Database Security

Most modern organizations rely heavily on the information stored in their database systems. It is important to find a balance between security and usability regarding database systems. Modern databases 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.

Databases in Different Capacities

Databases can be used in various capacities. Some are discussed below.

- **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. 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 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 the information is not any good 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. 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.

<u>**Security Layers**</u>

- **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 the databases will be hosted. 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.
- **Network Level Security**: Some standard “best practices” for securing databases include limiting the networks and/or network addresses that have direct access to the computer. Data encryption can a method for ensuring the safety of database information in regards to associated network. 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.
- **Operating System Security**: On most platforms, database security goes hand in hand with operating system security. Most relational database platforms allow operating system administrators to have many implicit permissions on the database. 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.

<u>**Types of Database Backups**</u>

In an ideal world, all of the resources would be available which are needed to back up all of the data almost instantly. However, in the real world, large databases and performance requirements can often constrain the operations that can be performed. Therefore, some compromises are need to be made. For example, instead of backing up all of the data hourly, one 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. 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.