, and hourly transaction log backups. Additionally, modern relational database systems allow database administrators to make backups of specific tables or portions of a database. For example, Microsoft's SQL Server platform allows database administrators to create tables on specific physical data files. These files can then be backed up and restored individually. Although using this method takes a lot of planning (for both backup and recovery operations), it can reduce backup times and provide for greater data protection on large, busy servers.

Another important consideration related to backups is where to store the database dumps that are created. The two main options are disk and tape. Both are commonly used solutions and have various pros and cons. Based on cost considerations, data volume, and performance requirements, you can choose to implement one or both of these solutions. If uptime and reliability are major concerns, your organization might also choose to implement a "hot backup" configuration (through the use of clustering or other solutions).

Now that we've covered the basics of database backup and recovery, let's take a look at a few remaining topics related to database security.

Keeping Your Servers Up to Date

An important general security best practice that also applies to databases is keeping systems up to date. In order to ensure that known vulnerabilities and server problems are repaired, you must apply the latest security and application patches. It's especially difficult to keep active databases up to date, since downtime, testing, and potential performance degradation can be real concerns. However, you should always review available updates and find out if the servers you manage have problems that are potentially solved by an update. If so, plan to install the updates as soon as you can test and deploy them.

Additionally, relevant patches should be applied to the operating system on which the database is running. Most database vendors offer support web sites that offer technical details and updates for their server platforms.

Database Auditing and Monitoring

The idea of accountability is an important one when it comes to network and database security. The process of auditing involves keeping a log of data modifications and permissions usage. Often, users that are attempting to overstep their security permissions (or users that are unauthorized altogether) can be detected and dealt with before significant damage is done; or, once data has been tampered with, auditing can provide details about the extent of loss or data changes. There's another benefit to implementing auditing: when users know that certain actions are being tracked, they might be less likely to attempt to snoop around your databases. Thus, this technique can serve as a deterrent. Unfortunately, in many environments, auditing is overlooked.

Though it won't necessarily prevent users from modifying information, auditing can be a very powerful security tool. Most relational databases provide you with the ability to track specific actions based on user roles or to track actions on specific database objects. For example, you might want to create an audit log entry whenever information in the EmployeeSalary

table is updated, or you might choose to implement auditing of logins and certain actions to deter systems administrators (who might require full permissions on a database) from casually "snooping around" in a database.

NOTE Be sure that you control permissions on auditing settings as well. Otherwise, a user with sufficient permissions could simply disable auditing, perform various actions on the system, and then re-enable auditing. To prevent this, it's also recommended that you audit any changes to the audit logging functionality itself.

Perhaps one of the reasons that auditing is not often implemented is because it requires significant planning and management. Unlike some types of "set and forget" functions, it's important to strike a balance between technical requirements and capturing enough information to provide meaningful analysis. In many cases, auditing too much information can decrease system performance. Also, audit logs can take up significant disk space. Finally, few database administrators would enjoy the task of looking through thousands of audit log entries just to find a few items that may be of interest.

Most relational database systems offer some level of auditing functionality. Even if one or more of the types of database you support does not include this feature, you can always implement your own (perhaps through the use of triggers, as described earlier in this chapter). At a minimum, most database administrators should configure logging of both successful and failed database login attempts. Although this measure, by itself, will provide limited information, it will provide for some level of accountability. Of course, capturing data is only one part of overall auditing.

Reviewing Audit Logs

In order for auditing to be truly useful, systems and database administrators should regularly review the data that has been collected. It is only through this activity that potential problems in security settings can be detected before they get worse. The challenge with reviewing audit logs is in determining what information is useful. Unfortunately, there's no simple method that will work for all situations. In some environments, you might want to perform "spot checks"—that is, review access to particularly sensitive data or review the actions that have been taken by a specific user. You may want to even have triggers set on auditing tables to automatically alert someone when some threshold of a known event has happened, such as multiple user account password changes happening at the same time.

Since activity logs can contain a lot of information, any methods for filtering the collected data can be helpful. Figure 12-9 provides an example of reviewing auditing logs and searching for important information using the tools included with SQL Server. For example, using features in Enterprise Manager, you can specify text to search for, and you can restrict the search to specific error numbers or severity levels.

Database Monitoring

Although auditing can provide an excellent way to track detailed actions, sometimes you just want to get a quick snapshot of who's using the server and for what purpose. Most databases provide easy methods for viewing this information (generally through graphical utilities). You may be able to get a quick snapshot of current database activity or view any long-running transactions that are currently in process.

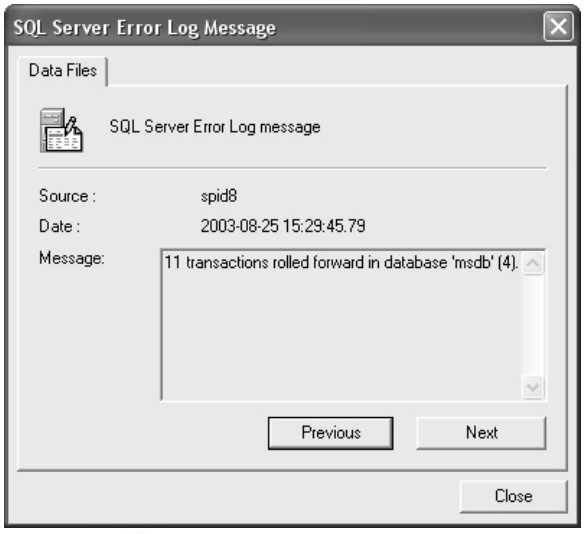

Figure 12-9 Searching for activity log information in Enterprise Manager

Although it's unlikely that you'll catch potential security breaches simply by starting at current activity information, this method can help you get a better idea of how your database is being used. By establishing a performance and usage baseline, you will be able to quickly identify any potential misuse of the system. For example, using the Windows Performance Toolkit (WPT) that is part of Microsoft's server-side operating systems, you can track many statistics related to database usage. You can also configure alerts that can be used to notify you when performance or other statistics are "out of bounds," based on normal activity. All of these mechanisms can be helpful in monitoring the usage of your database systems.

Summary

In this chapter, we covered a lot of information that is specific to implementing and maintaining security for relational databases. Although many of the same policies, procedures, tools, and techniques covered in earlier chapters also apply to databases, there are some special considerations that should be kept in mind.

We began by looking at the roles that databases can play in a typical organization. Then we examined the various levels of security that are implemented in most relational database platforms. Specifically, we looked at server-level, network-level, and database-level security. The permissions at each of these levels can help narrowly define what users can and cannot do, and can help prevent accidental or malicious data modifications.

Next, we looked at how application-level security can be used to maintain strict permissions while simplifying database administration. Another important aspect related to ensuring the security of database systems is implementing a data protection plan. We looked at the reasons for performing backups, how backups should be planned, and

various backup operations that can be performed in relational databases. Finally, we looked at the importance of auditing and monitoring servers.

Although the concepts described in this chapter may seem to require extensive effort to be mastered and performed effectively, most organizations will likely find it worthwhile to commit resources to protecting their most important information systems.

References

Basta, Alfred, and Melissa Zgola. *Database Security*. Delmar Cengage Learning, 2011.

Cherry, Denny. *Securing SQL Server: Protecting Your Database from Attackers*. Syngress, 2011.

Gertz, Michael, and Sushil Jajodia. *Handbook of Database Security: Applications and Trends*. Springer, 2010.

Kenan, Kevin. *Cryptography in the Database: The Last Line of Defense*. Addison-Wesley, 2005.

Knox, David, et al. *Applied Oracle Security: Developing Secure Database and Middleware Environments*. McGraw-Hill, 2009.

Litchfield, David, et al. *The Database Hacker's Handbook: Defending Databases*. Wiley, 2005.

Miller, Frederic, Agnes Vandome, and John McBrewster, eds. *Database Security*. Alphascript Publishing, 2012.

Rob, Peter, and Carlos Coronel. *Database Systems: Design, Implementation, and Management*. Course Technology, 2007.