Database security: protecting sensitive and critical information

Bankers would be considered negligent if they locked a bank's outer doors and left the vault's doors open at night.

Likewise, it doesn't make sense for an enterprise to lock down the network and leave databases vulnerable. Selectively protecting the most sensitive data that is at rest in databases from unauthorized access is critical, since that is where 90 percent of sensitive information resides.

There is an important distinction between network security and data security. Database security does not supercede other security technologies, such as network-layer firewalls, network monitoring, SSL-secured communications, operating system and application hardening. But data protection needs to be in place as the core element of a complete enterprise security infrastructure. There is a growing awareness of encryption technologies to protect critical corporate data.

Often companies do not realize the potential amount of risk associated with sensitive information within databases until they run an internal audit which details who has access to sensitive data. Imagine the financial damage to a company that could occur if an internal employee, such as a database administrator (DBA), who has complete access to database information, conducted a security breach regarding a secret formula, confidential business transactions, or personal customer identifiers and financial information. Also, the negative impact of media coverage about any security breaches can be severely damaging to a company's reputation, sales, customer confidence, and stock price.

When a large global investment bank conducted an audit of its proprietary banking data, it revealed that twelve DBAs had unrestricted access to their key sensitive databases and over one hundred employees had administrative access to the database's operating systems. It was decided that proprietary information in the database would be denied to employees who didn't require specific and approved access to perform their jobs.

The bank's internal audit also reported that although back-up data tapes were sent to be stored at an off-site location once a day, information was vulnerable during the backup process in the event that a data tape was lost or stolen. The CIO concluded that the database risk was high and real. He decided that the bank needed to protect against any internal compromise or outsider threat to its data about current, pending and future potential investment banking deals. A loss of the sensitive data was considered to be catastrophic to the well being of the business.

Deploying cryptographically enforced access control to information in the database at the investment bank ensures that authorized senior-level bankers can obtain the data they need. However, the encryption keys and access are not available to DBAs or other

employees in the IT department. The database security solution also protects information on back-up tapes that are stored off-site. The bank secures and stores in encrypted form root-level administrative passwords and passwords to other applications and systems (e.g. operating systems, email).

When considering ways to protect sensitive database information, it's important to ensure that the privacy protection process does not prevent authorized persons from obtaining the right data at the appropriate times. It is important that your database security solution is application transparent. This means there is no need to make any changes to the underlying applications. The benefits for deploying application-transparent database security are faster implementation and low support costs.

A key issue to consider when purchasing a database security solution is making sure you have a secure audit-trail for tracking and reporting activity around confidential data. Additional topics that must be addressed when selecting a database security technology are fast performance, the ability to work across applications, and how easy it is to implement. IT security experts often recommend selectively encrypting and securing sensitive database information at the data-item level to ensure excellent performance. You want to wrap each individual data item in a protective security, rather than simply building a firewall fence around the database. Once a firewall fence is penetrated, or if the security breach occurs from the inside, all of the data is immediately vulnerable.

One of the best ways to develop an effective database security is recognizing that securing data is essential to a company's reputation, profitability and critical business objectives. For example, as personal information such as Social Security, credit card or bank account numbers exist in more databases, there are more opportunities for identity theft. Law enforcement experts now estimate that more than half of all identity theft cases are committed by employees with access to large financial databases. Banks, companies that take credit cards and credit-rating bureaus have to place greater emphasis on safeguarding and controlling access to proprietary database information.

Audit committees have become stringent about protecting customer-related information and corporate sensitive data. Many companies are required to comply with data-privacy regulations, best practice requirements and industry guidelines regarding the usage and access to customer data.

Privacy requirements for protecting non-public personal information include: proper access control, selective encryption of stored data, separation of duties, and centralized independent audit functions. Data security is no longer an option - it is mandated by government legislation and industry regulations. For example, the U.S. Gramm-Leach-Bliley Act (GLBA) requires financial institutions and their partners to protect non-public personal data while in storage, while implementing a variety of access and security controls. Failure to comply with GLBA results in significant regulatory fines for the financial institution, and CEOs and directors can be held personally responsible and legally liable for any misuse of personally identifiable non-public information. The federal

government has stated that it has already begun checking financial institutions for GLBA compliance.

The 2002 Computer Security Institute (CSI) Computer Crime and Security Survey revealed that over half of the databases have some kind of breach on a yearly basis and the average breach is close to $4 million in losses. This percentage is staggeringly high given that these are only the security problems that companies are reporting. Organizations don't want to advertise the fact that their internal people have access to customer data and can cover up their tracks, take that data, give it to anybody, and stay undetected and employed while a crime is committed.

California recently enacted a law that mandates public disclosure of computer security breaches in which confidential information may have been compromised. The law covers not just state agencies but all private enterprises doing business in California. Starting July 1, 2003, any entity that fails to disclose that a breach has occurred could be liable for civil damages or face class actions.

There is much more illegal and unauthorized accesses to databases than corporations admit to their clients, stockholders and business partners, or report to law enforcement. According to Gartner, an estimated 70 percent of unauthorized access to information is committed by internal employees, as are more than 95 percent of intrusions that result in significant financial losses.

The insiders who commit database intrusions often have network authorization, knowledge of database access codes and a precise idea of the valuable data they want to exploit. You can assign all sorts of rights, logins, roles and passwords to restrict queries and application usage. However, if someone can simply access the database files directly (either on the server or from backup media) they can see everything and anything. Most database applications, even the most sophisticated high-end ones, store information in 'clear text' that is completely unprotected and viewable.

Given the high amounts at stake, incidents will increase and continue to be widespread, costly and commonplace. The CSI 2002 survey report noted that credit card information is the single, most common financially traded instrument that is desired by database attackers. The positive news is that database misuse or unauthorized access can be prevented with currently available database security products and new audit procedures.

Business executives are collectively acknowledging that the security and confidentiality of information needs to be a lot deeper than protecting only the perimeter. Protecting data at rest in the database can be achieved through out-of-the box application-transparent encryption technologies. Implementation time can be as fast as one to three days with negligible performance considerations. Security products are most effective when they segregate the responsibilities of access to sensitive information between the security officer and database administrators. Protecting confidential database information is not just an IT function - it is a business necessity that is critical to an organization's mission.

CH 5

Database security Requirement

Database security concerns the use of a broad range of information security controls to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network links) against compromises of their confidentiality, integrity and availability. It involves various types or categories of controls, such as technical, procedural/administrative and physical. *Database security* is a specialist topic within the broader realms of computer security, information security and risk management.

Security risks to database systems include, for example:

- Unauthorized or unintended activity or misuse by authorized database users, database administrators, or network/systems managers, or by unauthorized users or hackers (e.g. inappropriate access to sensitive data, metadata or functions within databases, or inappropriate changes to the database programs, structures or security configurations);
- Malware infections causing incidents such as unauthorized access, leakage or disclosure of personal or proprietary data, deletion of or damage to the data or programs, interruption or denial of authorized access to the database, attacks on other systems and the unanticipated failure of database services;
- Overloads, performance constraints and capacity issues resulting in the inability of authorized users to use databases as intended;
- Physical damage to database servers caused by computer room fires or floods, overheating, lightning, accidental liquid spills, static discharge, electronic breakdowns/equipment failures and obsolescence;
- Design flaws and programming bugs in databases and the associated programs and systems, creating various security vulnerabilities (e.g. unauthorized privilege escalation), data loss/corruption, performance degradation etc.;
- Data corruption and/or loss caused by the entry of invalid data or commands, mistakes in database or system administration processes, sabotage/criminal damage etc.

Ross J. Anderson has often said that by their nature large databases will never be free of abuse by breaches of security; if a large system is designed for ease of access it becomes 'insecure; if made watertight it becomes impossible to use. This is sometimes known as Anderson's Rule. Many layers and types of information security control are appropriate to databases, including:

- Access control
- Auditing
- Authentication
- Encryption
- Integrity controls
- Backups
- Application security
- Database Security applying Statistical Method

Databases have been largely secured against hackers through network security measures such as firewalls, and network-based intrusion detection systems. While network security controls remain valuable in this regard, securing the database systems themselves, and the programs/functions and data within them, has arguably become more critical as networks are increasingly opened to wider access, in particular access from the Internet. Furthermore, system, program, function and data access controls, along with the associated user identification, authentication and rights management functions, have always been important to limit and in some cases log the activities of authorized users and administrators. In other words, these are complementary approaches to database security, working from both the outside-in and the inside-out as it were.

Many organizations develop their own "baseline" security standards and designs detailing basic security control measures for their database systems. These may reflect general information security requirements or obligations imposed by corporate information security policies and applicable laws and regulations (e.g. concerning privacy, financial management and reporting systems), along with generally accepted good database security practices (such as appropriate hardening of the underlying systems) and perhaps security recommendations from the relevant database system and software vendors. The security designs for specific database systems typically specify further security administration and management functions (such as administration and reporting of user access rights, log management and analysis, database replication/synchronization and backups) along with various business-driven information security controls within the database programs and functions (e.g. data entry validation and audit trails). Furthermore, various security-related activities (manual controls) are normally incorporated into the procedures, guidelines etc. relating to the design, development, configuration, use, management and maintenance of databases.

Privileges

Two types of privileges are important relating to database security within the database environment: system privileges and object privileges.

System Privileges

System privileges allow a user to perform administrative actions in a database. These include privileges (as found in SQL Server) such as: create database, create procedure, create view, backup database, create table, create trigger, and execute.[2]

Object Privileges

Object privileges allow for the use of certain operations on database objects as authorized by another user. Examples include: usage, select, insert, update, and references.[3] –

The Principal of least Privilege, and Separation of duties:

Databases fall under internal controls, that is, data used for public reporting, annual reports, etc.) are subject to the separation of duties, meaning there must be segregation of tasks between development, and production. Each task has to be validated (via code walk-through/fresh eyes) by a third person who is not writing the actual code. The database developer should not be able to execute anything in production without an independent review of the documentation/code for the work that is being performed. Typically, the role of the developer is to pass code to a DBA; however, given the cutbacks that have resulted from the economic downturn, a DBA might not be readily available. If a DBA is not involved, it is important, at minimum, for a peer to conduct a code review. This ensures that the role of the developer is clearly separate.

Another point of internal control is adherence to the principle of providing the least amount of privileges, especially in production. To allow developers more access to get their work done, it is much safer to use impersonation for exceptions that require elevated privileges (e.g. *EXECUTE AS* or sudo to do that temporarily). Often developers may dismiss this as "overhead" while on their path to coding glory. Please be aware, however, that DBAs must do all that is considered responsible because they are the *de facto* data stewards of the organization and must comply with regulations and the law.[4]

Vulnerability Assessments to Manage Risk and Compliance

One technique for evaluating database security involves performing vulnerability assessments or penetration tests against the database. Testers attempt to find security vulnerabilities that could be used to defeat or bypass security controls, break into the database, compromise the system etc. Database administrators or information security administrators may for example use automated vulnerability scans to search out misconfiguration of controls (often referred to as 'drift') within the layers mentioned above along with known vulnerabilities within the database software. The results of such scans are used to harden the database (improve security) and close off the specific vulnerabilities identified, but other vulnerabilities often remain unrecognized and unaddressed.

In database environments where security is critical, continual monitoring for compliance with standards improves security. Security compliance requires, amongst other procedures, patch management and the review and management of permissions (especially public) granted to objects within the database. Database objects may include table or other objects listed in the Table link. The permissions granted for SQL language commands on objects are considered in this process. Compliance monitoring is similar to vulnerability assessment, except that the results of vulnerability assessments generally drive the security standards that lead to the continuous monitoring program. Essentially, vulnerability assessment is a preliminary procedure to determine risk where a compliance program is the process of on-going risk assessment.

The compliance program should take into consideration any dependencies at the application software level as changes at the database level may have effects on the application software or the application server.

Abstraction

Application level authentication and authorization mechanisms may be effective means of providing abstraction from the database layer. The primary benefit of abstraction is that of a single sign-on capability across multiple databases and platforms. A single sign-on system stores the database user's credentials and authenticates to the database on behalf of the user.

Database activity monitoring (DAM)

Another security layer of a more sophisticated nature includes real-time database activity monitoring, either by analyzing protocol traffic (SQL) over the network, or by observing local database activity on each server using software agents, or both. Use of agents or native logging is required to capture activities executed on the database server, which typically include the activities of the database administrator. Agents allow this information to be captured in a fashion that can not be disabled by the database administrator, who has the ability to disable or modify native audit logs.

Analysis can be performed to identify known exploits or policy breaches, or baselines can be captured over time to build a normal pattern used for detection of anomalous activity that could be indicative of intrusion. These systems can provide a comprehensive database audit trail in addition to the intrusion detection mechanisms, and some systems can also provide protection by terminating user sessions and/or quarantining users demonstrating suspicious behavior. Some systems are designed to support separation of duties (SOD), which is a typical requirement of auditors. SOD requires that the database administrators who are typically monitored as part of the DAM, not be able to disable or alter the DAM functionality. This requires the DAM audit trail to be securely stored in a separate system not administered by the database administration group.

CH5

Database Reliability and Integrity

Databases amalgamate data from many sources, and users expect a DBMS to provide access to the data in a reliable way. When software engineers say that software has reliability, they mean that the software runs for very long periods of time without failing. Users certainly expect a DBMS to be reliable, since the data usually are key to business or organizational needs. Moreover, users entrust their data to a DBMS and rightly expect it to protect the data from loss or damage. Concerns for reliability and integrity are general security issues, but they are more apparent with databases.

A DBMS guards against loss or damage in several ways that we study them in this section. However, the controls we consider are not absolute: No control can prevent an authorized user from inadvertently entering an acceptable but incorrect value.

Database concerns about reliability and integrity can be viewed from three dimensions:

- Database integrity: concern that the database as a whole is protected against damage, as from the failure of a disk drive or the corruption of the master database index. These concerns are addressed by operating system integrity controls and recovery procedures.
- Element integrity: concern that the value of a specific data element is written or changed only by authorized users. Proper access controls protect a database from corruption by unauthorized users.
- Element accuracy: concern that only correct values are written into the elements of a database. Checks on the values of elements can help prevent insertion of improper values. Also, constraint conditions can detect incorrect values.

Protection Features from the Operating System

~~For example, we discussed the protection an operating system provides for its users.~~ A responsible system administrator backs up the files of a database periodically along with other user files. The files are protected during normal execution against outside access by the operating system's standard access control facilities. Finally, the operating system performs certain integrity checks for all data as a part of normal read and write operations for I/O devices. These controls provide basic security for databases, but the database manager must enhance them.

Two-Phase Update

A serious problem for a database manager is the failure of the computing system in the middle of modifying data. If the data item to be modified was a long field, half of the field might show the new value, while the other half would contain the old. Even if errors of this type were spotted easily (which they are not), a more subtle problem occurs when several fields are updated and no single field appears to be in obvious error. The solution to this

problem, proposed first by Lampson and Sturgis [LAM76] and adopted by most DBMSs, uses a two-phase update.

Update Technique

During the first phase, called the intent phase, the DBMS gathers the resources it needs to perform the update. It may gather data, create dummy records, open files, lock out other users, and calculate final answers; in short, it does everything to prepare for the update, but it makes no changes to the database. The first phase is repeatable an unlimited number of times because it takes no permanent action. If the system fails during execution of the first phase, no harm is done because all these steps can be restarted and repeated after the system resumes processing.

The last event of the first phase, called committing, involves the writing of a commit flag to the database. The commit flag means that the DBMS has passed the point of no return: After committing, the DBMS begins making permanent changes.

The second phase makes the permanent changes. During the second phase, no actions from before the commit can be repeated, but the update activities of phase two can also be repeated as often as needed. If the system fails during the second phase, the database may contain incomplete data, but the system can repair these data by performing all activities of the second phase. After the second phase has been completed, the database is again complete.

Two-Phase Update Example

Suppose a database contains an inventory of a company's office supplies. The company's central stockroom stores paper, pens, paper clips, and the like, and the different departments requisition items as they need them. The company buys in bulk to obtain the best prices. Each department has a budget for office supplies, so there is a charging mechanism by which the cost of supplies is recovered from the department. Also, the central stockroom monitors quantities of supplies on hand so as to order new supplies when the stock becomes low.

Suppose the process begins with a requisition from the accounting department for 50 boxes of paper clips. Assume that there are 107 boxes in stock and a new order is placed if the quantity in stock ever falls below 100. Here are the steps followed after the stockroom receives the requisition.

1. The stockroom checks the database to determine that 50 boxes of paper clips are on hand. If not, the requisition is rejected and the transaction is finished.

2. If enough paper clips are in stock, the stockroom deducts 50 from the inventory figure in the database (107 - 50 = 57).

3. The stockroom charges accounting's supplies budget (also in the database) for 50 boxes

of paper clips.

4. The stockroom checks its remaining quantity on hand (57) to determine whether the remaining quantity is below the reorder point. Because it is, a notice to order more paper clips is generated, and the item is flagged as "on order" in the database.

5. A delivery order is prepared, enabling 50 boxes of paper clips to be sent to accounting.

All five of these steps must be completed in the order listed for the database to be accurate and for the transaction to be processed correctly.

Suppose a failure occurs while these steps are being processed. If the failure occurs before step 1 is complete, there is no harm because the entire transaction can be restarted. However, during steps 2, 3, and 4, changes are made to elements in the database. If a failure occurs then, the values in the database are inconsistent. Worse, the transaction cannot be reprocessed because a requisition would be deducted twice, or a department would be charged twice, or two delivery orders would be prepared.

When a two-phase commit is used, shadow values are maintained for key data points. A shadow data value is computed and stored locally during the intent phase, and it is copied to the actual database during the commit phase. The operations on the database would be performed as follows for a two-phase commit.

Intent:

1. Check the value of COMMIT-FLAG in the database. If it is set, this phase cannot be performed. Halt or loop, checking COMMIT-FLAG until it is not set.

2. Compare number of boxes of paper clips on hand to number requisitioned; if more are requisitioned than are on hand, halt.

3. Compute TCLIPS = ONHAND - REQUISITION.

4. Obtain BUDGET, the current supplies budget remaining for accounting department. Compute TBUDGET = BUDGET - COST, where COST is the cost of 50 boxes of clips.

5. Check whether TCLIPS is below reorder point; if so, set TREORDER = TRUE; else set TREORDER = FALSE.

Commit:

1. Set COMMIT-FLAG in database.

2. Copy TCLIPS to CLIPS in database.

3. Copy TBUDGET to BUDGET in database.

4. Copy TREORDER to REORDER in database.

5. Prepare notice to deliver paper clips to accounting department. Indicate transaction completed in log.

6. Unset COMMIT-FLAG.

With this example, each step of the intent phase depends only on unmodified values from the database and the previous results of the intent phase. Each variable beginning with T is a shadow variable used only in this transaction. The steps of the intent phase can be repeated an unlimited number of times without affecting the integrity of the database.

Once the DBMS begins the commit phase, it writes a commit flag. When this flag is set, the DBMS will not perform any steps of the intent phase. Intent steps cannot be performed after committing because database values are modified in the commit phase. Notice, however, that the steps of the commit phase can be repeated an unlimited number of times, again with no negative effect on the correctness of the values in the database.

The one remaining flaw in this logic occurs if the system fails after writing the "transaction complete" message in the log but before clearing the commit flag in the database. It is a simple matter to work backward through the transaction log to find completed transactions for which the commit flag is still set and to clear those flags.

Redundancy/Internal Consistency

Many DBMSs maintain additional information to detect internal inconsistencies in data. The additional information ranges from a few check bits to duplicate or shadow fields, depending on the importance of the data.

Error Detection and Correction Codes

One form of redundancy is error detection and correction codes, such as parity bits, Hamming codes, and cyclic redundancy checks. These codes can be applied to single fields, records, or the entire database. Each time a data item is placed in the database, the appropriate check codes are computed and stored; each time a data item is retrieved, a similar check code is computed and compared to the stored value. If the values are unequal, they signify to the DBMS that an error has occurred in the database. Some of these codes point out the place of the error; others show precisely what the correct value should be. The more information provided, the more space required to store the codes.

Shadow Fields

Entire attributes or entire records can be duplicated in a database. If the data are irreproducible, this second copy can provide an immediate replacement if an error is detected. Obviously, redundant fields require substantial storage space.

Recovery

In addition to these error correction processes, a DBMS can maintain a log of user accesses, particularly changes. In the event of a failure, the database is reloaded from a backup copy and all later changes are then applied from the audit log.

Concurrency/Consistency

Database systems are often multiuser systems. Accesses by two users sharing the same database must be constrained so that neither interferes with the other. Simple locking is done by the DBMS. If two users attempt to read the same data item, there is no conflict because both obtain the same value.

If both users try to modify the same data items, we often assume that there is no conflict because each knows what to write; the value to be written does not depend on the previous value of the data item. However, this supposition is not quite accurate.

To see how concurrent modification can get us into trouble, suppose that the database consists of seat reservations for a particular airline flight. Agent A, booking a seat for passenger Mock, submits a query to find which seats are still available. The agent knows that Mock prefers a right aisle seat, and the agent finds that seats 5D, 11D, and 14D are open. At the same time, Agent B is trying to book seats for a family of three traveling together. In response to a query, the database indicates that 8ABC and 11DEF are the two remaining groups of three adjacent unassigned seats. Agent A submits the update command

```
SELECT (SEAT-NO = '11D')
ASSIGN 'MOCK,E' TO PASSENGER-NAME
```

while Agent B submits the update sequence

```
SELECT (SEAT-NO = '11D')
ASSIGN 'EHLERS,P' TO PASSENGER-NAME
```

as well as commands for seats 11E and 11F. Then two passengers have been booked into the same seat (which would be uncomfortable, to say the least).

Both agents have acted properly: Each sought a list of empty seats, chose one seat from the

list, and updated the database to show to whom the seat was assigned. The difficulty in this situation is the time delay between reading a value from the database and writing a modification of that value. During the delay time, another user has accessed the same data.

To resolve this problem, a DBMS treats the entire queryupdate cycle as a single atomic operation. The command from the agent must now resemble "read the current value of seat PASSENGER-NAME for seat 11D; if it is 'UNASSIGNED', modify it to 'MOCK,E' (or 'EHLERS,P')." The readmodify cycle must be completed as an uninterrupted item without allowing any other users access to the PASSENGER-NAME field for seat 11D. The second agent's request to book would not be considered until after the first agent's had been completed; at that time, the value of PASSENGERNAME would no longer be 'UNASSIGNED'.

A final problem in concurrent access is readwrite. Suppose one user is updating a value when a second user wishes to read it. If the read is done while the write is in progress, the reader may receive data that are only partially updated. Consequently, the DBMS locks any read requests until a write has been completed.

Monitors

The monitor is the unit of a DBMS responsible for the structural integrity of the database. A monitor can check values being entered to ensure their consistency with the rest of the database or with characteristics of the particular field. For example, a monitor might reject alphabetic characters for a numeric field. We discuss several forms of monitors.

Range Comparisons

A range comparison monitor tests each new value to ensure that the value is within an acceptable range. If the data value is outside the range, it is rejected and not entered into the database. For example, the range of dates might be 131, "/," 112, "/," 19002099. An even more sophisticated range check might limit the day portion to 130 for months with 30 days, or it might take into account leap year for February.

Range comparisons are also convenient for numeric quantities. For example, a salary field might be limited to $200,000, or the size of a house might be constrained to be between 500 and 5,000 square feet. Range constraints can also apply to other data having a predictable form.

Range comparisons can be used to ensure the internal consistency of a database. When used in this manner, comparisons are made between two database elements. For example, a grade level from K8 would be acceptable if the record described a student at an elementary school, whereas only 912 would be acceptable for a record of a student in high school. Similarly, a person could be assigned a job qualification score of 75100 only if the person had completed college or had had at least ten years of work experience. Filters or patterns are more general types of data form checks. These can be used to verify that an automobile plate is two letters followed by four digits, or the sum of all digits of a credit card number is

a multiple of 9.

Checks of these types can control the data allowed in the database. They can also be used to test existing values for reasonableness. If you suspect that the data in a database have been corrupted, a range check of all records could identify those having suspicious values.

State Constraints

State constraints describe the condition of the entire database. At no time should the database values violate these constraints. Phrased differently, if these constraints are not met, some value of the database is in error.

In the section on two-phase updates, we saw how to use a commit flag, which is set at the start of the commit phase and cleared at the completion of the commit phase. The commit flag can be considered a state constraint because it is used at the end of every transaction for which the commit flag is not set. Earlier in this chapter, we described a process to reset the commit flags in the event of a failure after a commit phase. In this way, the status of the commit flag is an integrity constraint on the database.

For another example of a state constraint, consider a database of employees' classifications. At any time, at most one employee is classified as "president." Furthermore, each employee has an employee number different from that of every other employee. If a mechanical or software failure causes portions of the database file to be duplicated, one of these uniqueness constraints might be violated. By testing the state of the database, the DBMS could identify records with duplicate employee numbers or two records classified as "president."

Transition Constraints

State constraints describe the state of a correct database. Transition constraints describe conditions necessary before changes can be applied to a database. For example, before a new employee can be added to the database, there must be a position number in the database with status "vacant." (That is, an empty slot must exist.) Furthermore, after the employee is added, exactly one slot must be changed from "vacant" to the number of the new employee.

Simple range checks and filters can be implemented within most database management systems. However, the more sophisticated state and transition constraints can require special procedures for testing. Such user-written procedures are invoked by the DBMS each time an action must be checked.

Summary of Data Reliability

Reliability, correctness, and integrity are three closely related concepts in databases. Users trust the DBMS to maintain their data correctly, so integrity issues are very important to database security.