Module 2: Structured Query Language and Transaction Management

Unit 3: Transactions and Concurrency Management

1.0 Introduction

In a database management system (DBMS), a transaction consists of one or more data- manipulation statements and queries, each of which is reading and/or writing information into the database. A transaction is usually issued to the DBMS in SQL language wrapped in a transaction, using a pattern similar to the following:

- Begin the transaction
- Execute several data manipulations and queries

- If no errors occur then commit the transaction and end it
- If errors occur then rollback the transaction and end it

If there is no error during the execution of the transaction then the system commits the transaction. A transaction commit operation applies all data manipulations within the scope of the transaction and store the results to the database. If an error occurs during the transaction, or if the user specifies a rollback operation, the data manipulations within the transaction are not persisted to the database. In no case can a partial transaction be committed to the database since that would leave the database in an inconsistent state.

In this unit, we shall make use of use the pubs database found in Microsoft SQL Server DBMS if the need arises.

2.0 Objectives

By the end of this unit, you should be able to know:

a. What a database transaction is and what its properties are
b. How database transactions are managed
c. What concurrency control is and what role it plays in maintaining the database integr ity
d. What locking methods are and how they work
e. How database recovery management is used to maintain database integrity

3.0 MultiUser Databases

Multi-user database access is one of the DBMS properties that motivate transactions. In a Multi-user database more than one user processes the database at the same time. Several issues arise:

- How can we prevent users from interfering with each other's work?
- How can we safely process transactions on the database without corrupting or losing data?
- If there is a problem (e.g., power failure or system crash), how can we recover without loosing all of our data?

3.1 What is a Transaction?

A transaction is a logical unit of database processing, which include one or more database operations, such as insertion, deletion, modification, or retrieval operations.

Transaction processing systems are systems with large databases and hundreds of concurrent users.

A transaction must be entirely completed or aborted, no intermediate states are acceptable.
A consistent database state is one in which all data integrity constraints are satisfied.

A transaction can also be defined as a sequence of operations performed as a single logical unit of work. A transaction has four key properties that are abbreviated ACID. ACID is an acronym for Atomic, Consistent, Isolated, and Durability.

a. Atomic means that all the work in the transaction is treated as a single unit. Either it is all performed or none of it is.
b. Consistent means that a completed transaction leaves the database in a consistent internal state.
c. Isolations mean that the transaction sees the database in a consistent state. This transaction operates on a consistent view of the data. If two transactions try to update the same table, one will go first and then the other will follow.
d. Durability means that the results of the transaction are permanently stored in the system.

The simplest transaction in SQL Server is a single data modification statement. The following is a transaction even though it does not do much.

```
UPDATE  authors
SET    au_fname = 'John'
WHERE          au_id = '172-
32-1176'
```

SQL Server first writes to the log file what it is going to do. Then it does the actual update statement and finally it writes to the log that it completed the update statement. The writes to the log file are written directly to disk but the update itself is probably done to a copy of the data that resides in memory. At some future point that database will be written to disk. If the server fails after a transaction has been committed and written to the log, SQL Server will use the transaction log to "roll forward" that transaction when it starts up next.

3.1.1 Transaction Examples

Consider two users, each executing similar transactions:

Example #1:

<u>User A</u>	<u>User B</u>
Read Salary for emp 101	Read Salary for emp 101
Multiply salary by 1.03	Multiply salary by 1.04
Write Salary for emp 101	Write Salary for emp 101

Example #2:

User A	User B
Read inventory for Prod 200	Read inventory for Prod 200
Decrement inventory by 5	Decrement inventory by 7
Write inventory for Prod 200	Write inventory for Prod 200

First, what should the values for salary (in the first example) really be?

The DBMS must find a way to execute these two transactions concurrently and ensure the result is what the users (and designers) intended.

These two are examples of the Lost Update or Concurrent Update problem. Some changes to the database can be overwritten.

Consider how the operations for user's A and B might be interleaved as in example #2. Assume there are 10 units in inventory for Prod 200:

Read inventory for Prod 200	for user A
Read inventory for Prod 200	for user B
Decrement inventory by 5	for user A
Decrement inventory by 7	for user B
Write inventory for Prod 200	for user A
Write inventory for Prod 200	for user B

Or something similar like:

Read inventory for Prod 200	for user A
Decrement inventory by 5	for user A
Write inventory for Prod 200	for user A
Read inventory for Prod 200	for user B
Decrement inventory by 7	for user B
Write inventory for Prod 200	for user B

In the first case, the incorrect amount is written to the database. This is called the **Lost Update** problem because we lost the update from User A - it was overwritten by user B.

The second example works because we let user A write the new value of Prod 200 before user B can read it. Thus User B's decrement operation will fail.

Here is another example. User's A and B share a bank account. Assume an initial balance of $200.

User A reads the balance

User A deducts $100 from the balance
User B reads the balance
User A writes the new balance of $100
User B deducts $100 from the balance
User B writes the new balance of $100

The reason we get the wrong final result (remaining balance of $100) is because transaction B was allowed to read stale data. This is called **the inconsistent read** problem.

Suppose, instead of interleaving (mixing) the operations of the two transactions, we execute one after the other (note it makes no difference which order: A then B, or B then A)

User A reads the balance
User A deducts $100 from the balance
User A writes the new balance of $100
User B reads the balance (which is now $100)
User B deducts $100 from the balance
User B writes the new balance of $0

If we insist only one transaction can execute at a time, in serial order, then performance will be quite poor.

3.1.2 Multi-Statement Transactions

To make transactions a little more useful we really need to put two or more statements in them. These are called **Explicit Transactions**. For example,

```
BEGIN TRAN

UPDATE  authors
SET          au_fname = 'John'
WHERE au_id = '172-32-1176'

UPDATE  authors
SET          au_fname = 'Marg'
WHERE au_id = '213-46-8915'

COMMIT TRAN
```

Note that we have a BEGIN TRAN at the beginning and a COMMIT TRAN at the end. These statements start and complete a transaction. Everything inside these statements is considered a logical unit of work. If the system fails after the first update, neither update statement will be applied when SQL Server is restarted. The log file will contain a BEGIN TRAN but no corresponding COMMIT TRAN.

3.2 Transaction Management with SQL

Transaction support:

a. Commit
b. Rollback

User initiated transaction is executed in sequence until:

a. Commit statement is reached
b. Rollback statement is reached
c. End of a program is reached
d. Program reaches abnormal termination which leads to rollback

3.2.1 Rolling Back

You can also roll back a transaction if it does not do what you want. Consider the following transaction:

```
BEGIN TRAN

UPDATE  authors
SET          au_fname = 'John'
WHERE au_id = '172-32-1176'

UPDATE  authors SET
au_fname = 'JohnY'
WHERE city = 'Lawrence'

IF @@ROWCOUNT = 5
  COMMIT TRAN
ELSE
   ROLLBACK TRAN
```

Suppose that for whatever reason, the second update statement should update exactly five rows. If @@ROWCOUNT, which hold the number of rows affected by each statement, is five then the transaction commits otherwise it rolls back. The ROLLBACK TRAN statement undoes all the work since the matching BEGIN TRAN statement. It will not perform either update statement. Note that Query Analyzer will show you messages indicating that rows were updated but you can query the database to verify that no actual data modifications took place.

3.2.2 Transaction Log

The following are the functions of transaction log:

a. It tracks all transactions that update the database
b. It may also be used by rollback command
c. It may be used to recover from system failure
d. It log record for beginning of a transaction
e. It also log each SQL statement which include:
    i. Operation type (retrieve, update, insert, delete)
    ii. Names of objects

    iii. Before and after values for updated fields
    iv. Pointers to previous and next entries
  f. It log commit statement

3.2.3 Stored Procedures

A **stored procedure** is a subroutine available to applications accessing a relational database system. Stored procedures are actually stored in the database data dictionary.

Typical uses for stored procedures include data validation (integrated into the database) or access control mechanisms. Furthermore, stored procedures are used to consolidate and centralize logic that was originally implemented in applications. Large or complex processing that might require the execution of several SQL statements is moved into stored procedures, and all applications call the procedures only.

Hopefully most of transactions will occur in stored procedures. Let us look at the second example inside a stored procedure.

```
Create Proc TranTest1
AS
BEGIN TRAN

INSERT INTO [authors]
            ([au_id],
            [au_lname],
            [au_fname],
            [phone],
            [contract])

VALUES  ('172-32-1176',
            'Gates',
            'Bill',
            '800-BUY-MSFT',
            1)

UPDATE  authors
SET          au_fname = 'Johnzzz'
WHERE au_id = '172-32-1176'

COMMIT TRAN
GO
```

The problem with this stored procedure is that transactions do not care if the statements run correctly or not. They only care if SQL Server failed in the middle. If you run this stored procedure, it will try to insert a duplicate entry into the authors database. You will get a primary key violation error message. The message will even tell you the statement has been terminated. But the transaction is still going. The

UPDATE statement runs just fine and SQL Server then commits the transaction. The proper way to code this is:

```
Create Proc TranTest2
AS
BEGIN TRAN

INSERT INTO [authors]([au_id],
                [au_lname],
                [au_fname],
                [phone],
                [contract])
VALUES  ('172-32-1176',
                'Gates',
                'Bill',
                '800-BUY-MSFT',
                1)

IF @@ERROR <> 0
   BEGIN
ROLLBACK TRAN
     return 10
   END

UPDATE  authors
SET            au_fname = 'Johnzzz'
WHERE au_id = '172-32-1176'

IF @@ERROR <> 0
   BEGIN
ROLLBACK TRAN
     return 11
   END

COMMIT TRAN
GO
```

You will notice that we check each statement for failure. If the statement failed (i.e. @@ERROR <> 0) then we rollback the work performed so far and use the RETURN statement to exit the stored procedure. It is very important to note that if we do not check for errors after each statement we may commit a transaction improperly.

Activity A

1. What is Transaction?
   What are the properties of a transaction?

2. Explain the following transaction problems with examples
   i. Concurrent Update problem
   ii. Inconsistent read problem.

3.3 Concurrency Control and Locking

We need the ability to control how transactions are run in a multiuser database. Let us define some basic terms that are related to concurrency control:

a. **Concurrency Control** is a method for controlling or scheduling the operations in such a way that concurrent transactions can be executed. If we do concurrency control properly, then we can maximize transaction throughput while avoiding any chance.
   - Concurrency control ensure serializability of transactions in multiuser environment
   - It solve problems in multiuser environment which include:
     - Lost Updates
     - Uncommitted data
     - Inconsistent retrievals

b. **Transaction throughput**: The number of transactions we can perform in a given time period. Often reported as Transactions per second or TPS.

c. A group of two or more concurrent transactions are serializable if we can order their operations so that the final result is the same as if we had run them in serial order (one after another).

d. Consider transaction A, B, C and D. Each has 3 operations. If executing: A1, B1, A2, C1, C2, B2, A3, B3, C3 has the same result as executing: A1, A2, A3, B1, B2, B3, C1, C2, C3
   Then the above schedule of transactions and operations is serialized.

e. We need a way to guarantee that our concurrent transactions can be serialized. Locking is one such means. **Locking** is done to data items in order to reserve them for future operations.

f. A lock is a logical flag set by a transaction to alert other transactions the data item is in use.

3.3.1 The Scheduler

Scheduler establishes order of concurrent transaction execution. It interleaves execution of database operations to ensure serializability. It based its actions on concurrency control algorithms which are Locking and Time stamping.

3.3.2 Characteristics of Locks

Locks may be applied to data items in two ways: Implicit and Explicit.

Implicit Locks are applied by the DBMS, while **Explicit Locks** are applied by application programs.
Locks may be applied to:

i. a single data item (value)
ii. an entire row of a table
iii. a page (memory segment) (many rows worth)
iv. an entire table
v. an entire database

This is referred to as the Lock granularity.

Locks may be of the following types depending on the requirements of the transaction:

i. An **Exclusive Lock** prevents any other transaction from reading or modifying the locked item.
ii. A **Shared Lock** allows another transaction to read an item but prevents another transaction from writing the item.

3.3.3 Two Phase Locking

The most commonly implemented locking mechanism is called Two Phased Locking or **2PL**. 2PL is a concurrency control mechanism that ensure serializability.

2PL has two phases: Growing and shrinking.

i. A transaction acquires locks on data items it will need to complete the transaction. This is called the **growing phase**.
ii. Once one lock is released, all no other lock may be acquired. This is called the **shrinking phase**.

Let us consider our previous examples in section 3.11, this time using exclusive lock:

User A places an exclusive lock on the balance
User A reads the balance
User A deducts $100 from the balance

User B attempts to place a lock on the
balance but fails because A already has an
exclusive lock
User B is placed into a wait state
User A writes the new balance of $100
User A releases the exclusive lock on the balance

User B places an exclusive lock on the balance
User B reads the balance
User B deducts $100 from the balance
User B writes the new balance of $100
Here is a more involved illustration using exclusive and shared locks:

User A places a shared lock on item **raise_rate**
User A reads **raise_rate**
User A places an exclusive lock on item
Amy_salary User A reads **Amy_salary**

User B places a shared lock on item **raise_rate**
User B reads **raise_rate**

User A calculates a new salary as Amy_salary * (1+raise_rate)

User B places an exclusive lock on item **Bill_salary**
User B reads **Bill_salary**
User B calculates a new salary as Bill_salary * (1+raise_rate)
User B writes **Bill_salary**

User A writes **Amy_salary**
User A releases exclusive lock on **Amy_salary**

User B releases exclusive lock on
Bill_Salary User B releases shared lock on
raise_rate

User A releases shared lock on **raise_rate**

Here is another example:

User A places a shared lock on **raise_rate**

User B attempts to place an exclusive lock on **raise_rate**
Placed into a wait state

User A places an exclusive lock on item **Amy_salary**
User A reads **raise_rate**
User A releases shared lock on **raise_rate**

User B places an exclusive lock on **raise_rate**

User A reads **Amy_salary**

User B reads **raise_rate**

User A calculates a new salary as Amy_salary * (1+raise_rate)

User B writes a new **raise_rate**

User B releases exclusive lock on **raise_rate**

User A writes **Amy_salary**
User A releases exclusive lock on **Amy_salary**

3.3.4 Deadlock

Deadlock refers to a specific condition when two or more processes are each waiting for each other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a *software lock* or *soft lock*.

Deadlock occurs when two transactions wait for each other to unlock data.

Example of deadlock in database as follows:

Client applications using the database may require exclusive access to a table, and in order to gain exclusive access they ask for a *lock*. If one client application holds a lock on a table and attempts to obtain the lock on a second table that is already held by a second client application, this may lead to deadlock if the second application then attempts to obtain the lock that is held by the first application. (But this particular type of deadlock is easily prevented, e.g., by using an *all-or-none* resource allocation algorithm.)

Locking can cause problems, however, let us consider another example:

User A places an exclusive lock on item 1001
User B places an exclusive lock on item 2002
User A attempts to place an exclusive lock on item
2002 User A placed into a wait state
User B attempts to place an exclusive lock on item
1001 User B placed into a wait state

This is also called a **deadlock**. One transaction has locked some of the resources and is waiting for locks so it can complete. A second transaction has locked those needed items but is awaiting the release of locks the first transaction is holding so it can continue.

3.3.5 How Prevent Deadlock

Two main ways to deal with deadlock.

1. Prevent it in the first place by giving each transaction exclusive rights to acquire all locks needed before proceeding.

2. Allow the deadlock to occur, and then break it by aborting one of the transactions.

3.4 Database Recovery Management

There are many situations in which a transaction may not reach a commit or abort point.

a. If an operating system crashes it can terminate the DBMS processes
b. The DBMS can crash
c. Power failure i.e., The system might lose power
d. Disk or hardware failure.
e. Human error can result in deletion of critical data.

In any of these situations, data in the database may become inconsistent or lost.

Database Recovery is the process of restoring the database and the data to a consistent state. This may include restoring lost data up to the point of the event (e.g. system crash).

Two approaches are discussed in this section: Reprocessing and Rollback/Rollforward.

3.4.1 Reprocessing

In a **Reprocessing** approach, the database is periodically backed up (a database save) and all transactions applied since the last save are recorded

If the system crashes, the latest database save is restored and all of the transactions are re-applied (by users) to bring the database back up to the point just before the crash.

3.4.2 Automated Recovery with Rollback / Rollforward

In this approach, we apply a similar technique: Make periodic saves of the database (time consuming operation). However, maintain a more intelligent log of the transactions that have been applied. This transaction log Includes before images and after images

- **Before Image**: A copy of the table record (or page) of data before it was changed by the transaction.
- **After Image**: A copy of the table record (or page) of data after it was changed by the transaction.
- **Rollback**: Undo any partially completed transactions (ones in progress when the crash occurred) by applying the before images to the database.
- **Rollforward**: Redo the transactions by applying the after images to the database. This is done for transactions that were committed before the crash.

- Recovery process uses both rollback and rollforward to restore the database.
- In the worst case, we would need to rollback to the last database save and then rollforward to the point just before the crash.
- **Checkpoints** can also be taken (less time consuming) in between database saves.
- The DBMS flushes all pending transactions and writes all data to disk and transaction log.

- Database can be recovered from the last checkpoint in much less time.

3.5 Database Backup

When secondary media (disk) fails, data may become unreadable. We typically rely on backing up the database to cheaper magnetic tape or other backup medium for a copy that can be restored. However, when a DBMS is running, it is not possible to backup its files as the resulting backup copy on tape may be inconsistent.

One solution: Shut down the DBMS (and thus all applications), do a full backup - copy everything on to tape. Then start up again.

Most modern DBMS allow for incremental backups.

- An **Incremental backup** will backup only those data changed or added since the last full backup. Sometimes called a delta backup.
- Follows something like:
  a. Weekend: Do a shutdown of the DBMS, and full backup of the database onto a fresh tape(s).
  b. Nightly: Do an incremental backup onto different tapes for each night of the week.

Activity B

1. Explain the terms in relation to database transactions management:
   i. Concurrency Control
   ii. Transaction Throughput
   iii. Serialization
   iv. Implicit Locking

2. What is deadlock? How can deadlock be prevented?

4.0 Conclusion

Database transaction management is a crucial issue in order to maintain data consistency.

5.0 Summary

In this unit, we have learnt:

lv. Transactions are set of read and write operations that must either execute to completion or not at all.

lvi. A transaction has four key properties that are abbreviated ACID. ACID is an acronym for Atomic, Consistent, Isolated, and Durability. lvii. Concurrency Control is a method for controlling or scheduling the operations in such a way that concurrent transactions can be executed.

lviii. Transaction throughput is the number of transactions we can perform in a given time period.

lix. A group of two or more concurrent transactions are serializable if we can order their operations so that the final result is the same as if we had run them in serial order (one after another).

lx. Locking is one of the ways to guarantee that our concurrent transactions can be serialized.

lxi. Locks may be applied to data items in two ways: Implicit and Explicit

lxii. Deadlock refers to a specific condition when two or more transactions are each waiting for each other to release a lock.

lxiii. Database Recovery is the process of restoring the database and the data to a consistent state. This may include restoring lost data up to the point of the event (e.g. system crash). lxiv. An **Incremental backup** will backup only those data changed or added since the last full backup.

6.0 Tutor Marked Assignment

1. Explain the following SQL Transaction terms with an example:

   i. Lost Update problem
   ii. Dirty read
   iii. Non-repeatable read
   iv. Phantom read

2. Explain the following Database recovery terms:

   i. Before Image
   ii. After image
   iii. Rollback
   iv. Roll forward
   v. Checkpoints

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall

Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.

Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman

Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

w3schools.com (2009). SQL Tutorial. Retrieved April 10th, 2009, from: http://www.w3schools.com/sql/

Unit 4: Database Security

Page

1.0 Introduction

Database security refers to the protection of data against unauthorized access, alteration, or destruction. System needs to be aware of certain constraints that users must nit violate; those constraints must be specified in a suitable language and must be maintained in the system catalog; DBMS must monitor users to ensure that the constraints are enforced.

Security in a database involves mechanisms to protect the data and to ensure that it is not accessed, altered or deleted without proper authorization. Access to data should be restricted and should be protected from accidental destruction.

This unit provides an overview of database security and recovery. The need for database security, classification of database security and different type of database failures were discussed.

2.0 Objectives

By the end of this unit, you should be familiar with the following concepts:

s. Need for database security
t. Classification of database security
u. Database security at design and maintenance Level
v. Database security through access control

3.1 Database Security

The issues relating to database security will be discussed in this section.

3.1.1 Need for Database Security

The need for database security is given below:

a. In a multi-user database, multiple users try to access the data at the same time. In order to maintain the consistency of the data, database security is needed
b. For the data that are being accessed through the World Wide Web, there is need to protect the data against hackers.
c. The use of credit cards is becoming more popular, the money transaction has to be saved. More specialized software both to enter the system illegally to extract data and to analyze the information obtained are available. Hence, it is necessary to protect the data.

3.1.2 Approaches to Data Security

There are numerous aspects to the security problem, some of them are:

a. Legal, social, and ethical aspects
b. Physical controls
c. Policy questions
d. Operational problems
e. Hardware control
f. Operating system support
g. Issues that the specific concern of the database system itself

As database practitioners, we must provide a means of preventing unauthorized use of data in a database. Three areas are considered:

a. Access Control: Who should be allowed access to which databases? This is typically enforced using system level accounts with passwords.
b. Authorization: for the purposes of:
   Reading Data - Such as reading another employee's salary (using a SELECT statement for example).
   Writing Data - Such as changing a value in a database (using UPDATE or DELETE).
c. Statistical Information: Enforcing who should be allowed access to information derived from underlying databases.

3.1.3 Database Security Goals and Threats

Some of the goals of Database security are:

a. Confidentiality (Secrecy and Privacy – data are only accessible by authorized users
b. To ensure data integrity which means data can only be modified by authorized users
c. Availability – data are accessible to authorized users

Some of the threats of database security are:

a. Improper release of information caused by reading of data through intentional or accidental access by unauthorized users
b. Improper modification of data
c. Action could prevent users from accessing data for which they are authorized

3.1.4 Security Threat Classification

Security threat can be classified into accidental and intentional, according to the way they occur.

The accidental threats include human errors, software errors, and natural or accidental disasters:

a. Human errors include giving incorrect input and incorrect use of applications
b. Software errors include incorrect application of security policies, denial of access to authorized users.
c. Natural or accidental threat includes damage of hardware or software

The intentional threat includes authorized users who abuse their privileges and authority, hostile agents like unauthorized users executing improper reading or writing of data.

3.1.5 Classification of Database Security

Database security can be classified into physical and logical secur ity.

a. Physical Security: this refers to the security of hardware associated with the system and the protection of site where the computer resides. Natural events such as fire, floods, and earthquakes can be considered as part of the physical threats. It is advisable to have backup copies of databases in the face of massive disasters.
b. Logical Security: This refers to the security measures residing in the operating system or the DBMS to handle threats to the data.

3.1.6 Database Security at Design Level

It is necessary to take care of security at the database design stage. The following guidelines should be put into consideration at the design stage:

a. The database should be simple and easier to use.
b. Database must be normalized
c. You should decide the privilege for each group of users.
d. Create unique view for each user or group of users.

3.1.7 Database Security at Maintenance Level

The security issues with respect to maintenance are as follows:

a. Operating system availability
b. Confidentiality and Accountability through Authorization rules
c. Encrypt ion
d. Authentication Scheme

a. **Operating system Availability**: The operating system should verify that users
and application programs attempting to access the system are authorized.

b. **Confidentiality and Accountability:** Accountability means that the system does not allow illegal entry. Accountability is related to both prevention and detection of illegal actions. Accountability is assured by monitoring the authentication and authorization of users.

Authorization rules are controls incorporated in the data management system that restrict access to data and also restrict the actions that people may take when they access data

Authentication may be carried out by the operating system or the relational database management system. Users are giving individual account or username and password.

c. **Encryption:** This is the coding of data so that they cannot be read and understood easily. Some DBMS products include encryption routines that automatically encode sensitive data when they are stored. They also provide complementary routines for decoding the data.

Activity A

1. What are the goals of database security?
2. What do you understand by database security threat?

3.1.8 Database Security System

The person responsible for security of the database is database administrator (DBA). The database administrator must consider variety of potential threats to the system. Database administrators create authorization rules that define who can access what parts of database for what operations. Enforcement of authorization rules involves authenticating the user and ensuring that the authorization rules are not violated by access request.

The database security system stores authorization rules and enforces them for each database access. When a group of users access the data in the database, then privileges may be assigned to the groups rather than individual users. In this section we shall consider authorization subsystem of database security system

3.1.9 Authorization Subsystem

Typically secur ity for database authorization purposes is implemented in an *authorization subsystem* that monitors every transaction in the database. This is part of the DBMS

Authorization rules take into account a few main ideas:

i. **Subjects**: Individuals who perform some activity on the database. Might include specific people or a group of users.

ii. **Objects**: Database units that require authorization in order to manipulate. Database units might include an entire table, specific columns in a table, specific rows in a table, etc.
iii. **Actions**: Any action that might be performed on an object by a subject. For example: Read, Modify, Insert, Write, Delete, Grant (the ability to grant authorizations to others)
iv. **Constraint**: A more specific rule regarding an aspect of the object and action.

These elements are commonly combined into an *authorization table*

Subject	**Object**	**Action**	**Constraint**
Rasheed	Employee	Read	None
Rasheed	Employee	Insert	None
Rasheed	Employee	Modify	None
Rasheed	Employee	Grant	None
Bola	Employee	Read	Salary < 50,000
Sola	PurchaseOrder	Insert	Total < 1,000
Sola	PurchaseOrder	Modify	Total < 1,000
PayrollClerks	Employee	Read	None

- What happens as the number of subjects and objects grows?
- Presently, **no commercial DBMS support this level of authorization flexibility.**
- Typically, a DBMS supports some basic authorization models. Application developers must provide more complex constraint enforcement.

i. Subject-Based Security

a. Subjects are individually defined in the DBMS and each object and action is specified.
b. For example, user Salim (a Subject) has the following authorizations:

	Objects			
Actions	**EMPLOYEES**	**ORDERS**	**PRODUCTS**	**...**
Read	Y	Y	Y	...
Insert	N	Y	N	...

Modify	N	Y	Y	...
Delete	N	N	N	...
Grant	N	N	N	...

ii. Object-Based Security

a. Objects are individually defined in the DBMS and each subject and action is specified.
b. For example, the EMPLOYEES table (an Object) has the following authorizations:

	Subjects				
Actions	**SALIM**	**JOHN**	**GAFAR**	**DBA**	**...**
Read	Y	Y	N	Y	...
Insert	N	N	N	Y	...
Modify	N	N	N	Y	...
Delete	N	N	N	Y	...
Grant	N	N	N	Y	...

3.2 The SQL GRANT and REVOKE Statements

SQL provides two main statements for setting up authorization namely: grant and revoke.

3.2.1 SQL Grant Statement

GRANT is used to grant an *action* on an *object* to a *subject*. The SQL syntax is:

```
GRANT action1, action2 ...
ON object1, object2, ...
TO subject1, subject2
...
```

Another option of the GRANT statement is WITH GRANT OPTION. This allows the grantee to propagate the authorization to another subject.

Example:

Let us assume that we have a database with:

Tables: employees, departments, orders, products
Users: salim, john, gafar, dba

```
    GRANT INSERT, DELETE, UPDATE, SELECT
    ON employees, departments
    TO salim ;

    GRANT INSERT, SELECT
    ON
    orders
    TO
    salim;
    GRANT
    SELECT
    ON products
    TO
salim;

    GRANT SELECT
    ON employees, departments
    TO john;



    GRANT INSERT, DELETE, UPDATE, SELECT
    ON employees, departments, orders, products
    TO dba
    WITH GRANT
OPTION ;
```

Grants can also be done on specific columns in a table:

```
GRANT SELECT ON products     TO jones ;

GRANT UPDATE ON products (price) TO jones ;
```

If no GRANT statement is issued, it is assumed that no authorization is given (e.g., user GREEN above).

3.2.1 SQL Revoke Statement

REVOKE is used to revoke authorizations from subjects. The SQL syntax is:

```
    REVOKE action1, action2, ...
ON      object1, object2, ...
FROM    subject1, subject2, ...
```

Example:

If Mr. Salim leaves the company, then we should revoke the privileges given to him as follows:

```
REVOKE INSERT, DELETE, UPDATE, SELECT
ON   employees, departments
FROM salim;
```

```
REVOKE INSERT, SELECT ON orders FROM salim ;

REVOKE SELECT ON products FROM salim;
```

Many RDBMS have an ALL PRIVILEGES option that will revoke all of the privileges on an object from a subject:

```
REVOKE ALL PRIVILEGES
ON   employees, departments, orders, products
FROM salim ;
```

Activity B

1. Explain SQL commands with examples:
   a. Grant statement
   b. Revoke statement
2. Let us assume that we have a database with the following information:

Tables: employees, departments, orders, products
Users: salim, john, gafar, dba

Write SQL statement that:
   a. Grant read access to salim on employees table
   b. Grant read, insert, update access on all tables to Mr Gafar
   c. Grant modify access on orders table to Mr. John

4.0 Conclusion

Database security is the system, processes, and procedures that protect a database from unintended activity. Unintended activity can be categorized as authenticated misuse, malicious attacks or inadvertent mistakes made by authorized individuals or processes.

Databases provide many layers and types of information security, typically specified in the data dictionary, including: Access control, Auditing, Authentication, Encrypt ion, and Integrity controls

5.0 Summary

In this unit, we have learnt:

lxv. Database security refers to the protection of data against unauthorized access, alteration, or destruction.

lxvi. Access to data should be restricted and should be protected from accidental destruction. lxvii. In a multi-users environment, database security is needed in order to maintain the consistency of the data. lxviii. Some of the goals of Database security include confidentiality, integrity, and availability

lxix. Security threat can be classified into accidental and intentional, according to the way they occur.

lxx. Database security can be classified into physical and logical security.

lxxi. The database security system stores authorization rules and enforces them for each database access. lxxii. Authorization rules take into account a few main ideas such as: Subjects, Objects, Actions, and Constraints.

lxxiii. **GRANT** is used to grant an *action* on an *object* to a *subject*

lxxiv. **REVOKE** is used to revoke authorizations from subjects

6.0 Tutor Marked Assignment

1. Use the table below to answer the following questions:
   a. Write SQL statement that grant authorization to database users shown in the table
   b. Write SQL statement that revoke authorization granted to Mr. Sola

Subject	Object	Action	Constraint
Rasheed	Employee	Read	None
Rasheed	Employee	Insert	None
Rasheed	Employee	Modify	None
Rasheed	Employee	Delete	None
Bola	Employee	Read	Salary < 50,000
Sola	PurchaseOrder	Insert	Total < 1,000
Sola	PurchaseOrder	Modify	Total < 1,000
Chidi	Employee	Read	None

2. How would you ensure database security at Design and Maintenance levels?

7.0 Further Reading and other Resources

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall
Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.
Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman
Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.
w3schools.com (2009). SQL Tutorial. Retrieved April 10th, 2009, from: http://www.w3schools.com/sql/