Database System II

Notes

By: Nahason Matoke

5.0 TRANSACTIONS MANAGEMENT AND CONCURRENCY CONTROL

A transaction is a series of actions carried out by a single user or application program, which must be treated as a single logical unit of work. It results from the execution of a user program delimited by statements (or function calls) of the form begin transactions and end transactions.

5.1 Properties Of Transactions

(i) Atomicity

It is the all or nothing property.
Either all the operations of the transactions are reflected in the database properly or none are. This means that a transaction is an indivisible unit.

(ii) Consistency

Execution of a transaction in isolation preserves the consistency of the database. So a transaction will normally transform a database from one consistent state to another consistent state.

(iii) Isolation (Independent)

Transaction execute independently of one another i.e. even though multiple transactions may execute concurrently the system quantities that for ever pair of transactions T_i and T_j it appears to T_i that either T_j finished execution after T_i started or T_j started execution after T_i finished each transactions is unaware of other transactions executing concurrently in the system.

(iii) Durability /Persistence

The effects of a successfully completed (committed) transaction are permanently recorded in the database and cannot be undone.
These properties are usually referred to as ACID properties.

5.2 Interference between Concurrent transactions

Concurrency transactions can present the following problems among others.

(i) Lost update problem
(ii) Uncommitted dependency problem
(iii) Inconsistency analysis problem

(i) **Lost Update Problem**

Another user can override an apparently successfully completed update operation by one user.

Consider this situation.

Transaction A	**Time**	**Transaction B**
Fetch R	t_1	—
	t_2	Fetch R
Update R	t_3	
	t_4	Update R

Transaction A retrieves some record R at time t_1.
Transaction B retrieves the same record R at the time t_2.
Transaction A updates the record at time t_3 on the basis of valves read at time t_1.
Transaction B updates the same record at time t_4 on the basis of values read at time t_2.
Update at t_3 is lost

(ii) Uncommitted Dependency Problem

Violations of integrity constraints governing the database can arise when 2 transactions are allowed to execute concurrently without being synchronized.
Consider.

Transaction A	***Time***	***Transaction B***
—	t_1	Fetch R
—	t_2	Update R
Fetch R	t_3	—
	t_4	Roll back

Transaction B reads R at t_1 and updates it at t_2 .
Transaction A reads an uncommitted update at time t3, and then the update is undone at time t_4.
Transaction A is therefore operating on false assumption. Transaction A becomes dependent on an uncommitted update at time t_2.

(iii) Inconsistency Analysis Problem

Transactions that only read the database can obtain the wrong result if they're allowed to read partial result or incomplete transactions, which has simultaneously updated the database. Consider 2 transactions A & B operating on an account records. Transaction A is summing account balances while transaction B is transferring amount 10 from account 3 to account 1.

Transaction A	*Time*	*Transaction B*
Fetch account1 (40) (Sum = 40)	t_1	____
Fetch account2 (50) (Sum = 90)	t_2	____
	t_3	Fetch account 3 (30)
___	t_4	Update account 3 by subtracting the mount of 10 (30-10) = 20
___	t_5	Fetch account 1 (40)
___	t_6	Updates account 1(40 + 10 = 50)
___	t_7	Commit
Fetch account 3(20) (Sum 110 instead of 120)	t_8	____

5.3 Schedules And Serialisation

A transaction consists of a sequence of reads and writes of database. The entire sequence of reads and writes by all concurrent transactions in a database taken together is known as schedule. The order of interleaving of operations from different transactions is crucial to maintaining consistency of the database.

A serial schedule is the way in which all the reads and writes of each transaction are run sequentially one after another.

A schedule is said to be serialised if all reads and writes of each transaction can be re ordered in such a way that when they are grouped together as in a serial schedule, they net affect of executing this re-organised schedule is the same as that of the original schedule.

5.4 Concurrency Control Techniques

There are 3 basic concurrency control techniques:

(i) Locking Method
(ii) Time Stamp Method
(iii) Optimistic Method

(i) Locking method

A lock guarantees exclusive use of data item to a current transaction. Transaction T_1 does not have access to a data item that is currently used by transaction T_2. A transaction acquires a lock prior to data access. The lock is released (Unlock) when the transaction is completed so that another transaction can lock the data item for its exclusive use.

Shared Locks

These are used during read operations since read operations cannot conflict. More than one transaction is permitted to hold read locks simultaneously of the same data item.

Exclusive Locks (White Locks)

These give a transaction exclusive access to a data item. As long as a transaction holds an exclusive lock no other transaction can read or update that data item.

2-Phase locking

To ensure serialisability the 2- phase locking protocol defines how transaction acquire and relinquish locks. 2-phase locking guarantees serialisability but it does not prevent deadlocks. The 2-phases are:

(a)Growing phase in which a transaction acquires all the required locks without unlocking any data. Once all the locks have been acquired the transaction is in its locked point.
(b)Stinking phase in which a transaction releases all locks and cannot obtain any new lock.

Rules governing the 2-Phase protocol are:

(i) 2 transactions cannot have conflicting locks
(ii) No unlock operation can precede an unlock operation in the same transaction.
(iii) No data is affected until all locks are obtained i.e. until the transaction is in the locked point.

Deadlocks

It is used when 2 transactions T_1 and T_2 exist in the following modes:

T_1 = access data items X and Y
T_2 = access data item Y and X

If T_1 has not unlocked Y then T_2 cannot begin.
If T_2 has not unlocked data item X, T_1 cannot continue.
Consequently T_1 and T_2 wait indefinitely each waiting for the other to unlock the required data item. Such a deadlock is known as **deadly embrace**.

Techniques To Control Deadlocks

1. Deadlock Prevention

A transaction requesting a new lock is aborted if there is a possibility that a dead lock can occur. If the transaction is aborted, all the changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlock.

2. Deadlock Detection

The DBMS periodically tests the database for deadlocks. If the deadlock is found one of the transactions (the "victim") is aborted (rolled back and restarted) and the other transaction continues.

3. Deadlock Avoidance

The transaction must obtain all the locks it needs before it can be executed. This technique avoids rolled up of conflicting transactions by requiring that locks be obtained in successions, but the serial lock assignment increase action response times.

Conclusion:

The best deadlock control method depends on the database environment, if the probability is low, deadlock detection is recommended, if probability is high, deadlock prevention is recommended and if response time is not high on the system priority list deadlock avoidance might be employed.

(ii) Time Stamping Method

The time stamping approach, to schedule concurrent transactions assign a global unique time stamp to each transaction. The time stamp value uses an explicit order in which transactions are submitted to the DBMS. The stamps must have 2 properties;

i. Uniqueness - which assures that no equal time stamp values can exist.
ii. Monotonicity - which assures that time stamp values always increase.

All database operations read and write within the same transaction must have the same time stamp. The DBMS executes conflicting operations in the time stamp order thereby ensuring serialisability of the transactions.

If 2 transactions conflict, one is often stopped, re-scheduled and assigned a new time stamp value. The main draw back of time stamping approach is that each value stored in the database requires 2 additional time- stamp fields, one for the last time the field was read and one for the last update. Time stamping thus increases the memory needs and the databases.

(iii) Optimistic Methods

The optimistic approach is based on the assumption that the majority of database operations do not conflict. The optimistic approach does not require locking or time stamping techniques; instead a

transaction is executed without restrictions until it is committed. In this approach, each transaction moves through 2 or 3 phases; read, validation and write phase.

Read Phase

The transaction reads the database, executes the needed computations and makes the updates to private copy of the database values. All update operations of the transaction are recorded in a temporary update file, which is not accessed by the remaining transactions.

Validation Phase

The transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If a validation phase is negative, the transaction is restarted and the changes are discarded.

Write Phase

The changes are permanently applied (written) to the database.

Conclusion

The optimistic approach is acceptable for mostly read or query database system that require very few update transactions.

6.0 DISTRIBUTED DATABASES

6.1 File Servers

The first attempt to alleviate centralization problems were made in traditional file processing environments. While some central (host) computer still stored all data files applications were allowed to run on separate computers, usually cheaper Pcs.
The host could be an existing mainframe or mini (the host link approach), a PC (the LAN file server approach).

Without the benefits of DBMSs, file serving has severe limitations in networked environments. Each application, on its own, must ensure integrity and security, control concurrency and recover from failures, except that now there are additional complications from the network multiple nodes.

The fact is that there is no real distribution to speak of here; that is, neither the data nor application processing form a "whole" as the concept of distribution implies.

Applications do data processing on multiple nodes but files servers are, in effect," dumb" site storage drives for application: files are downloaded from the host to application nodes to be processed, and if updates are involved, they are uploaded back to the host computer.

Transmitting files back and forth congests the network, causing performance problems. This is exacerbated by one-record-at-a- time data request that burdens the network traffic with heavy messaging.

6.2 Distributed Database Approaches

Only database approaches offer truly distributed solutions. Here different distribution strategies are based on different assumptions about access to information and, therefore are suitable for the specific circumstances that fit the underlying assumptions.

Client/Server

In database environment, some processing (database functions) is performed at the back end by the DBMS and some (application functions) at the front end by applications that suggests one obvious way to distribute by separating the two processing components on different computers. This is the approach underlying the increasingly popular (though not necessarily well understood) client/server database architecture.

Note: Client/server implies separation between database and application functions, but not necessarily across nodes. Thus DBMS and application software on the same node that achieves a clean separation comply with the definition. In practice, however, the separation is usually, across nodes, to balance the processing load using inexpensive Pcs.

In a client/server configuration every database resides on one network node, a database server, and is managed by DBMS running on that node.
If there are multiple databases on a node, each is a separate entity. There can be multiple database servers on the network work but a database does not span multiple nodes, and DBMSs running on different servers operate independently of one another.

From a database perspective, practically nothing has changed relative to a centralized environment. Database functions such as integrity and security enforcement, concurrency control and recovery are performed by the DBMS and present the results to users. Now, however, as with file servers, applications can run on network nodes separate from servers, the client (or requestor) nodes.

There are server and client software components that interface with the network. Data requests by remote applications are passed as messages to the client component, which sends then through the network to the appropriate server component, which in turn passes them to the resident DBMS for execution.
The DBMS passes the results as messages back to its server DC component, which returns them through the network to the proper client component, which passes them to the requesting application.

This approach retains the advantages of centralized database management, but by off-loading application functions to clients, the processing burden on servers is reduced. So is traffic on the network, because the amount of data traversing it is not whole files (or sections thereof), but if only the results the applications need.
While with file server's files are shipped to applications for all processing and returned for storage afterwards, with database servers application data requests are shipped to DBMS processing, and results are shipped in return to applications.

Here, the processing by the DBMS and client applications forms a "whole" so it is appropriate to label the client/server database approach distributed processing. Distribution also implies, however, treating collection of parts as a whole, and with client/server the system facilities for such treatment of the processing components is quite limited.

NOTE: There are at least two major ways to distribute processing, and as will be seen shortly, they are significantly different. Applying the same label to both, while correct, causes confusion. To distinguish between the two approaches, the term" distributed processing" is reserved for client/server, and another term is used for the other approach (see the next section). Users are advised to exercise care, however, because both terms are used rather indiscriminately.

The distribution details (what databases are available on what servers, which application nodes can request data from them, and so on) are not handled by the DBMS itself and are therefore unknown to it. In other words, in a client/server environment managing the distribution is not a database function, but the responsibility of users and applications.

For example, if a database is created on one server, the relevant client components must be explicability updated to reflect this fact (usually by the network administrator), for client applications to be able to issue data requests to it. And if the database needs to be subsequently

split into two (independent!) databases to be stored on different servers, the administrator must manually perform the split and then update server and client components accordingly. But even if this is done limitations remain.

Consider operations spanning multiple databases, possibly on multiple servers. Applications may be able to access multiple databases simultaneously, but because each database is managed independently by the DBMS on the server on which it resides, an application must:

- Issue separate data requests for each pertinent database
- Get the results from the pertinent DBMSs to a common node (usually the requesting client node)
- Execute data operations on the results it.

This means that users, including application developers, must know what data exists in which databases on what nodes, and refer to them explicitly. They are also responsible for performance optimization, which must now take network communication as well as platform differences into account.

If the operation involves multi-database updates, the application must also ensure overall integrity and recovery (because the DBMSs do not cooperate).
Of course, when the distribution changes, applications that reference the old details must be modified and re-optimized.
In other words, client/server environments do not support distribution independence in general and location independence in particular.

Implicit in the client/server approach is the assumption that database boundaries are both distinct and relatively stable.

For example, consider a bank where there are clearly delineated branch databases, each of which is frequently accessed (and maintained!) by branch personnel and only rarely accessed from other branches and headquarters.

Data from multiple branches, however, is very rarely accessed simultaneously, usually only by headquarters, and for queries only. A client/server solution yields branch databases residing on server nodes local to the branches, managed by local DBMSs.

On the rare occasions where multi-branch information is required by headquarters, data is extracted from the branch databases and processed in some user-defined way on the headquarters node.

Note: The processing can, for example, be performed in the traditional (non-database) way or by applications accessing a (redundant!) database on the headquarters node, into which data is extracted from branch databases, and which is managed by the headquarters' DBMS.

6.2.1 Distributed DBMS

Where the assumption behind client/server fits, database servers are an improvement over no distributed database systems and file servers. But in many cases, the assumption is not realistic. Database boundaries are mostly based on quite arbitrary cutoff points, whose optimally might fluctuate over time, depending on the dynamics of business reality and data access patterns.

What if, for example interaction between branches increase and they need to access each other's data more frequently? or if, in order to improve decision making, headquarters needs to access bank wide information more frequently and process it in more ways? Or if branches are frequently merged or split?

In such circumstances the database boundaries are neither distinct nor stable. There is one database, which happens to be distributed (and redistributed) on multiple nodes. In the bank's case, each branch component is still a database in its own (local) right, but it is also part of the bank wide database.
Facilities must, therefore, exist to manage the latter, and the DBMS on the network must obviously co-operate to perform multi-branch database functions and ensure distributing independence, which the client/server approach does not do.
In other words, both the database and the database functions (that is, the DBMS) are distributed, an approach referred to as distributed DBMS (DDBMS).

NOTE: Now it should be clear why, even though a distributed DBMS does involve a type of distributed processing (database processing itself is distributed on different nodes). It is more appropriate to use a different term to distinguish it from the type of distribution in client/server, where database and application processing are separated on different nodes. Distributed DBMS are sometimes referred to as co-operative processing, but this is also too broad a term, that also applies to client/server (the DBMS and applications cooperate) as well as to certain types of hardware architectures (the central processors cooperate)

Date defines a distributed database as:

A virtual database whose component parts are physically stored in a number of distinct "real" databases at a number of distinct nodes. The distributed database definition is the union of the various components database definitions.

Full DBMS support for managing a distributed database implies that:

Any single application should be able to operate transparently on data that is spread across a variety of databases, managed by (multiple) DBMSs, running on a variety of different computers, supported by a variety of operating systems, and connected together by a variety of communication networks.

6.2.1.1 Date's 12 rules of distribution.

Transparency - is the "Rule 0"b of DDBMS, as formulated by date:

0 : Distribution Independence

To the user, a distributed database should look exactly kike a non-distributed database, at least in so far as data manipulation is concerned. In other words, a DDBMS insulates applications from the physical distribution details (and changes thereof).

To comply with this rule (and thus offer full transparent), a DDBMS must achieve at least twelve objectives, also spelled out by Date. This is done usually by implementing a component that is added to all DBMSs on the network, which allows them to cooperate with one another in providing database functions spanning more than one local database.
In other words, database functions such as catalog management; query processing, concurrency control, recovery, and performance optimization become distributed themselves.

NOTE: A DDBMS usually subsumes client/server features. Many commercial products start as client/server and are gradually extended with DDBMS features. It is, however, important to point out that unless client /servers are explicitly implemented to allow such extensions, and server databases are explicitly design as components of one database, there is no guarantee that all twelve objectives will be achieved in compliance with Rule 0.

Date's Rule 0 for distributed databases can be considered the equivalent of Codd's Distribution Independence Rule, with the 12 objectives spelling out in detail what compliance with either rule (date's or Codd's) means.

Similar to Codd's 12 rules for relational databases, the objectives can be used to evaluate DDBMS product claims: Any objective that is not achieved defeats some transparency aspect, imposing costs on users.
Products vary on which objectives they achieve and to what degree, and users must understand the practical consequences of the objectives and make decisions based on how consequential each is for their specific environment.
It is, however, important to note that, like Codd's rules the objectives are not independent, and giving up any will have ramifications for the others.

1 : Local Autonomy

As much as possible, no node should depend on any other node for its successful functioning.

This objective is rather straightforward. The fact that local databases are accessed remotely should not prevent local DBMSs from operating effectively, regardless of whether other nodes are on-lime or not.

Note: There are certain circumstances where dependence on other nodes is unavoidable.

2: NO Reliance On A Central Site

There must not be any reliance on a central "master" node for some central service, such that the entire system is dependent on that central site.

Obviously, if this objective is not achieved, the system is not fully distributed, but at least partially centralized.

Note: If there is reliance on a central site; there is no local autonomy. On the other hand, the absence of such reliance does not automatically imply local autonomy.

3: Continuous Operation

There should, ideally, never be any need for a planned system shutdown.

If shutdowns are required to perform certain functions (such a adding new node or upgrading the DBMS on an existing node,) transparency is affected, because then the DDBMS does not operate exactly as a non-distributed one, which requires no such shutdowns.

4: Location Independence

Users should not have to know where data is physically stored and should be able to behave - from a logical standpoint - as if the data were all stored on their own local node.

This objective is not achieved by client/server implementations. Other than simplifying data access (and application development), the major benefit from it , is that the database can be first distributed or redistributed without impairing applications.

5: Fragmentation Independence

A major purpose of distribution is to store data as physically close to its most frequent users as possible (to maximize performance) without preventing other users from accessing it logically as if it were local to them.

Suppose another location is added to existing organization database, and some departments, employees, projects, tasks, and assignments, will migrate to it. Some subset of the database representing those entities will also migrate, distributing the database into two locations.
One option is to redesign the database, by substitution two base tables for every database table, and store each of two in relevant location. Note, however, that unless full transparency cans be achieved with views applications accessing the initial tables will have to be modified.

But with a DDBMS supporting fragmentation of logical database objects into fragments physically stored on different nodes, such complications can be avoided. For example, the EMPLOYEES table will be horizontally split into two physical fragments, each of which will reside on a different node (tables can also be vertical fragmented).
Fragmentation does not impair existing applications, because in so far as users are concerned, the database is logically the same. When databases are first fragmented or re-fragmented, or when fragments are redistributed, the DDBMS keeps track of fragmentation details in its **distributed global catalog**, and transparently reconstructs (logical) tables from their (physical) fragments

when applications require it, optimizing performance in the process. Thus, an application accessing the EMPLOYEES table will continue to work properly after the table is fragmented on two modes.

6: Replication Independence
Users should be able to behave-from a logical standpoint-as if the data were not, fact, replicated at all.

Given the highly dynamic nature of database access patterns, even fragmentation must not be sufficient in certain circumstances, particularly from a performance perspective. For, example, what if the two nodes of the software project are on an international WAN, and there is heavy interaction between them that necessitated frequent access to each other's data? Chances are that performance will not always be acceptable.

For such cases, a DDBMS must support replication of data objects (or fragments thereof) on multiple nodes. For example, the EMPLOYEES table could be physically replicated on each of the two nodes. As with fragmentation the DBMS must insulate applications from replication details, if it is to achieve objective 6.
The DBMS catalogs and keeps track of the replicas and their distribution; transparently makes the proper ones available to users; and propagates updates through replicas to keep them in sync.

7: Distributed Query Processing.
The system should support distributed queries.
This too is a rather obvious objective, as one of the main purposes of distributed DBMS is to overcome the lack of support for multi-database operations in client/server environments.

Networked database environments are much more sensitive to performance than non-distributed ones. Optimization requires taking network transmission costs into account. Full support of distributed (or global) optimization by the DDBMS, a clear case where multiple local DBMSs must cooperate.

8: Distributed Transaction Processing.
The system should support distributed transactions.
Other than individual operations, and retrievals, a DDBMS must also support-distributed transactions, that is, sets of multiple operations spanning multiple nodes. As in the non-distributed case, concurrency control and recovery are critical functions. The two-phase commit (2PC) recovery feature comes in handy.

9: Hardware Independence.
Users should be able to run the same DBMS on different hardware systems, and have those systems all participate as equal partners in a distributed system.

10: Software Independence.
Users should be able to run the same DBMS under different operating systems, even on the same hardware.

11. Network Independence.
Users should be able to run the same DBMS on different communication networks.

12: DBMS Independence
Users should be able to run different DBMSs, and have them participate as equal partners in the distributed system.

7.0 DATABASE RECOVERY MANAGEMENT

Recovery restores a database from a given state usually inconsistent to a previously consistent state recovery techniques are based on the atomic transaction property.

All portions of the transaction must be treated as a single logical unit of work in which all operations must be applied and completed to produce a consistent database.
If for any reason any transaction operation cannot be completed the transaction must be aborted and any changes to the database must be rolled back.

Recovery techniques also apply to the database or the system after some type of critical error has occurred.
Backup and recovery functions constitute a very important component of today's DBMS's

Levels of Backups

1. A full back up of the database of dump of the database.
2. Reference backup of the database in which only the last modifications done on the database are copied.
3. A back-up of the transaction log only this level backup all the transaction log operations that are not reflected in the previous back-up copy of the database. The database backup is stored in a secure place usually in a different building and protected against dangers such as fire, theft flood and other potential calamities back-up existence guarantees recovery system (hardware/software) failures. Failures that claim databases and systems are generally induced by software, hardware, program exemption, transactions and external factors.

7.1 Causes Of Database Failures

1. Software - Software induced failures may be traceable to the O.S, DBMS, S/W application programs or viruses.
2. Hardware - Hardware induced failures may include memory chip errors, disk crashes, bad disk sectors, disk full error etc
3. Programming exemptions - Application programs end-users may roll back transactions when certain conditions are defined e.g. a recovery procedure may be initiated if withdrawal funds is made when customer funds are at O or when en-user has initiated an unintended keyboard error such as pressing Ctrl c, the system detects deadlocks and aborts one of the transactions.
4. External factors - Backups are especially important when a system suffers complete distraction due to fire, earthquakes, floods etc. The database recovery process generally follows a predictable scenario where first you determine the type and the extent of the required recovery.

7.2 Transaction And Recovery

It is the role of recovery manager to guarantee at least durability and atomicity in the presence of unpredictable failures.
Log file.

All operations on the database carried out by all transactions are recorded in the log file. In a distributed database, each site may have its own separate log.

An entry is made in the local log file each time the following commands are issued by a transaction:-

- Begin transaction
- Write (insert, delete, update)
- Commit transaction
- Abort transaction

Each log record contains:

1. Transaction identifier
2. The type of log record i.e. as listed above
3. Identifier of data object - affected
4. Before - image of the data object
5. Log management information

Check Pointing
The recovery manager periodically check points (dumps) and on recovery it only has to go back as far as the last check point)

7.2.1 Causes Of Failures

1. Ways:
   - Transaction induced abort e.g. insufficient memory space-time slice.
   - Unforeseen transaction failure arising from bugs.
   - System induced aborts e.g. when transaction manager explicitly aborts a transaction causes it to conflicts with another transaction or to break a deadlock.

2. Site failures - This occur due to failure of the local C.P.U or power supply and results in a system crash, its of 2 types:
   - Total failure - all sites in a distributed database system are down
   - Partial failure - Only some sites are down

3. Medium failure - Commonly caused by disk head crash.
4. Network failure - Although most networks are reliable, a failure may occur in the communication lines.

7.3 Recovery Protocols

1. Restart Procedures - It assumes that no transactions are accepted until the database has been repaired and included.

   (i) Emergency restart this follows when a system fails without warning e.g. due to power failure.
   (ii) Cold restart - The system is restarted from archive when the log and restart file has been corrupted.
   (iii)Warm restart - It allows controlled shut down of the system

2. Archiving - Creation of periodic back-ups.
3. Mirroring - 2 complete copies of the database are maintained online in different stable storage devices. It is used in environments with non-stop fault tolerant operations
4. Undo/Redo - It is undoing and re-doing a transaction after failure. The transaction manager keeps an active list and abort list and a commit list. These comprise of transactions that have begun, abort and committed transactions respectively.
5. 2-Phase Commit - It is used in a DBS and has 2 phases:
   (i) Voting phase - where participants are asked whether or not they are prepared to commit the transaction.
   (i) Decision phase where the transaction is committed.

6. 3-Phase Commit - This one has:
   (i) Voting Phase
   (ii) Pre-commit phase and
   (iii)Decision phase

Salvation Program
When all other recovery techniques fail, a salvation program may be used. This is a specially designed program that scans the database after failure to assess the damages and to restore a valid state by rescuing whatever data are recognizable.

NB: Different recovery techniques (protocols) maintain different kinds of recovery data and are effective if and only if their recovery data have not also been contaminated or destroyed by the failure.

8.0 SECURITY, INTEGRITY AND CONTROL

This is the protection of data from accidental or deliberate threats, which might cause unauthorized modification disclosure or destruction of data and the protection of the Information System from the degradation of non-availability of services.

Data integrity in this context of security is when data are the same as in source documents and have not been accidentally or intentionally altered, destroyed or disclosed.

Misuse of the database can be categorized as being either intentional (malicious) or accidental.

8.1 Objectives of IS security

To control loss of assets
To insure the integrity and reliability of data
To improve the efficiency/effectiveness of Information systems

Risks:
These are various dangers to information systems, the people, hardware, software, data and other assets with which they are associated.
The dangers include:
Natural disasters, thieves, industrial spies, disgruntled employees.
There Risk means the potential loss to the firm.

Threats:
Refer to people, actions, events or other situations that could trigger losses, they are potential causes of loss.

Vulnerabilities: Flaws, problems or other conditions that make a system open/prone to threats.

Common Controls
Controls are counter measures to threats. They are tools that are used to counter risks from the variety of people, actions, events or situations that can threaten an IS.
Are used to identify risk, prevent risk, reduce risks and recover from actual losses.

Physical Controls
These are controls that use conventional physical protection measures.
Might include door locks, keyboard locks, fire doors, surp pumps.
Control over access and use of computer facilities and equipment and controls for prevention of theft.
Inclusive controls to reduce contain or eliminate the damage from natural disasters, power outages, humidity, dust, high temperature and other conventional threats.

Electronic Controls
Are controls that use electronic measures to prevent or identify the threats.
Might include intruder detection and biological access compels e.g. log-on ID, passwords, badges and hand, voice or retina print access controls.

Software Controls
Are program code and controls used in IS applications to prevent, identify or recover from errors, un-authorized access and other threats.
e.g. programming code placed in payroll application to prevent a data entry clerk from entering hourly rate of pay that is too high.

Management Controls
Result from setting, implementing and enforcing policies and procedures e.g. employees required to back up or archive their data at regular interval and take backup copies of data files to secure, off-site locations for storage.

Common Threats
Natural disasters, unauthorized access (e.g. theft, vandalism, invasion of privacy), computer crime and computer viruses.

Natural disasters
E.g. five, floods, water damage, earthquakes, tornadoes, hurricanes, mud slides, wind and storm damage
Security planning should consider

- Disaster prevention
- Disaster containment
- Disaster recovery

e.g. **Prevention**: Use of backup power supplies or special building materials, locations, drainage system or structural modifications to avoid damage during floods, storms fires and earthquakes.
Containment: Consider sprinkler systems, halon gas fire
Suppression: System or watertight ceilings to contain water damage from fire hoses.
Recovery: developing contingency plans for use of computer facilities of vendors or non-competitors with similar computer systems

Employee errors
Ordinary carelessness or poor employee training e.g. formatting the hard disk rather than drive A, keying incorrect data.

Computer crime, fraud and abuse
Computer crime: stealing data, damaging or vandalizing hard ware, software or data or using computer software illegally or committing fraud.

Industrial espionage
It's the theft of original data by competitors. Also called economic espionage
Hacking
Also known as cracking. It's the unauthorized entry by a person into a computer system or network.
Hackers are people who illegally gain access to the computer systems of others.

They can insert viruses onto networks, steal data and software, damage data or vandalize a system.
Toll Fraud
Swindling companies and organisations e.g. through telephone bills through false pretences – e.g. use of slugs instead of real coins
Toll hackers use maintenance ports, modem pools, voice mail systems, automated attendants or other facilities of PBX, the private branch exchanges that are the computerized telephone switches at customer sites.

Signs of frauds:

1. Numerous short calls
2. Simultaneous use of one telephone access mode
3. Numerous calls after business hours
4. Large increases in direct inward system access dialing or DISA

Data diddling
Use of a computer system by employees to forge documents or change data in records for gain.

Trojan horses and salami slicing
This is a change in code that is made to a program without authorization.
It appears to be performing a proper task but may actually perform a variety of mischievous or criminal activities e.g. printing paychecks to employees or vendors who don't exist.

Trap doors
Are procedures or code that allows a person to avoid the usual security procedures for use of or access to a system or data.

Computer viruses
A computer virus is a hidden program that inserts itself into your computer system and forces the system to clone the virus (i.e. it replicates itself.)
They may cause serious damage by modifying data, erasing files or formatting disks.
e.g. cruise or stealth virus might lie dormant until it can capture financial information and transmit the data to thieves
Antivirus programs or vaccination products can be used.
Antivirus programs help in:

- Preventing the virus program inserting itself in your system
- Detecting a virus program so you can take emergency action
- Controlling the damage virus can do once they have been detected.

Hardware theft and vandalism

Software privacy – any reproduction or a copyright program is theft.

Privacy violations
Privacy is the capacity of individuals or organizations to control information about themselves.
Privacy rights imply:

- Type and amounts of data that may be collected about individuals or organizations are limited.
- That individuals and organizations have the ability to access, examine and correct the data stored about them.
- Disclosure, use or dissemination of those data is restricted.

Program bugs
Bugs are defects in programming code. They are prevalent to new software, are normally discovered by users, and software vendors provide "patches" to their code.

8.2 Accidental Loss

Accidental loss of data consistency may result from:-
- Crashes during transaction processing
- Anomalies caused by concurrent access to the database
- Anomalies caused by the distribution of data over several computers
- Logical errors that violate the assumption that transactions preserve the database consistency constraints.

8.3 Malicious Damage

Among the forms of malicious access are the following: -
- Unauthorized reading of data (theft of data)
- Unauthorized modification of data
- Unauthorized destruction of data

Absolute protection of the database from malicious abuse is not possible, but the cost to the perpetrator can be made significantly high to deter most if not all attempts to access the database without proper authority.

8.4 Protection.

To protect the database, measures must be taken at several levels:-

Physical. The site or sites containing the computer systems must be physically secured against armed or surreptitious entry by intruders.

Human. Users must be authorized carefully to reduce the chance of any such user giving access to an intruder in exchange of a bribe or other favors.

Operating system. No matter how secure the database is, weakness in operating system security may serve as a means of unauthorized access to the database.

Network. Since almost all database systems allow remote access through terminals or networks, software level security may serve as a means of unauthorized access to the database.

Database. Some database system users may be authorized to access only a limited portion of the database. Others may be allowed to issue queries, but may be forbidden to modify the data. It is the responsibility of the database system to ensure that these authorization restrictions are not violated.

8.5 Authorization

A user may have several forms of authorization on parts of the database. Among them are the following: -

Read authorization. Allows reading, but no modification, of data.
Insert authorization. Allows insertion of new data, but no modification of existing data.
Update authorization. Allows modification, but not deletion of data.
Delete authorization. Allows deletion of data.

A user may be assigned all, none or a combination of these types of authorization. In addition to these forms of authorization for access to data, a user may be granted authorization to modify the database schema: -

Index authorization. Allows the creation and deletion of indices.
Resource authorization. Allows the creation of new relations.
Alteration authorization. Allows the addition or deletion of attributes in a relation.
Drop authorization. Allows the deletion of relations.

Exercise

Make a list of security concerns for a bank. For each item on your list, state whether this concern relates to physical security, human security, operating system security or database security.

System Integrity

This refers to the system operation conforming to the design specifications despite attempts to make it behave incorrectly.

9.0 QUERY OPTIMIZATION.

Optimization represents both a challege and opportunity for relational systems. A challege because optimization is required in such a system is to achieve acceptable performance and an opportunity because it's one of the strenghs of the relational approach. The advantage of system-managed optimization is not just that users donot have to worry about how nest to state their queries but the real possibility that the optimizer might actually do better than a human programmer.

Reasons for this state of affairs.

- A good optimizer will have a wealth of information available to it that a human programmers typically do not have. It will have certain statistical information e.g. cardinality of each domain, cardinality of each base relation, the number of times each distinct value occurs in each such attribute.
- Reoptimization is possible if the database statistics change significantly e.g. if the database is phasically reorganized.
- The optimizer is a program and is therefore by definition much more patient than a typical human programmer.
- The optimizer is regarded in a as embodyingthe skills and services of 'the best' human programmers.

9.1 The optimization process.

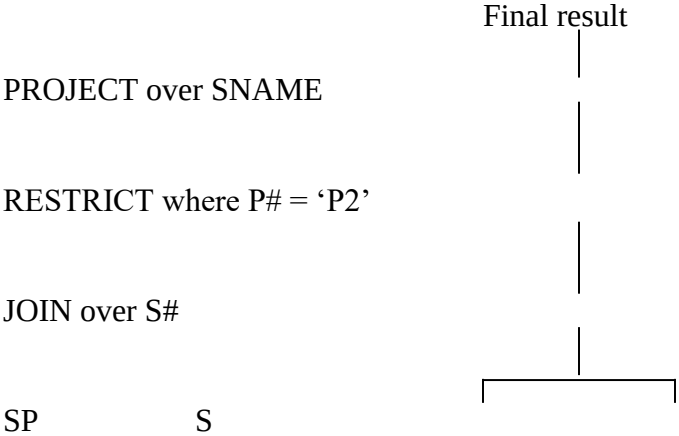
Stages

- Cast the query into some internal form.
- Convert into canonical form.
- Choose candidate low level procedures.
- Generate query plans and choose the cheapest.

Stage 1 : Cast the query into some internal form.

One formalism empoyed in the internal representation is the relational algebra or the relational calculus. The internal form that is ussually chosen is some kind of **abstruct syntax tree** or **query tree**.

Example. ((SP JOIN S) WHERE P# = 'P2') [SNAME] stands for 'get the names of suppliers who supply part P2.



Query tree for 'Names of suppliers who supply part P2'

Stage 2 : Convert to canonical form.
The optimizer performs a number of optimizations that are 'guaranteed to be good' regardless of the data values and access paths that exist in the stored database i.e. relational database allow all but the simplest of the queries to be expressed in a variety of ways that are atleast superficially distinct.

The internal representation is converted into some equivalent cananocal form with the objective of eliminating superficial distinctions and finding a representation that is more efficient than the original in some respect.

In order to transform the output in stage one into some equivalent but more efficient form, the optimizer makes use of well defined **transformation rules** or **laws.**

Example
The expression

(A JOIN B) WHERE restriction-on-A

can be transformed into the equivalent but more efficient expression

(A WHERE restriction-on-A) JOIN B

Stage 3 : Choose candidate low level procedures.
The optimizer decides how to execute the transformed query represented by the converted form. At this stage consideration such as the existence of indexes or other access paths, distribution of stored data values, physical clustering of stored data e.t.c. come into play.
The strategy is to consinder the query expression as specifying a series of **'low level' operations** (join, restriction e.t.c.)

For each low level operation, the optimizer will have available to it a set of predifined **implementation procedures**. Each procedure will have a (parameterized) **cost formula** associated with it, indicating the cost (in terms of disk I/O's, processor utilization). The cost formulas are used in stage 4.
Using the information from the catalogue regarding the current state in the database the optimizer will choose one or more candidate procedures for implementing each of the low level operations in the query expressions. The process is sometimes referred to as **access path selection.**

Stage 4 : Generate query plans and choose the cheapest.
This final stage involves the construction of a set of candidate **query plans,** followed by the choice of the best (i.e. cheapest) of the plans. The cost of a given plan is the. sum of the costs of the individual procedures that go to make up that plan.

9.2 Relational Algebra.

Consists of a collection of operators (e.g. join), that take relations as their operands and return relations as their result. It consits of eight operators, two groups of four each :

- The traditional set of operations **union**, **intersection**, **difference** and **cartesian** product.
- The special relational operations **restrict**, **project**, **join**,and **divide**.

Restrict : Returns a relation consisting of all tuples from a specified relation that satisfy a specific condition.

Project : Returns a relation consisting of all tuples that remain as (sub)tuples in s specified relation after specified attributes have been eliminated.

Product : Returns a relation consisting of all possible tuples that are a combination of two tuples , one from each of two specified relations.

Union : Returns a relation consisting of all tuples appering in either or both of two specified relations.

Intersect : Returns a relation consisting of all possible tuples appearing in both of two specified relations.

Difference : Returns a relation consisting of all possible tuples appearing in the first and not in the second of the two specified relations.

Join : Returns a relation consisting of all possible tuples that are a combination of two tuples , one from each of two specified relations, such that the two tuples contributing to any given combination have a common value for the common attribute(s) of the two relations (and that common value appears just once, not twice, in the result tuple. This is a natural join.

Divide : Takes two relations, one binary and one unary, and returns a relation consisting of all values of one atrribute of the binary relation that match (in the other attribute) all values in the unary relation.

10.0 LATEST DEVELOPMENTS IN DATABASE TECHNOLOGY.

10.1 Developments

- **Integrating other types of information.**

Images	Records
Graphics	Text
Video	Documents
Rules	

- **Web technology**
- **Geographical information systems (GIS)**

Capturing, storing, checking, integrating, analysing and displaying spartilly referenced data about the earth. It includes topographic maps, remote sensing images, photograhic images, geodetic e.t.c.

- **Knowledge-based databases.**

Databases that support logical rules of inference.

- **Computer aided manufacturing.**
- **Data warehousing.**

A subject oriented integrated, time variant and non volatile collection of data in support of management's decision making process.
Subject oriented : a data warehouse is organised around the major subjects of an organization e.g. customers, products, sales rather than the major application areas e.g. stock control, invoicing.
Integrated : Comming from different sources.
Time variant : Data in the data warehouse is only accurate and valid at some point in time or over a time interval.
Non-volatile : the data is not updated in real time but is refreshed from operational systems from time to time.

- **Data mart :**

A subset of data warehouse that supports the requirements of a particular department or business function.

- **Data mining :**

The process of extracting valid, previusly unknown, comprehensible and actionable information from large databases and using it to make crucial business decisions.

- **Active databases :**

Databases that are based on events, conditions and actions. They are triggered upon the occurrence of certain events in the system. Related to process control systems.

- **Online analytical processing (OLAP) :**

The dynamic sysnthesis, analysis and consolidation of large volumes of multi-dimensional data.

- **Digital publishing.**
- **Computer aided software engineering.**
- **Data Warehouses.**

Major components of a data warehouse.

- **Operating data.**
- **Load manager :** Performs all the operations associated with the extraction and loading of the data in the warehouse.

- **Warehouse manager :** Performs all operations associated with the management of the data in the warehouse e.g. analysis of data to ensure consistency, transforming and merging of data sources.
- **Query manager :** Performs all operations associated with the management of user queries e.g. directing queries to appropriate tables and scheduling execution of queries.
- **End-user access tools :** Data reporting and query tools, application development tools, executive information system tools, OLAP tools, data mining tools.

10.2 Applications

Decision-support Systems

As online availability of data has grown, businesses have began to exploit the available data to make better decisions about their activities, such as what items to stock and how best to target customers to increase sales.

Data analysis

Although complex statistical analysis is best left to statistics packages, databases should support simple, commonly used, forms of data analysis. Since the data in the databases are usually large in volume, they need to be summarized in some fashion if we are to derive information that humans can use. The SQL aggregation functionality is limited; so several extensions have been implemented by different databases. For instance, although SQL defines only a few aggregate functions, many database systems provide a richer set of functions including variance, median, and so on.

Data mining

The term data mining refers loosely to finding relevant information, or "discovering knowledge" from a large volume of data. Like knowledge discovery in artificial intelligence, data mining attempts to discover statistical rules and patterns automatically form data.

Data warehousing

Large companies have presences at numerous sites, each of which may generate a large volume of data. A data warehouse is a repository (or archive) of information gathered from multiple sources, stored under a unified schema at a single site. Once gathered, the data are stored for a long time, permitting access to historical data. Thus data warehouses provide the user a single consolidated interface to data, making decision support queries easier to write.

Spatial and geographical Databases

Spatial databases store information related to spatial locations, and provide support for efficient querying and indexing based on spatial locations. Two types of spatial databases are: -

- Design databases or CAD databases –are spatial databases used to stored design information about how objects such as buildings, car, or aircraft are constructed. Other important CAD databases are integrated circuit and electronic-device layouts.

•Geographical databases are spatial databases used to store graphic information, such as maps. Geographic databases are often called geographic information systems.

Multimedia databases

Recently there interest in databases that store multimedia data, such as images, audio and video. Database functionality becomes important when the number of multimedia objects stored is large. Issues such as transaction updates, querying facilities, and indexing then become important.
Multimedia objects often have descriptive attributes, such as those indicating when they were created, who created them and to what category they belong.

One approach to building a database for such multimedia objects is to use databases the descriptive attributes and for keeping track of files in which the multimedia objects are stored.
Storing multimedia outside the database makes it harder to provide database functionality, such as indexing on the bases of actual multimedia data content. It can also lead to inconsistencies, such as a file that is noted in the database but whose contents are missing and vise versa.

Several issues have to been addressed if multimedia data are to be stored in a database.

(i) The database must support large objects, since multimedia object such as video can occupy up to a few gigabytes of storage. Many relational objects do not support such large objects.
(ii) Similarity-based retrieval is needed in many multimedia database applications. For example in a database that stores fingerprint images, a query fingerprint image is provided and finger prints in the database that are similar to the query finger print must be retrieved.
(iii) The retrieval of some types of data, such as audio and video, has the requirement that data delivery must proceed at a guaranteed steady rate. Such data are sometimes called isochronous data or continuous-media data.

Mobility and personal databases

Large scale, commercial databases have traditionally been stored in central computing facilities. Incase of distributed database applications, there has been a strong central and network administration.

Two technology trends have combined to create applications in which this assumption of central control and administration is not entirely correct:

(i) The increasingly widespread use of personal computers, and, more important of laptop or “notebook” computers.
(ii) The development of a relatively low-cost wireless digital communication infrastructure, based on wireless local-area networks, cellular digital packet networks, and other technologies.

Mobile computing has proved useful in many applications. Many business travelers use lap top computers to enable them to work and access data en route. Delivery services use mobile computers to assist in packet tracking. Emergency response services use mobile computers at the

scene of disasters, medical emergencies, and the like to access information and to enter data pertaining to the situation.

Distributed information systems

It is easy for a person located in one area to connect to computers based in distant geographical area and retrieve information stored on the distant computer. The most widespread distributed information system today is the World Wide Web.

The World Wide Web

The World Wide Web is a distributed information system based on hypertext. The user of a web system sees formatted text along with images, rather than the raw text with formatting instructions.