UNIT 3 DATABASE RECOVERY AND SECURITY

Structure **Page Nos.**

3.0 INTRODUCTION

In the previous unit of this block, you have gone through the concepts of transactions and Concurrency management. In this unit we will introduce two important issues relating to database management systems.

A computer system suffers from different types of failures. A DBMS controls very critical data of an organisation and therefore must be reliable. However, the reliability of the database system is linked to the reliability of the computer system on which it runs. In this unit we will discuss recovery of the data contained in a database system following failure of various types and present the different approaches to database recovery. The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, software failures, power outages, accidents, unforeseen situations and natural or man-made disasters. Database recovery techniques are methods of making the database consistent till the last possible consistent state. The aim of recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

The second main issue that is being discussed in this unit is Database security. "Database security" is protection of the information contained in the database against unauthorised access, modification or destruction. The first condition for security is to have Database integrity. "Database integrity" is the mechanism that is applied to ensure that the data in the database is consistent.

Let us discuss all these terms in more detail in this unit.

3.1 OBJECTIVES

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

- describe the terms RECOVERY and INTEGRITY;
- describe Recovery Techniques;
- define Error and Error detection techniques, and
- describe types of Authorisation.

3.2 WHAT IS RECOVERY?

During the life of a transaction, that is, a after the start of a transaction but before the transaction commits, several changes may be made in a database state. The database during such a state is in an inconsistent state. What happens when a failure occurs at this stage? Let us explain this with the help of an example:

Assume that a transaction transfers Rs.2000/- from A's account to B's account. For simplicity we are not showing any error checking in the transaction. The transaction may be written as:

Transaction T1:

READ A
A = A – 2000
WRITE A
⟶ Failure
READ B
B = B + 2000
WRITE B
COMMIT

What would happen if the transaction fails after account A has been written back to database? As far as the holder of account A is concerned s/he has transferred the money but that has never been received by account holder B.

Why did this problem occur? Because although a transaction is considered to be atomic, yet it has a life cycle during which the database gets into an inconsistent state and failure has occurred at that stage.

What is the solution? In this case where the transaction has not yet committed the changes made by it, the partial updates need to be undone.

How can we do that? By remembering information about a transaction such as when did it start, what items it updated etc. All such details are kept in a log file. We will study about log in Section 3.3. But first let us analyse the reasons of failure.

Failures and Recovery

In practice several things might happen to prevent a transaction from completing. Recovery techniques are used to bring database, which does not satisfy consistency requirements, into a consistent state. If a transaction completes normally and commits then all the changes made by the transaction on the database are permanently registered in the database. They should not be lost (please recollect the durability property of transactions given in Unit 2). But, if a transaction does not complete normally and terminates abnormally then all the changes made by it should be discarded. An abnormal termination of a transaction may be due to several reasons, including:

a) user may decide to abort the transaction issued by him/ her
b) there might be a deadlock in the system
c) there might be a system failure.

The recovery mechanisms must ensure that a consistent state of database can be restored under all circumstances. In case of transaction abort or deadlock, the system remains in control and can deal with the failure but in case of a system failure the system loses control because the computer itself has failed. Will the results of such failure be catastrophic? A database contains a huge amount of useful information and

any system failure should be recognised on the restart of the system. The DBMS should recover from any such failures. Let us first discuss the kinds of failure for identifying how to recover.

3.2.1 Kinds of Failures

The kinds of failures that a transaction program during its execution can encounter are:

1) **Software failures:** In such cases, a software error abruptly stops the execution of the current transaction (or all transactions), thus leading to losing the state of program excution and the state/ contents of the buffers. But what is a buffer? A buffer is the portion of RAM that stores the partial contents of database that is currently needed by the transaction. The software failures can further be subdivided as:
    a) Statement or application program failure
    b) Failure due to viruses
    c) DBMS software failure
    d) Operating system failure

    A Statement of program may cause abnormal termination if it does not execute completely. This happens if during the execution of a statement, an integrity constraint gets violated. This leads to abnormal termination of the transaction due to which any prior updates made by the transaction may still get reflected in the database leaving it in an inconsistent state.

    A failure of transaction can occur if some code in a transaction program leads to its
    abnormal termination. For example, a transaction can go into an infinite loop. In such a case the only way to break the loop is to abort the program. Similarly, the failure can be traced to the operating system or DBMS and transactions are aborted abruptly. Thus part of the transaction that was executed before abort may cause some updates in database, and hence the database is updated only partially which leads to an inconsistent state of database.

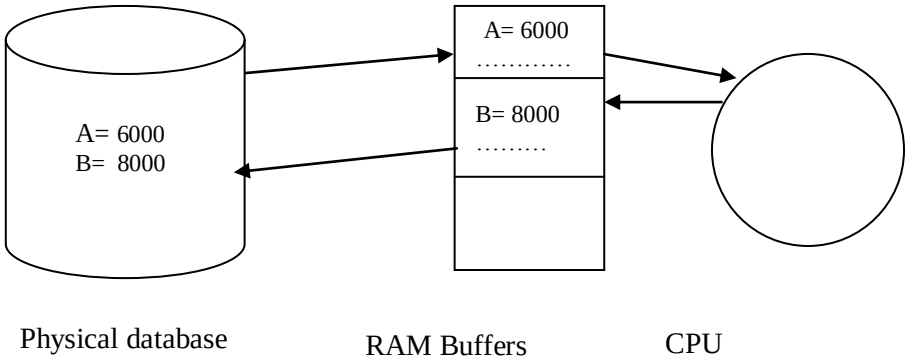
2) **Hardware failure:** Hardware failures are those failures when some hardware chip or disk fails. This may result in loss of data. One such problem can be that a disk gets damaged and cannot be read any more. This may be due to many reasons. For example, a voltage fluctuation in the power supply to the computer makes it go off or some bad sectors may come on disk or there is a disk crash. In all these cases, the database gets into an inconsistent state.

3) **External failure:** A failure can also result due to an external cause, such as fire, earthquakes, floods, etc. The database must be duly backed up to avoid problems occurring due to such failures.

In practice software failures are more common than hardware failures. Fortunately, recovery from software failures is much quicker.

The basic unit of recovery is a transaction. But, how are the transactions handled during recovery? Consider that some transactions are deadlocked, then at least one of these transactions has to be restarted to break the deadlock and thus the partial updates made by such restarted program in the database need to be **undone** so that the database does not go to an inconsistent state. So the transaction may have to be rolled back which makes sure that the transaction does not bring the database to an inconsistent state. This is one form of recovery. Let us consider a case when a transaction has committed but the changes made by the transaction have not been communicated to permanently stored physical database. A software failure now occurs and the contents of the CPU/ RAM are lost. This leaves the database in an inconsistent state. Such failure requires that on restarting the system the database be brought to a consistent state using **redo** operation. The redo operation makes the

changes made by the transaction again to bring the system to a consistent state. The database system then can be made available to the users. The point to be noted here is that the database updates are performed in the buffer in the memory. *Figure 1* shows some cases of undo and redo. You can create more such cases.



	Physical Database	RAM	Activity
Case 1	A=6000 B=8000	A=4000 B=8000	Transaction T1 has just changed the value in RAM. Now it aborts, value in RAM is lost. No problem. But we are not sure that the physical database has been written back, so must undo.
Case 2	A=4000 B=8000	A=4000 B=8000	The value in physical database has got updated due to buffer management, now the transaction aborts. The transaction must be undone.
Case 3	A=6000 B=8000	A=4000 B=10000 Commit	The value B in physical database has not got updated due to buffer management. In case of failure now when the transaction has committed. The changes of transaction must be redone to ensure transfer of correct values to physical database.

Figure 1: Database Updates And Recovery

3.2.2 Failure Controlling Methods

Failures can be handled using different recovery techniques that are discussed later in the unit. But the first question is do we really need recovery techniques as a failure control mechanism? The recovery techniques are somewhat expensive both in terms of time and in memory space for small systems. In such a case it is more beneficial to better avoid the failure by some checks instead of deploying recovery technique to make database consistent. Also, recovery from failure involves manpower that can be used in some other productive work if failure can be avoided. It is, therefore, important to find out some general precautions that help in controlling failure. Some of these precautions may be:

- having a regulated power supply.
- having a better secondary storage system such as RAID.
- taking periodic backup of database states and keeping track of transactions after each recorded state.
- properly testing the transaction programs prior to use.
- setting important integrity checks in the databases as well as user interfaces etc.

However, it may be noted that if the database system is critical it must use a DBMS that is suitably equipped with recovery procedures.

3.2.3 Database Errors

An error is said to have occurred if the execution of a command to manipulate the database cannot be successfully completed either due to inconsistent data or due to state of program. For example, there may be a command in program to store data in database. On the execution of command, it is found that there is no space/place in database to accommodate that additional data. Then it can be said that an error has occurred. This error is due to the physical state of database storage.

Broadly errors are classified into the following categories:

1) **User error:** This includes errors in the program (e.g., Logical errors) as well as errors made by online users of database. These types of errors can be avoided by applying some check conditions in programs or by limiting the access rights of online users e.g., read only. So only updating or insertion operation require appropriate check routines that perform appropriate checks on the data being entered or modified. In case of an error, some prompts can be passed to user to enable him/her to correct that error.

2) **Consistency error:** These errors occur due to the inconsistent state of database caused may be due to wrong execution of commands or in case of a transaction abort. To overcome these errors the database system should include routines that check for the consistency of data entered in the database.

3) **System error:** These include errors in database system or the OS, e.g., deadlocks. Such errors are fairly hard to detect and require reprogramming the erroneous components of the system software.

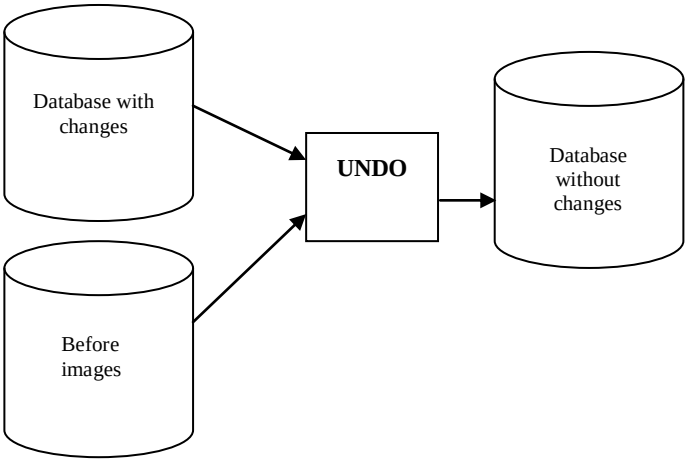
Database errors can result from failure or can cause failure and thus will require recovery. However, one of the main tasks of database system designers is to make sure that errors minimised. These concepts are also related to database integrity and have also been discusses in a later section.

3.3 RECOVERY TECHNIQUES

After going through the types of failures and database errors, let us discuss how to recover from the failures. Recovery can be done using/restoring the previous consistent state (backward recovery) or by moving forward to the next consistent state as per the committed transactions (forward recovery) recovery. Please note that a system can recover from software and hardware failures using the forward and backward recovery only if the system log is intact. What is system log? We will discuss it in more detail, but first let us define forward and backward recovery.

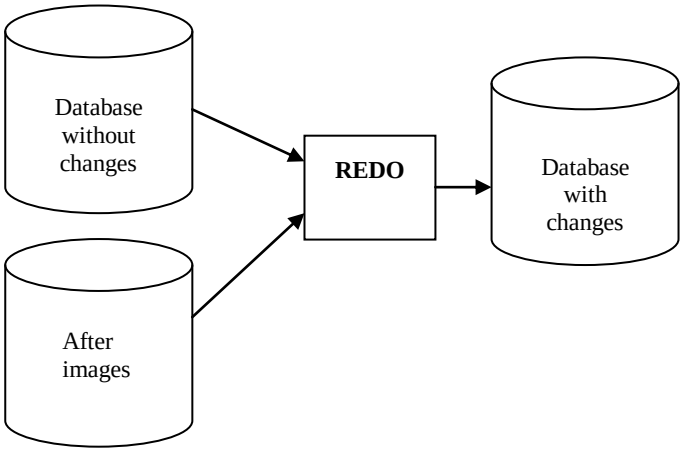
1) **Backward Recovery (UNDO)**

In this scheme the uncommitted changes made by a transaction to a database are undone. Instead the system is reset to the previous consistent state of database that is free from any errors.



2) **Forward Recovery (Redo)**

In this scheme the committed changes made by a transaction are reapplied to an earlier copy of the database.



In simpler words, when a particular error in a system is detected, the recovery system makes an accurate assessment of the state of the system and then makes the appropriate adjustment based on the anticipated results - had the system been error free.

One thing to be noted is that the Undo and Redo operations must be idempotent, i.e., executing them several times must be equivalent to executing them once. This characteristic is required to guarantee correct behaviour of database even if a failure occurs during the recovery process.

Depending on the above discussed recovery scheme, several types of recovery methods have been used. However, we define the most important recovery schemes used in most of the commercial DBMSs

Log based recovery

Let us first define the term transaction log in the context of DBMS. A transaction log is a record in DBMS that keeps track of all the transactions of a database system that update any data values in the database. A log contains the following information about a transaction:

- A transaction begin marker
- The transaction identification: The transaction id, terminal id or user id etc.

- The operations being performed by the transaction such as update, delete, insert.
- The data items or objects that are affected by the transaction including name of the table, row number and column number.
- The before or previous values (also called UNDO values) and after or changed values (also called REDO values) of the data items that have been updated.
- A pointer to the next transaction log record, if needed.
- The COMMIT marker of the transaction.

In a database system several transactions run concurrently. When a transaction commits, the data buffers used by it need not be written back to the physical database stored on the secondary storage as these buffers may be used by several other transactions that have not yet committed. On the other hand, some of the data buffers that may have updates by several uncommitted transactions might be forced back to the physical database, as they are no longer being used by the database. So the transaction log helps in remembering which transaction did which changes. Thus the system knows exactly how to separate the changes made by transactions that have already committed from those changes that are made by the transactions that did not yet commit. Any operation such as begin transaction, insert /delete/update and end transaction (commit), adds information to the log containing the transaction identifier and enough information to undo or redo the changes.

But how do we recover using log? Let us demonstrate this with the help of an example having three concurrent transactions that are active on ACCOUNTS table as:

Transaction T1	**Transaction T2**	**Transaction T3**
Read X	Read A	Read Z
Subtract 100	Add 200	Subtract 500
Write X	Write A	Write Z
Read Y		
Add 100		
Write Y		

Figure 2: The sample transactions

Assume that these transactions have the following log file (hypothetical) at a point:

Transaction Begin Marker	**Transaction Id**	**Operation on ACCOUNTS table**	**UNDO values (assumed)**	**REDO values**	**Transaction Commit Marker**
Y	T1	Sub on X Add on Y	500 800	400 Not done yet	N
Y	T2	Add on A	1000	1200	N
Y	T3	Sub on Z	900	400	Y

Figure 3: A sample (hypothetical) Transaction log

Now assume at this point of time a failure occurs, then how the recovery of the database will be done on restart.

Values	**Initial**	**Just before the failure**	**Operation Required for recovery**	**Recovered Database Values**
X	500	400 (assuming update has been done in physical database also)	UNDO	500
Y	800	800	UNDO	800

A	1000	1000 (assuming update has not been done in physical database)	UNDO	1000
Z	900	900 (assuming update has not been done in physical database)	REDO	400

Figure 4: The database recovery

The selection of REDO or UNDO for a transaction for the recovery is done on the basis of the state of the transactions. This state is determined in two steps:

- Look into the log file and find all the transactions that have started. For example, in *Figure 3*, transactions T1, T2 and T3 are candidates for recovery.
- Find those transactions that have committed. REDO these transactions. All other transactions have not committed so they should be rolled back, so UNDO them. For example, in *Figure 3* UNDO will be performed on T1 and T2; and REDO will be performed on T3.

Please note that in Figure 4 some of the values may not have yet been communicated to database, yet we need to perform UNDO as we are not sure what values have been written back to the database.
But how will the system recover? Once the recovery operation has been specified, the system just takes the required REDO or UNDO values from the transaction log and changes the inconsistent state of database to a consistent state. (Please refer to *Figure 3* and *Figure 4*).

Let us consider several transactions with their respective start & end (commit) times as shown in *Figure 5*.

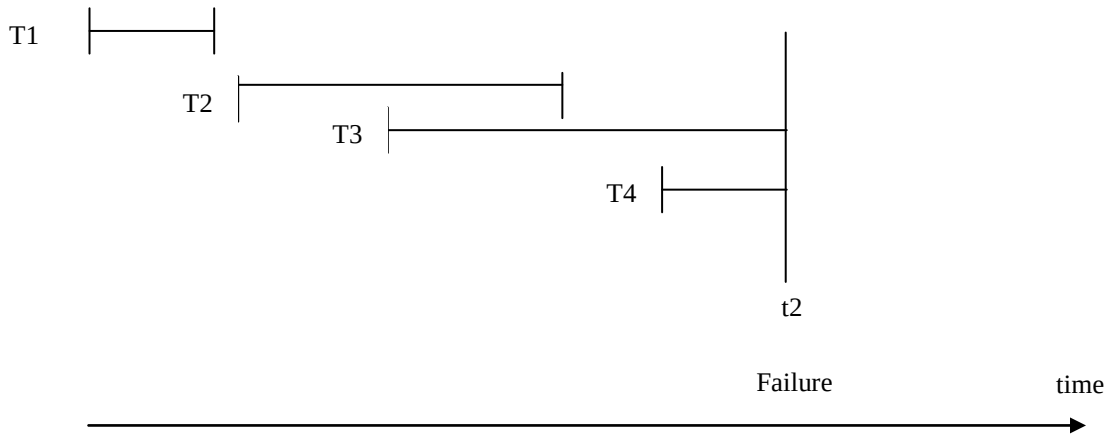


Figure 5: Execution of Concurrent Transactions

In the figure above four transactions are executing concurrently, on encountering a failure at time t2, the transactions T1 and T2 are to be REDONE and T3 and T4 will be UNDONE. But consider a system that has thousands of parallel transactions then all those transactions that have been committed may have to be redone and all uncommitted transactions need to be undone. That is not a very good choice as it requires redoing of even those transactions that might have been committed even hours earlier. So can we improve on this situation? Yes, we can take checkpoints. *Figure 6* shows a checkpoint mechanism:

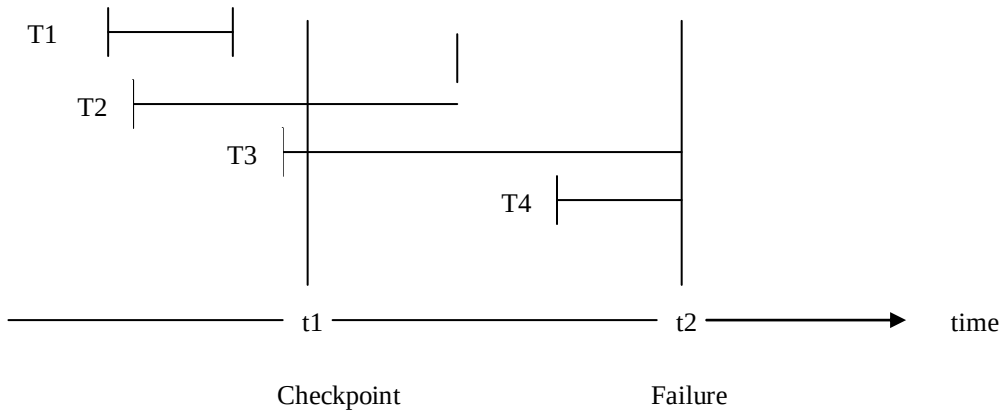


Figure 6: Checkpoint In Transaction Execution

A checkpoint is taken at time t_1 and a failure occurs at time t_2. Checkpoint transfers all the committed changes to database and all the system logs to stable storage (it is defined as the storage that would not be lost). At restart time after the failure the stable check pointed state is restored. Thus, we need to only REDO or UNDO those transactions that have completed or started after the checkpoint has been taken. The only possible disadvantages of this scheme may be that during the time of taking the checkpoint the database would not be available and some of the uncommitted values may be put in the physical database. To overcome the first problem the checkpoints should be taken at times when system load is low. To avoid the second problem some systems allow some time to the ongoing transactions to complete without restarting new transactions.

In the case of *Figure 6* the recovery from failure at t_2 will be as follows:

- The transaction T1 will not be considered for recovery as the changes made by it have already been committed and transferred to physical database at checkpoint t_1.
- The transaction T2 since it has not committed till the checkpoint t_1 but have committed before t_2, will be REDONE.
- T3 must be UNDONE as the changes made by it before checkpoint (we do not know for sure if any such changes were made prior to checkpoint) must have been communicated to the physical database. T3 must be restarted with a new name.
- T4 started after the checkpoint, and if we strictly follow the scheme in which the buffers are written back only on the checkpoint, then nothing needs to be done except restarting the transaction T4 with a new name.

The restart of a transaction requires the log to keep information of the new name of the transaction and maybe give higher priority to this newer transaction.

But one question is still unanswered that is during a failure we lose database information in RAM buffers, we may also lose log as it may also be stored in RAM buffers, so how does log ensure recovery?

The answer to this question lies in the fact that for storing transaction log we follow a **Write Ahead Log Protocol.** As per this protocol, the transaction logs are written to stable storage before any item is updated. Or more specifically it can be stated as; **the undo potion of log is written to stable storage prior to any updates and redo portion of log is written to stable storage prior to commit.**

Log based recovery scheme can be used for any kind of failure provided you have stored the most recent checkpoint state and most recent log as per write ahead log

protocol into the stable storage. Stable storage from the viewpoint of external failure requires more than one copy of such data at more than one location. You can refer to the further readings for more details on recovery and its techniques.

Check Your Progress 1

1) What is the need of recovery? What is it the basic unit of recovery?

……

……

2) What is a checkpoint? Why is it needed? How does a checkpoint help in recovery?

……

……

3) What are the properties that should be taken into consideration while selecting recovery techniques?

……

……..……..……..

3.4 SECURITY AND INTEGRITY

After going through the concepts of database recovery in the previous section, let us now deal with an important concept that helps in minimizing consistency errors in database systems. These are the concepts of database security and integrity.

Information security is the protection of information against unauthorised disclosure, alteration or destruction. Database security is the protection of information that is maintained in a database. It deals with ensuring only the "right people" get the right to access the "right data". By right people we mean those people who have the right to access/update the data that they are requesting to access/update with the database. This should also ensure the confidentiality of the data. For example, in an educational institution, information about a student's grades should be made available only to that student, whereas only the university authorities should be able to update that information. Similarly, personal information of the employees should be accessible only to the authorities concerned and not to everyone. Another example can be the medical records of patients in a hospital. These should be accessible only to health care officials.

Thus, one of the concepts of database security is primarily a specification of access rules about who has what type of access to what information. This is also known as the problem of Authorisation. These access rules are defined at the time database is defined. The person who writes access rules is called the authoriser. The process of ensuring that information and other protected objects are accessed only in authorised ways is called access control. There may be other forms of security relating to physical, operating system, communication aspects of databases. However, in this unit, we will confine ourselves mainly to authorisation and access control using simple commands.

The term integrity is also applied to data and to the mechanism that helps to ensure its consistency. Integrity refers to the avoidance of accidental loss of consistency. Protection of database contents from unauthorised access includes legal and ethical issues, organization policies as well as database management policies. To protect database several levels of security measures are maintained:

1) **Physical:** The site or sites containing the computer system must be physically secured against illegal entry of unauthorised persons.

2) **Human**: An Authorisation is given to a user to reduce the chance of any information leakage and unwanted manipulations.

3) **Operating System:** Even though foolproof security measures are taken to secure database systems, weakness in the operating system security may serve as a means of unauthorised access to the database.

4) **Network:** Since databases allow distributed or remote access through terminals or network, software level security within the network software is an important issue.

5) **Database system:** The data items in a database need a fine level of access control. For example, a user may only be allowed to read a data item and is allowed to issue queries but would not be allowed to deliberately modify the data. It is the responsibility of the database system to ensure that these access restrictions are not violated. Creating database views as discussed in Unit 1 Section 1.6.1 of this block is a very useful mechanism of ensuring database security.

To ensure database security requires implementation of security at all the levels as above. The Database Administrator (DBA) is responsible for implementing the database security policies in a database system. The organisation or data owners create these policies. DBA creates or cancels the user accounts assigning appropriate security rights to user accounts including power of granting and revoking certain privileges further to other users.

3.4.1 Relationship between Security and Integrity

Database security usually refers to access, whereas database integrity refers to avoidance of accidental loss of consistency. But generally, the turning point or the dividing line between security and integrity is not always clear. *Figure 7* shows the relationship between data security and integrity.

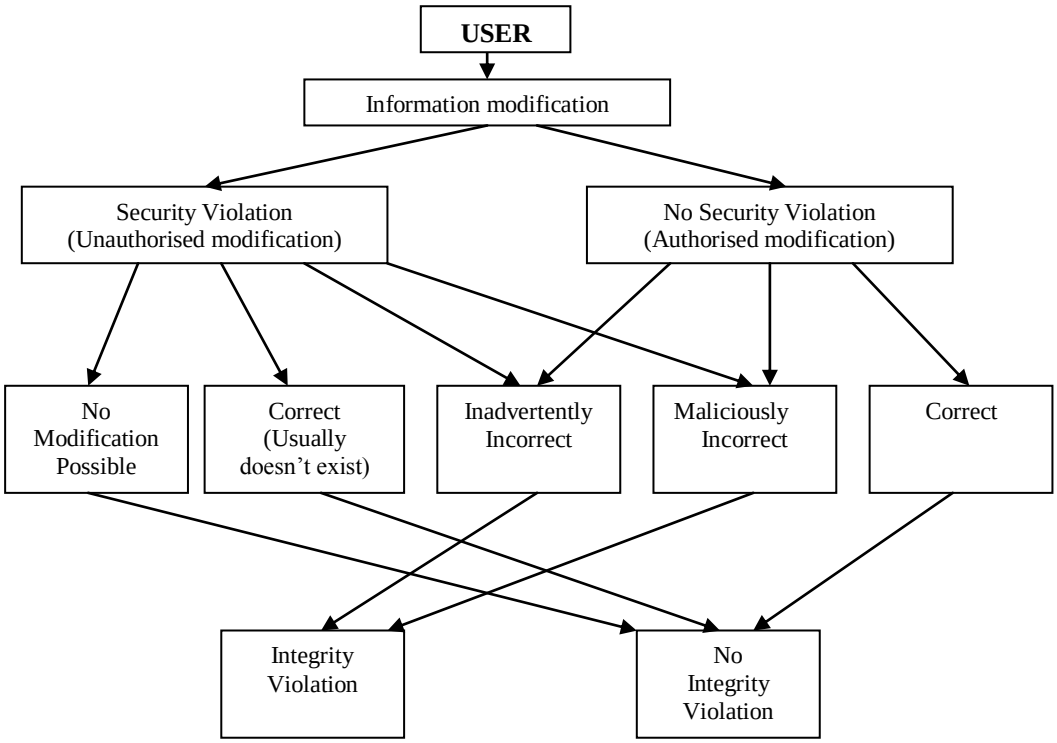


Figure 7: Relationship between security and Integrity

3.4.2 Difference between Operating System and Database Security

Security within the operating system can be implemented at several levels ranging from passwords for access to system, to the isolation of concurrent executing processes with the system. However, there are a few differences between security measures taken at operating system level as compared to those that of database system. These are:

- Database system protects more objects, as the data is persistent in nature. Also database security is concerned with different levels of granularity such as file, tuple, an attribute value or an index. Operating system security is primarily concerned with the management and use of resources.
- Database system objects can be complex logical structures such as views, a number of which can map to the same physical data objects. Moreover different architectural levels viz. internal, conceptual and external levels, have different security requirements. Thus, database security is concerned with the semantics – meaning of data, as well as with its physical representation. Operating system can provide security by not allowing any operation to be performed on the database unless the user is authorized for the operation concerned.

Figure 8 shows the architecture of a database security subsystem that can be found in any commercial DBMS.

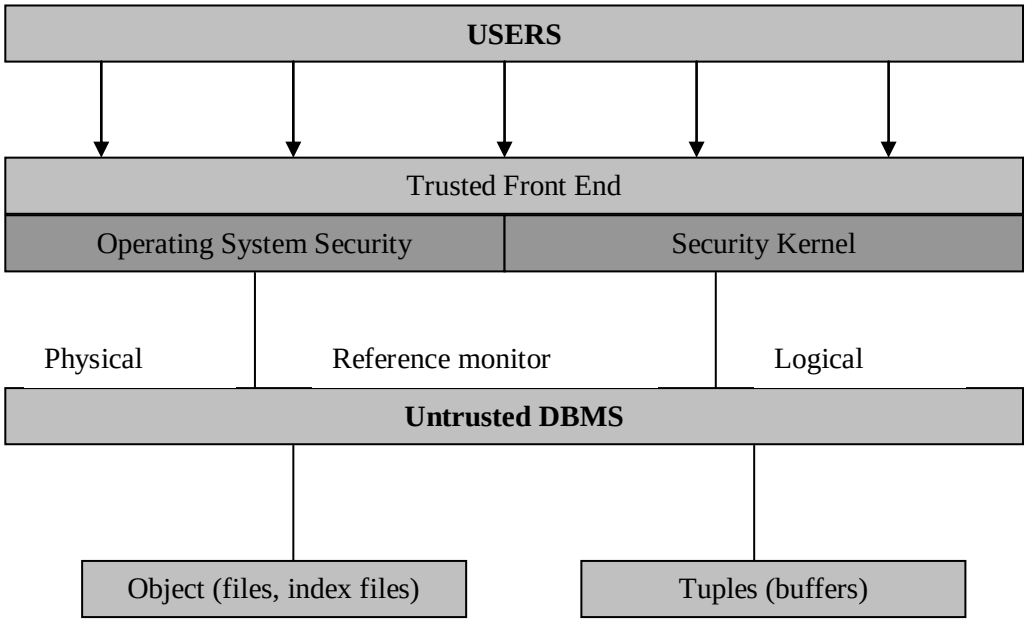


Figure 8: Database Security subsystem

3.5 AUTHORISATION

Authorisation is the culmination of the administrative policies of the organisation. As the name specifies, authorisation is a set of rules that can be used to determine which user has what type of access to which portion of the database. The following forms of authorisation are permitted on database items:

1) **READ**: it allows reading of data object, but not modification, deletion or insertion of data object.
2) **INSERT**: allows insertion of new data, but not the modification of existing data, e.g., insertion of tuple in a relation.
3) **UPDATE**: allows modification of data, but not its deletion. But data items like primary-key attributes may not be modified.

4) **DELETE**: allows deletion of data only.

A user may be assigned all, none or a combination of these types of Authorisation, which are broadly called access authorisations.

In addition to these manipulation operations, a user may be granted control operations like

1) Add: allows adding new objects such as new relations.

2) Drop: allows the deletion of relations in a database.

3) Alter: allows addition of new attributes in a relations or deletion of existing attributes from the database.

4) Propagate Access Control: this is an additional right that allows a user to propagate the access control or access right which s/he already has to some other, i.e., if user A has access right R over a relation S, then if s/he has propagate access control, s/he can propagate her/his access right R over relation S to another user B either fully or part of it. In SQL you can use WITH GRANT OPTION for this right.

You must refer to Section 1.5 of Unit 1 of this block for the SQL commands relating to data and user control.

The ultimate form of authority is given to the database administrator. He is the one who may authorize new users, restructure the database and so on. The process of Authorisation involves supplying information known only to the person the user has claimed to be in the identification procedure.

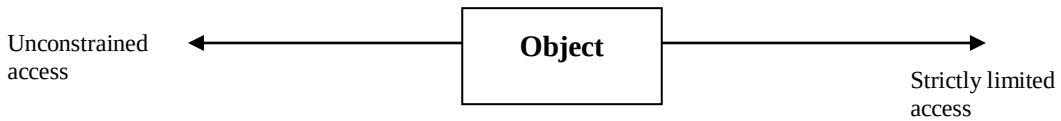
A basic model of Database Access Control

Models of database access control have grown out of earlier work on protection in operating systems. Let us discuss one simple model with the help of the following example:

Security problem: Consider the relation:

Employee (Empno, Name, Address, Deptno, Salary, Assessment)

Assuming there are two users: Personnel manager and general user . What access rights may be granted to each user? One extreme possibility is to grant an unconstrained access or to have a limited access.



One of the most influential protection models was developed by Lampson and extended by Graham and Denning. This model has 3 components:

1) A set of objects: where objects are those entities to which access must be controlled.

2) A set of subjects: where subjects are entities that request access to objects.

3) A set of all access rules: which can be thought of as forming an access (often referred to as authorisation matrix).

Let us create a sample authorisation matrix for the given relation:

Object / Subject	Empno	Name	Address	Deptno	Salary	Assessment
Personnel Manager	Read	All	All	All	All	All
General User	Read	Read	Read	Read	Not accessible	Not accessible

As the above matrix shows, Personnel Manager and general user are the two subjects. Objects of database are Empno, Name, Address, Deptno, Salary and Assessment. As per the access matrix, Personnel Manager can perform any operation on the database of an employee except for updating the Empno that may be self-generated and once given can never be changed. The general user can only read the data but cannot update, delete or insert the data into the database. Also the information about the salary and assessment of the employee is not accessible to the general user.

In summary, it can be said that the basic access matrix is the representation of basic access rules. These rules may be implemented using a view on which various access rights may be given to the users.

Check Your Progress 2

1) What are the different types of data manipulation operations and control operations?

……

……..…….

……..……..……

……..……...…..…….

……..……..……...…….

……

……

2) What is the main difference between data security and data integrity?

……

……

……

……

……

3) What are the various aspects of security problem?

……

……..……

……..……

……..……

4) Name the 3 main components of Database Access Control Model?

……

……

……

……

3.6 SUMMARY

In this unit we have discussed the recovery of the data contained in a database system after failures of various types. The types of failures that the computer system is likely to be subject to include that of components or subsystems, software failures, power outages, accidents, unforeseen situations, and natural or man-made disasters. Database recovery techniques are methods of making the database fault tolerant. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost.

The basic technique to implement database recovery is by using data redundancy in the form of logs, and archival copies of the database. Checkpoint helps the process of recovery.

Security and integrity concepts are crucial since modifications in a database require the replacement of the old values. The DBMS security mechanism restricts users to only those pieces of data that are required for the functions they perform. Security mechanisms restrict the type of actions that these users can perform on the data that is accessible to them. The data must be protected from accidental or intentional (malicious) corruption or destruction. In addition, there is a privacy dimension to data security and integrity.

Security constraints guard against accidental or malicious tampering with data; integrity constraints ensure that any properly authorized access, alteration, deletion, or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data and this correctness has to be preserved in the presence of concurrent operations, error in the user's operation and application programs, and failures in hardware and software.

3.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Recovery is needed to take care of the failures that may be due to software, hardware and external causes. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost. One of the common techniques is log-based recovery. A transaction is the basic unit of recovery.

2) A checkpoint is a point when all the database updates and logs are written to stable storage. A checkpoint ensures that not all the transactions need to be REDONE or UNDONE. Thus, it helps in faster recovery from failure. You should create a sample example, for checkpoint having a sample transaction log.

3) The following properties should be taken into consideration:

- Loss of data should be minimal
- Recovery should be quick
- Recovery should be automatic
- Affect small portion of database.

Check Your Progress 2

1) Data manipulation operations are:

   - Read
   - Insert
   - Delete
   - Update

   Data control operations are:
   - Add
   - Drop
   - Alter
   - Propagate access control

2) Data security is the protection of information that is maintained in database against unauthorised access, modification or destruction. Data integrity is the mechanism that is applied to ensure that data in the database is correct and consistent.

3)
   - Legal, social and ethical aspects
   - Physical controls
   - Policy questions
   - Operational problems
   - Hardware control
   - Operating system security
   - Database administration concerns

4) The three components are:

   - Objects
   - Subjects
   - Access rights