KABARAK UNIVERSITY

BSCIT, BMIT &BSC TLCM

INTE 214: ADVANCED DATABASE MANAGEMENT SYSTEM

REVISED 4th EDITION(2014)

Relational Data Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Tables are logical structures maintained by the database manager. The relational model is a combination of three components, such as Structural, Integrity, and Manipulative parts.

Structural Part

The structural part defines the database as a collection of relations.

Integrity Part

The database integrity is maintained in the relational model using primary and foreign keys.

Manipulative Part

The relational algebra and relational calculus are the tools used to manipulate data in the database. Thus relational model has a strong mathematical background.

Relational Algebra

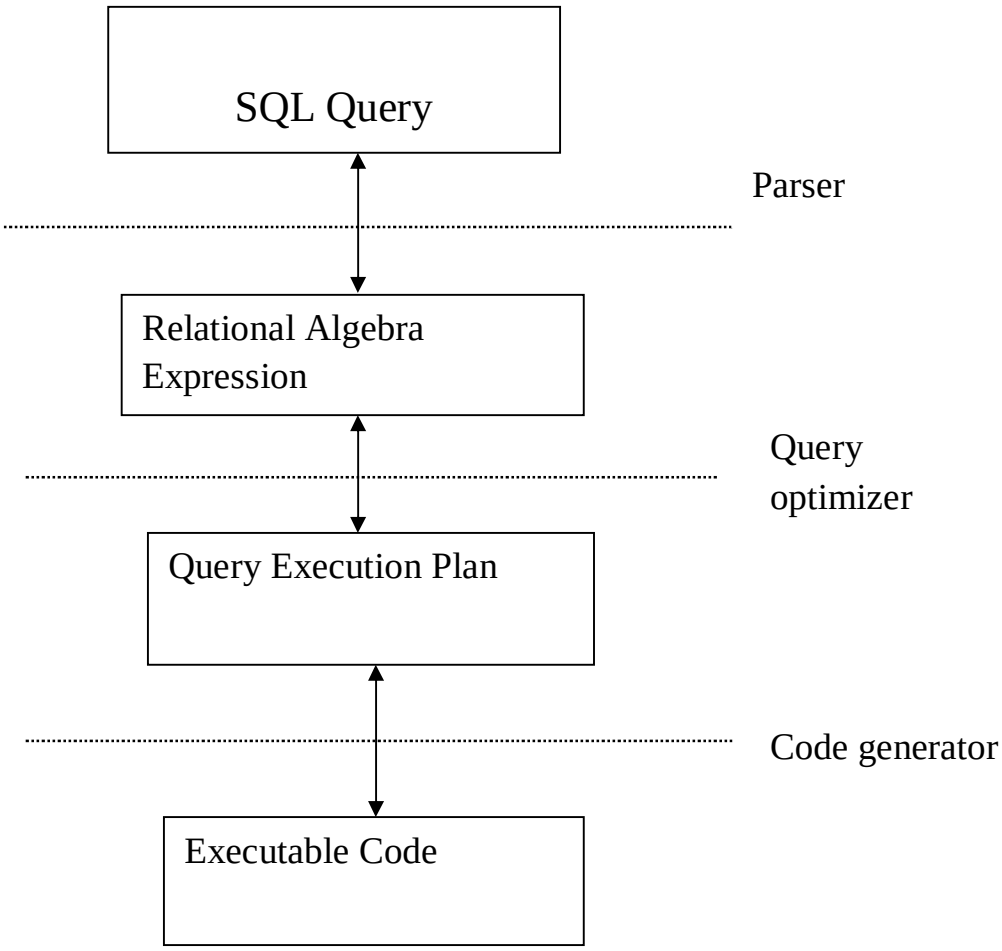
The relational algebra is a theoretical language with operators that work on one or more relations to define another relation without changing the original relation. Thus, both the operands and the results are relations; hence the output from one operation can become the input to another operation.

This allows expressions to be nested in the relational algebra. This property is called closure. Relational algebra is an abstract language, which means that the queries formulated in relational algebra are not intended to be executed on a computer. Relational algebra consists of group of relational operators that can be used to manipulate relations to obtain a desired result.

Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system.

Role of Relational Algebra in DBMS

Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system. From the diagram below when an SQL query has to be converted into an executable code, first it has to be parsed to a valid relational algebraic expression, then there should be a proper query execution plan to speed up the data retrieval. The query execution plan is given by query optimizer. The diagram below is on relational algebra:



Relational Algebra Operators

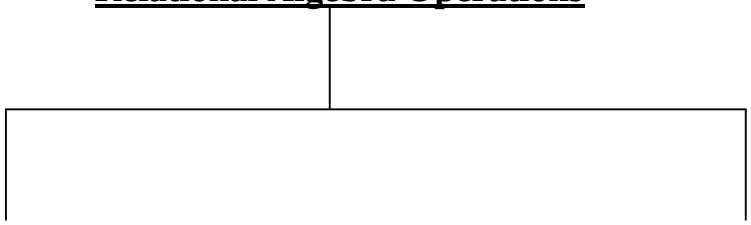
Operators in relational algebra can be broadly classified into set operators and database operators.

Unary and Binary Operations

Unary operation involves one operand, whereas binary operation involves two operands. The selection and projection are unary operations. Union, difference, Cartesian product, and Join operations are binary operations:

- Unary operation operate on one relation
- Binary operation operate on more than one relation

Relational Algebra Operations



Set operations
- Union
- Intersection
- Difference
- Cartesian product

Database operations
- Selection
- Projection
- Join

Three main database operations are SELECTION, PROJECTION, and JOIN.

Selection Operation

The selection operation works on a single relation R and defines a relation that contains only those tuples of R that satisfy the specified condition (Predicate). We use the lowercase Greek letter sigma **(σ)** to denote selection. The predicate appears as a subscript to **σ**. Selection operation can be considered as row wise filtering.

Syntax of Selection Operation

The syntax of selection operation is: **σ** Predicate (R). Here R refers to relation and predicate refers to condition.

Illustration of Selection Operation

To illustrate the SELECTION operator consider the STUDENT relation with the attributes Registration number, Name, and GPA (Grade Point Average).

Question: List the Reg_no, Name, and GPA of those students who are having GPA of above 8.0

Answer: Query expressed in relational algebra as $\sigma_{GPA > 8}$ (Student).

Question: Give the details of first four students in the class.

Answer: Relational algebra expression is $\sigma_{Reg\_no \leq 4}$(student).

NB:

In general, we allow comparisons using **=, ≠, <, ≤, >, ≥** in the selection predicate.
Furthermore, we can combine several predicates into a larger predicate by using the connectives and **(∧)**, or (**∨**), and not (¬).

Projection Operation

The projection operation works on a single relation R and defines a relation that contains a vertical subject of R, extracting the values of specified attributes and elimination duplicates. The projection operation can be considered as column wise filtering. Projection is denoted by the uppercase Greek letter pi **(Π).** We list the attributes that we wish to appear in the result as a subscript to **Π**.

Syntax of Projection Operation

The syntax of projection operation is given by: $\Pi a_1, a_2 \ldots a_n (R)$. Where $a_1, a_2 \ldots a_n$ are attributes and R stands for relation.

Illustration of Projection Operation

To illustrate projection operation consider the relation STAFF, with the attributes Staff number, Name, Gender, Date of birth, and Salary.

Question: Produce the list of salaries for all staff showing only the Name and salary detail.

Answer: Relational algebra expression: $\Pi_{\text{Name, salary}}(\text{staff})$

Question: Give the name and Date of birth of the all the staff in the STAFF relation.

Answer: Relational algebra expression: $\Pi_{\text{Name, date of birth}}(\text{staff})$

Rename operation (ρ)

The rename operator returns an existing relation under a new name. The rename operator is denoted by the lowercase Greek letter rho (**ρ**). For example: **ρ**A (B) is the relation B with its name changed to A. The results of operation in the relational algebra do not have names. It is often useful to name such results for use in further expressions later on. The rename operator can be used to name the result of relational algebra operation.

Union Compatibility

In order to perform the Union, Intersection, and the Difference operations on two relations, the two relations should be union compatible. Two relations are union compatible if they have same number of attributes and belong to the same domain. Mathematically UNION COMPATIBILITY it is given as:

Let R ($A_1, A_2 \ldots A_n$) and S ($B_1, B_2 \ldots B_n$) be the two relations. The relation R has the attributes $A_1, A_2, \ldots\ldots A_n$ and the relation S has the attributes $B_1, B_2, \ldots\ldots\ldots\ldots B_n$. The two relations R and S are union compatible if $dom(A_i)=dom(B_i)$ for i = 1 to n.

Union Operation

The union of two relations R and S defines a relation that contains all the tuples of R or S or both R and S, duplicate tuples being eliminated.

Relational Algebra Expression

The union of two relations R and S are denoted by $R \cup S$.

Given the following relations:

The ***borrower*** relation

customer-name	loan-number
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17
Adams	L-16

The ***depositor*** relation

customer-name	*account-number*
Johnson	A-101
Smith	A-215
Hayes	A-102
Turner	A-305
Johnson	A-201
Jones	A-217
Lindsay	A-222

An example query would be to find the name of all bank customers who have either a deposit account or a loan or both. To find this result we will need the information in the depositor relation and in the borrower relation. To find the names of all customers with a loan in the bank we would write:

πcustomer-name (borrower)

and to find the names of all customers with an account in the bank, we would write:

πcustomer-name (depositor)

Then by using the union operation on these two queries we have the query we need to obtain the wanted results. The final query is written as:

$$\pi\text{customer-name (borrower)} \cup \pi\text{customer-name (depositor)}$$

The result of the query is the following:

customer-name
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Adams

Intersection Operation

The intersection operation defines a relation consisting of the set of all tuples that are in both R and S.

Relational Algebra Expression

The intersection of two relations R and S is denoted by R ∩ S.

An example query of the operation to find all customers who have both a loan and and deposit account can be written as:

$$\pi_{\text{customer-name}}(\text{borrower}) \cap \pi_{\text{customer-name}}(\text{depositor})$$

Difference Operation

The set difference operation defines a relation consisting of the tuples that are in relation R but not in S.

Relational Algebra Expression

The difference between two relations R and S is denoted by R – S. It results in finding tuples that are in one relation but are not in another.

For example, the query to find all customers of the bank who have deposit account but not a loan, is written as:

$$\pi_{\text{customer-name}}(\text{depositor}) - \pi_{\text{customer-name}}(\text{borrower})$$

The result of the query is the following:

customer-name
Johnson
Turner
Lindsay

Cartesian Product Operation

The Cartesian product operation defines a relation that is the concatenation of every tuples of relation R with every tuples of relation S. The result of Cartesian product contains all attributes from both relations R and S.

Relational Algebra Symbol for Cartesian Product:

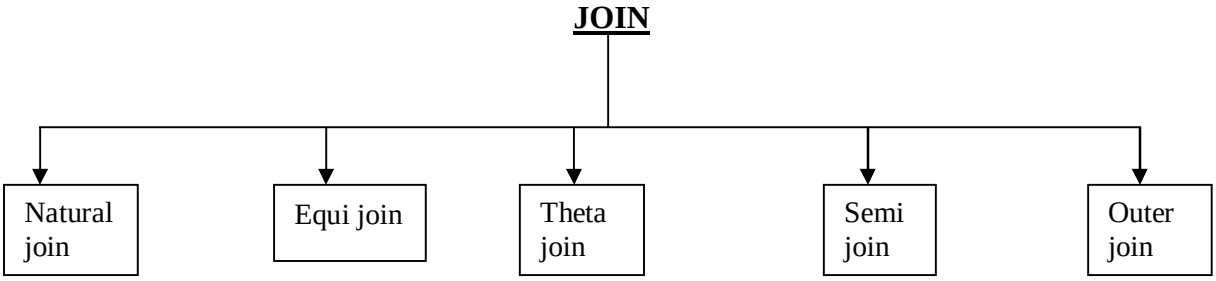
The Cartesian product between the two relations R and S is denoted by
R × S.
Note: If there are n1 tuples in relation R and n2 tuples in S, then the number of tuples in R × S is n1*n2.

Example
If there are 5 tuples in relation "R" and 2 tuples in relation "S" then the number of tuples in R × S is 5 × 2 = 10.

Join Operations

Join operation combines two relations to form a new relation. The tables should be joined based on a common column. The common column should be compatible in terms of domain.

Types of Join Operation



Natural Join

The natural join performs an equi join of the two relations R and S over all common attributes. One occurrence of each common attribute is eliminated from the result. In other words a natural join will remove duplicate attribute. In most systems a natural join will require that the attributes have the same name to identity the attributes to be used in the join. This may require a renaming mechanism. Even if the attributes do not have same name, we can perform the natural join provided that the attributes should be of same domain.
Input: Two relations (tables) R and S
Notation: R ⋈ S
Purpose: Relate rows from second table and
– Enforce equality on all column attributes
– Eliminate one copy of common attribute
* Short hand for π L(R × S):
– L is the union of all attributes from R and S with duplicate removed
– P equates all attributes common to R and S

Equi Join

A special case of condition joins where the condition C contains only equality. There is duplication of the attributes common in both relations.

Theta Join

A conditional join in which we impose condition other than equality condition. If equality condition is imposed then theta join become equi join. The symbol θ stands for the comparison operator which could be **>, <, >=, <=.**

Expression of Theta Join:

$\sigma\theta(R \times S)$

It combines the attributes of both relations.

Outer Join

In outer join, matched pairs are retained unmatched values in other tables are left null.

Types of Outer Join

1) **Left Outer Join.** Left outer joins is a join in which tuples from R that do not have matching values in the common column of S are also included in the result relation.
2) **Right Outer Join**. Right outer join is a join in which tuples from S that do not have matching values in the common column of R are also included in the result relation.
3) **Full Outer Join.** Full outer join is a join in which tuples from R that do not have matching values in the common columns of S still appear and tuples in S that do not have matching values in the common columns of R still appear in the resulting relation.

Semi-Join

The semi-join of a relation R, defined over the set of attributes A, by relation S, defined over the set of attributes B, is the subset of the tuples of R that participate in the join of R with S. The advantage of semi-join is that it decreases the number of tuples that need to be handled to form the join. In centralized database system, this is important because it usually results in a decreased number of secondary storage accesses by making better use of the memory. It is even more important in distributed databases, since it usually reduces the amount of data that needs to be transmitted between sites in order to evaluate a query.

Expression for Semi-Join

$R\alpha_F S = \Pi A(R \alpha_F S)$ where F is the predicate.

Example of Semi-Join

In order to understand semi-join consider two relations EMPLOYEE and PAY. The semi-join of EMPLOYEE with the PAY is denoted by:

EMPLOYEE $\alpha_{\text{EMPLOYE.DESIGNATION=PAY.DESIGNATION}}$ PAY.

Limitations of Relational Algebra

- The relational algebra cannot do arithmetic. For example, if we want to know the price of 10 l of petrol, by assuming a 10% increase in the price of the petrol, which cannot be done using relational algebra.
- The relational algebra cannot sort or print results in various formats. For example we want to arrange the product name in the increasing order of their price. It cannot be done using relational algebra.
- Relational algebra cannot perform aggregates. For example we want to know how many staff are working in a particular department. This query cannot be performed using relational algebra.
- The relational algebra cannot modify the database. For example we want to increase the salary of all employees by 10%. This cannot be done using relational algebra.

- The relational algebra cannot compute "transitive closure." In order to understand the term transitive closure, consider the relation RELATIONSHIP, which describes the relationship between persons.

Relational Calculus

The purpose of relational calculus is to provide a formal basis for defining declarative query languages appropriate for relational databases. Relational Calculus comes in two flavors (1) Tuple Relational Calculus (TRC) and (2) Domain Relational Calculus (DRC). The basic difference between relational algebra and relational calculus is that the former gives the procedure of how to evaluate the query whereas the latter gives only the query without giving the procedure of how to evaluate the query:

- The variable in tuple relational calculus formulae range over tuples.
- The variable in domain relational calculus formulae range over individual values in the domains of the attributes of the relations.

Relational calculus is nonoperational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)

Relational Calculus and Relational Algebra:

The major difference between relational calculus and relational algebra is summarized later:

- A relational calculus query specifies ***what*** information is retrieved
- A relational algebra query specifies ***how*** information is retrieved

Tuple Relational Calculus

Tuple relational calculus is a logical language with variables ranging over tuples. The general form of tuple relational calculus is given by:

{<tuple variable list> | <conditions>}

{t | COND(t)}

Here t is the tuple variable, which stands for tuples of relation. COND (t) is a formula that describes t. The meaning of the earlier expression is to return all tuples T that satisfy the condition COND:

- {T/R (T)} means return all tuples T such that T is a tuple in relation R.
- For example, {T.name/FACULTY (T)} means return all the names of faculty in the relation FACULTY.
- {T.name/ FACULTY (T) AND T.deptid='EEE'} means return the value of the name of the faculty who are working in EEE department.

Quantifiers

Quantifiers are words that refer to quantities such as "some" or "all" and tell for how many elements a given predicate is true. A predicate is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables. Quantifiers can be broadly classified into two types (1) Universal Quantifier and (2) Existential Quantifier.

Existential Quantifier

Symbol: ∃

∃T ε Cond (R)

It will succeed if the condition succeeds for at least one tuple in T.

(∃)(C) – Existential operator – True if there exists a tuple t such that the condition(s) C are true.

Example of existential quantifier is ∃(m) such that m2 =m.(i.e., m=1).

Universal Quantifier
Symbol: ∀
– (∀t) (C) – Universal operator – True if C is true for every tuple t.
– Example of universal quantifier is ∀(2), sin2 (2) + cos2 (2) = 1.
The example refers to the fact that for all values of 2 sin2 (2) + cos2 (2) = 1.

Domain Relational Calculus (DRC)

Domain relational calculus is a nonprocedural query language equivalent in power to tuple relational calculus. In domain relational calculus each query is an expression of the form:

$\{<X_1,X_2,.....,X_n>/P(X_1,X_2,.....,X_n)\}$ where

– $X_1,X_2,.....,X_n$ represent domain variables
– P represents a formula similar to that of the predicate calculus.
Domain variable: A domain variable is a variable whose value is drawn from the domain of an attribute.

Transaction Processing and Query Optimization

Transaction Processing

Introduction

Managing Data is the critical role in each and every organization. To achieve the hike in business they need to manage data efficiently. DBMS provides a better environment to store and retrieve the data in an economical and efficient manner. User can store and retrieve data through various sets of instructions. These sets of instructions do several read and write operations on database. These processes are denoted by a special term "Transaction" in DBMS.
Transaction is the execution of user program in DBMS. It is different from the execution of the program external to DBMS. In other words it can be stated as the various read and write operations done by the user program on the DBMS, when it is executed in DBMS environment.

Transaction Management plays a crucial role in DBMS. It is responsible for the efficiency and consistency of the DBMS. Partial transaction let the database in an inconsistency state, so they should be avoided.

Key Notations in Transaction Management

The key notations in transaction management are as follows:

- **i.** Object. The smallest Data item which is read or updated by the Transaction is called as Object in this case.
- **ii.** Transaction. Transaction is represented by the symbol T. It is termed as the execution of query in DBMS.
- **iii.** Read Operation. Read operation on particular object is notated by symbol R (object-name).
- **iv.** Write Operation. Write operation on particular object is notated by symbol W (object-name).
- **v.** Commit. This term used to denote the successful completion of one Transaction.
- **vi.** Abort. This term used to denote the unsuccessful interrupted Transaction.

Concept of Transaction Management

User program executed by DBMS may claim several transactions. In the web environment, there is a possibility to several users' attempt to access the data stored in same database. To maintain the accuracy and the consistency of the database several scheduling algorithms are used.

To improve the effective throughput of the DBMS we need to enforce certain concurrent executions in DBMS. Transaction Manager is responsible for scheduling the Transactions and providing the safest path to complete the task. To maintain the data in the phase of concurrent access and system failure, DBMS need to ensure four important properties. These properties are called as ACID properties.

ACID Properties of DBMS

ACID is an acronym for Atomicity, Consistency, Isolation, and Durability.
A – Atomicity
C – Consistency
I – Isolation
D – Durability
Atomicity and Durability are closely related.
Consistency and Isolation are closely related.

Atomicity and Durability

Atomicity

Either all Transactions are carried out or none are. The meaning is the transaction cannot be subdivided, and hence, it must be processed in its entirety or not at all. Users should not have to worry about the effect of incomplete Transactions in case of any system crash occurs.
Transactions can be incomplete for three kinds of reasons:

- **i.** Transaction can be aborted, or terminated unsuccessfully. This happens due to some anomalies arises during execution. If a transaction is aborted by the DBMS for some internal reason, it is automatically restarted and executed as new.
- **ii.** Due to system crash. This may be happen due to Power Supply failure while one or more Transactions in execution.
- **iii.** Due to unexpected situations. This may be happen due to unexpected data value or be unable to access some disk. So the transaction will decide to abort. (Self termination).

Durability

If the System crashes before the changes made by a completed Transaction are written to disk, then it should be remembered and restored during the system restart phase.

Partial Transaction

If the Transaction is interrupted in the middle way it leaves the database in the inconsistency state. These types of transactions are called as Partial Transactions.

Partial Transactions should be avoided to gain consistency of database.
To undo the operations done by the Partial Transactions DBMS maintains certain log files. Each and every moment of disk writes are recorded in this log files before they are reflected to disk. These are used to undo the operations done when the system failure occurs.

Consistency and Isolation

Consistency

Users are responsible for ensuring transaction consistency. User who submits the transaction should make sure the transaction will leave the database in a consistent state.

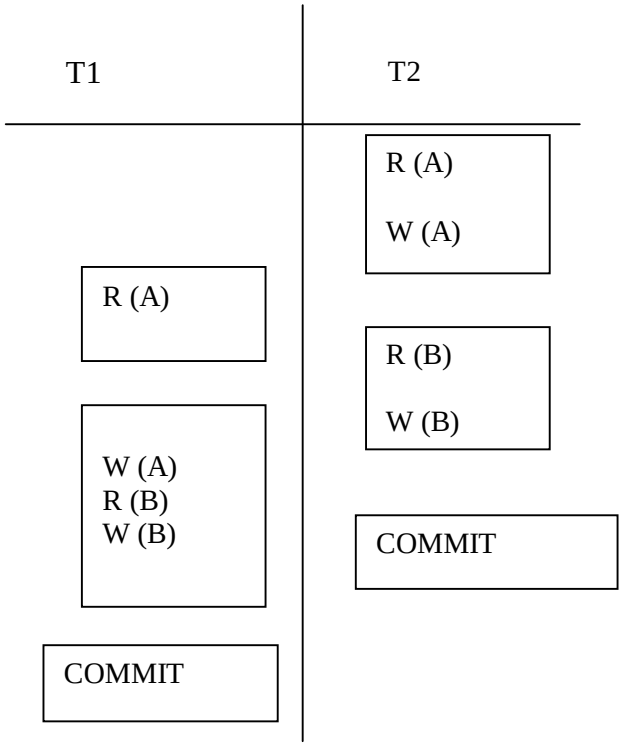
Example
If the transaction of money between two accounts "A" and "B" is manually done by the user, then first thing he has to do is, he deducts the amount (say Ksh.100) from the account "A" and add it with the account "B." DBMS do not know whether the user subtracted the exact amount from account "B." User has to do it correctly. If the user subtracted Ksh. 99 from account "B" instead of Ksh.100 DBMS is not responsible for that. This will leave DB in inconsistency state.

Isolation
In DBMS system, there are many transaction may be executed simultaneously.
These transactions should be isolated to each other. One's execution should not affect the execution of the other transactions. To enforce this concept DBMS has to maintain certain scheduling algorithms. One of the scheduling algorithms used is Serial Scheduling.

Serial Scheduling
In this scheduling method, transactions are executed one by one from the start to finish. An important technique used in this serial scheduling is interleaved execution.

Interleaved Execution
In DBMS to enforce concurrent Transactions, several Transactions are ordered in a serial manner and executed on by one according to the schedule. So there will be the switching over of execution between the Transactions. This is called as Interleaved Execution.

Example
The example for the serial schedule is illustrated below:



Explanation
In the above example, two Transactions T1, T2 and two Objects A, B are taken into account. Commit denotes successful completion of both Transactions.
First one read and one write operation are done on the object A by Transaction T2. This is followed by T1. It does one write operation on object A.
The same procedure followed by other transactions. Finally both Transactions are ended successfully.

Anomalies due to Interleaved Transactions

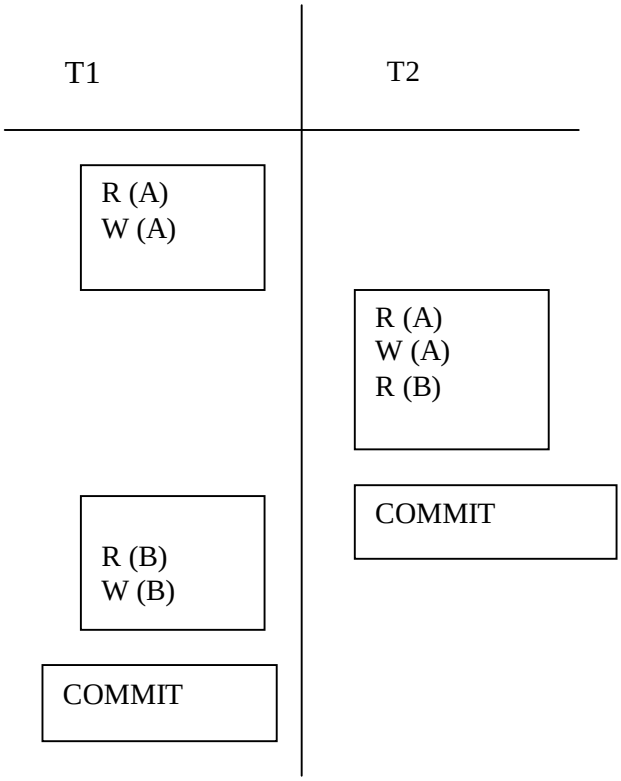
If all the transactions in DBMS systems are doing read operation on the Database then no problem will arise. When the read and write operations done alternatively there is a possibility of some type of anomalies. These are classified into three categories.

- Write–Read Conflicts (WR Conflicts)
- Read–Write Conflicts (RW Conflicts)
- Write–Write Conflicts (WW Conflicts)

WR Conflicts

Dirty Read
This happens when the Transaction T2 is trying to read the object A that has been modified by another Transaction T1, which has not yet completed (committed). This type read is called as dirty read.

Example
Consider two Transactions T1 and T2, each of which, run alone, preserves database consistency. T1 transfers Ksh. 200 from A to B, and T2 increments both A and B by 6%.
If the Transaction is scheduled as shown below:



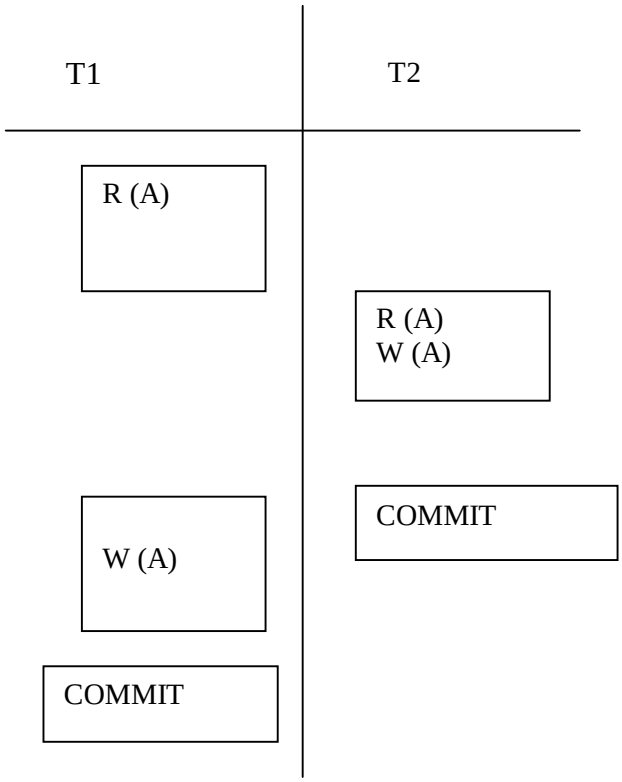
Reading uncommitted data

Explanation
Suppose if the transactions are interleaved according to the above schedule then the account transfer program T1 deducts Ksh.100 from account A, then the interest deposit program T2 reads the current values of accounts A and B and adds 6% interest to each, and then the account transfer program credits Ksh.100 to account B. The outcome of this execution will be different from the normal execution like if the two instructions are executed one by one. This type of anomalies leaves the database in inconsistency state.

RW Conflicts

Unrepeatable Read
In this case anomalous behavior could result is that a Transaction T2 could change the value of an object A that has been read by a Transaction T1, while T2 is still in progress. If T1 tries to read A again it will get different results. This type of read is called as Unrepeatable Read.

Example
If 'A' denotes an account. Consider two Transactions T1 and T2. Duty of T1 and T2 are reducing account A by $100. Consider the following Serial Schedule:



Explanation

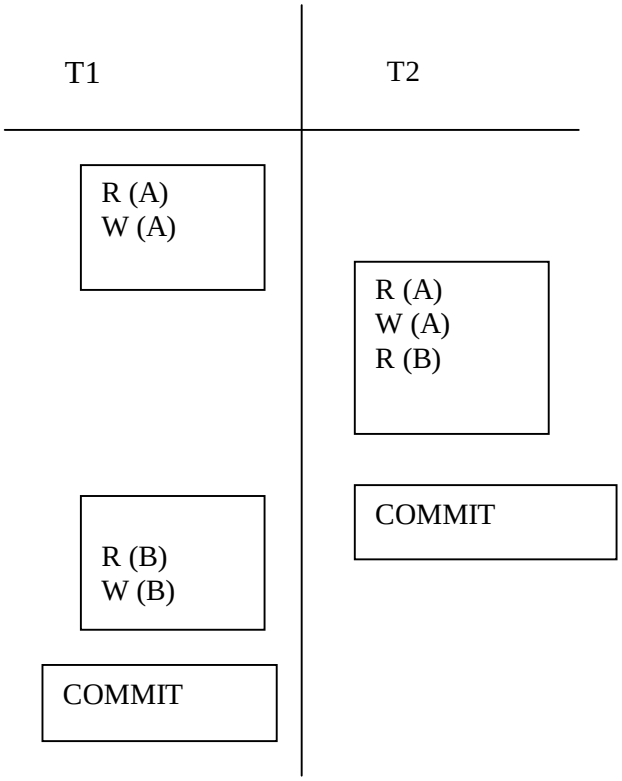
At first, T1 checks whether the money in account A is more than Ksh.100. Immediately it is interleaved and T2 also checks the account for money and reduce it by Ksh.100. After T2, T1 is tried to reduce the same account A. If the initial amount in A is Ksh.101 then, after the execution of T2 only $1 will remain in account A. Now T1 will try to reduce it by Ksh. 100. This makes the Database inconsistent.

WW Conflicts

The third type of anomalous behavior is that one Transaction is updating an object while another one is also in progress.

Example

Consider the two Transactions T1, T2.
Consider the two objects A, B.
Consider the following Serial Schedule as illustrated below:



Explanation

If A and B are two accounts and their values have to be kept equal always, Transaction T1 updates both objects to 3,000 and T2 updates both objects to 2,000. At first T1 updates the value of object A to 3,000. Immediately T2 makes A as 2,000 and B as 2,000 and committed.
After the completion of T2, T1 updates B to 3,000. Now the value of A is 2,000 and value of B is 3,000, they are not equal. Constraint violated in this case due to serial scheduling.

Durability

Durable means the changes are permanent. Once a transaction is committed, no subsequent failure of the database can reverse the effect of the transaction.

Lock-Based Concurrency Control

Concurrency Control is the control on the Database and Transactions which are executed concurrently to ensure that each Transaction completed healthy.
Concurrency control is concerned with preventing loss of data integrity due to interference between users in a multiuser environment.

Need for Concurrency Control

In database management system several transactions are executed simultaneously.
In order to achieve concurrent transactions, the technique of interleaved execution is used. But in this technique there is a possibility for the occurrence of certain anomalies, due to the overriding of one transaction on the particular Database Object which is already referred by another Transaction.

Lock-Based Concurrency Control

It is the best method to control the concurrent access to the Database Objects by providing suitable permissions to the Transactions. Also it is the only method which takes less cost of Time and less program complexity in terms of code development.

Key Terms in Lock-Based Concurrency Control

1) Database Object

Database Object is the small data element, the value of which one is altered during the execution of transactions.

2) Lock

Lock is a small object associated with Database Object which gives the information about the type of operations allowed on a particular Database Object. Lock can be termed as a type of permission provided by the transaction manager to the transactions to do a particular operation on a Database Object.
The transaction must get this permission from Transaction Manager to access any Database Object for alteration. Locking mechanisms are the most common type of concurrency control mechanism. With locking, any data that is retrieved by a user for updating must be locked, or denied to other users, until the update is complete.

Locking Protocol

It is the set of rules to be followed by each transaction, to ensure that the net effect of execution of each Transaction in interleaved fashion will be same as, the result obtained when the Transactions executed in serial fashion. Generally locks can be classified into two.
The two types of Lock are:

i. Strict Two-Phase Locking (Strict 2PL)
ii. Deadlock

Strict Two-Phase Locking (Strict 2PL)

It is a most widely used locking protocol. It provides few rules to the Transactions to access the Database Objects. They are:
Rule 1:
If a Transaction "T" wants to read, modify an object, it first requests a shared, exclusive lock on the Database Object respectively.
Rule 2:
All Locks held by the Transaction will be released when it is completed.

Shared Lock: It is type of lock established on a Database Object. It is like a component which is sharable within all active transactions. A Database Object can be shared locked by more than one number of transactions. To get a shared lock on particular Database Object the Database Object should satisfy the following condition.
Condition: It should not be exclusively locked by any of the other Transactions.

Example
If a person updates his bank account then the Database will lock that Database Object exclusively to avoid RW conflicts. So the Transactions which will be requesting to read that Database Object will be suspended until the completion of updating.

Exclusive Lock: It is type of lock established on a Database Object. It is like a component which cannot be shared within all active Transactions. It is dedicated to particular transaction; only that particular transaction can access and modify that object.
Condition: It should not be exclusively locked by any one of the other Transactions.

Example
Assume the situation in reservation of Bus tickets in KPN Travels agencies.
Assume that the number of tickets remaining in bus no. 664 V is only one. Two persons who are in different places are trying to reserve that particular ticket. See the situation here that only one of them should be allowed to access the Database while the other should wait until previous one is completed. Otherwise one terminal will check the number of seats available and due to interleaved actions next terminal will do the same and finally both of them will try to modify the Database (Decrement the seats available) this leads to more anomalies in Database and finally Database will be left into inconsistent state.

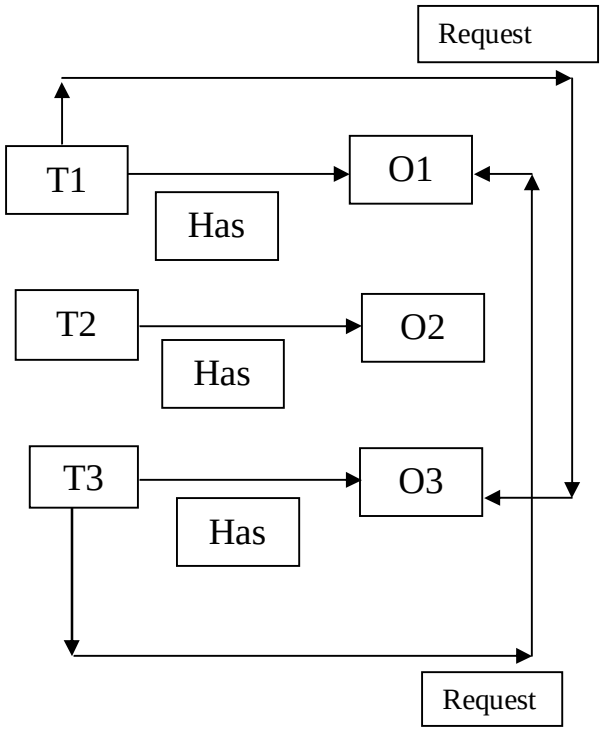
Deadlock

Deadlock occurs within the Transactions in DBMS system. Due to this neither one will be committed. This is the dead end to the execution of transactions.
DBMS has to use suitable recovery systems to overcome Deadlocks.

Reasons for Deadlock Occurrence:

Deadlock occurs mainly due to the Lock-Based Concurrency Control. The exclusive lock type will isolate one particular Database Object from the access of other transactions. This will suspend all the transactions who request Shared lock on that particular Database Object until the transaction which holds Exclusive lock on that object is completed. This will create a loop in Database which leads to Deadlock within transactions. This will leave the Database in inconsistent state.

Example
Assume, Transactions T1, T2, T3 as illustrated below. Database Objects are O1, O2, O3.



Deadlock

Explanation
Here we can observe that the loop occurs between T1 and T3. Neither T1 nor T3 are completed.

Methods to Overcome Deadlock
Mostly it is not possible to avoid the occurrence of Deadlock. Most of the methods available are detection and recovery based.

Detection from the Average Waiting Time of the Transaction
If more than two transactions are waiting for the long time, then it implies that at some part of the database, deadlock has occurred. So we can detect Deadlock easily.

Deadlock Detection algorithm
Deadlock detection algorithms are used to find any loops in the Database. The main advantage is that we can find the locked transactions quickly and it is enough to restart only those particular transactions.

Recovery Mechanism
Once if Deadlock is found then we have several methods to release locked Transactions.
Method 1: Release of Objects from Big Transaction
In this method the transaction which holds more number of Database Object will be taken and all Database Objects associated with that Big Transaction will be removed.

Example
If Transaction say, T1 holds exclusive lock on four objects, T2 holds same on three objects and T3 holds same on two objects then if Deadlock occurred within these transactions then T1 will be restarted.

Method 2: Restarting the Locked Transactions
In this method Database Objects associated with all Transactions are released and they will be restarted.

Example
If Transaction say, T1 holds exclusive lock on four objects, T2 holds same on three objects and T3 holds same on two objects then if Deadlock occurred within these all Transactions are restarted.

Query Optimization

As we are in the comfortable age of information technology, databases have become a necessary and fundamental tool for managing and exploiting the power of information. Because the amount of data in a database grows larger and larger as time passes, one of the most important characteristics of a database is its ability to maintain a consistent and acceptable level of performance.
The principal mechanism through which a database maintains an optimal level of performances is known as the database query optimizer; without a well-designed query optimizer, even small databases would be noticeably sluggish.
The query optimizers for some of the most popular commercial-quality databases are estimated to have required about 50 man-years of development.
It should therefore go without saying that the specific processes involved in designing the internal structure of a real-world optimizer can be overwhelmingly complex. Nevertheless, because of the optimizer's paramount importance to the robustness and flexibility of a database, it is worthwhile to engage in a survey of the theory behind the rudimentary components of a basic, cost-based query optimizer.

Query Processing

The activities involved in retrieving data from the database are known as query processing. The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries. The aims of query processing are to transform a query written in a high-level language typically SQL, into a correct and efficient execution strategy expressed in a low-level language (implementing relational algebra), and to execute the strategy to retrieve the required data. An important aspect of query processing is Query Optimization.
The activity of choosing an efficient execution strategy for processing a query is called as query optimization. As there are many equivalent transformations of the same high-level query, the aim of query optimization is to choose the one that minimizes the resource usage.
A DBMS uses different techniques to process, optimize, and execute high-level queries (SQL). A query expressed in high-level query language must be first scanned, parsed, and validated.
The scanner identifies the language components (tokens) in the text of the query, while the parser checks the correctness of the query syntax. The query is also validated (by accessing the system catalog) whether the attribute names and relation names are valid. An internal representation (tree or graph) of the query is created.
Queries are parsed and then presented to a query optimizer, which is responsible for identifying an efficient plan. The optimizer generates alternative plans and chooses the plan with the least estimated cost.

Need for Query Optimization

In high-level query languages, any given query can be processed in different ways. Resources required by each query will be different.

DBMS has the responsibility to select the optimized way to process the query. Query optimizers do not "optimize" – just try to find "reasonably good" evaluation strategies. Query optimizer uses relational algebra expressions.
The goal of query optimization is to **reduce the system resources required to fulfill a query**, and ultimately provide the user with the correct result set faster. Query optimization is important for at least a few reasons. First, it provides the user with faster results, which makes the application seem faster to the user. Secondly, it allows the system to service more queries in the same amount of time, because each request takes less time than unoptimized queries. Thirdly, query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

Basic Steps in Query Optimization

The two basic steps involved in query optimization are:

- Enumerating alternative plans for evaluating the expression. Because number of alternative plans are large.
- Estimating the cost of each enumerated plan and choosing the plan with least estimated cost.

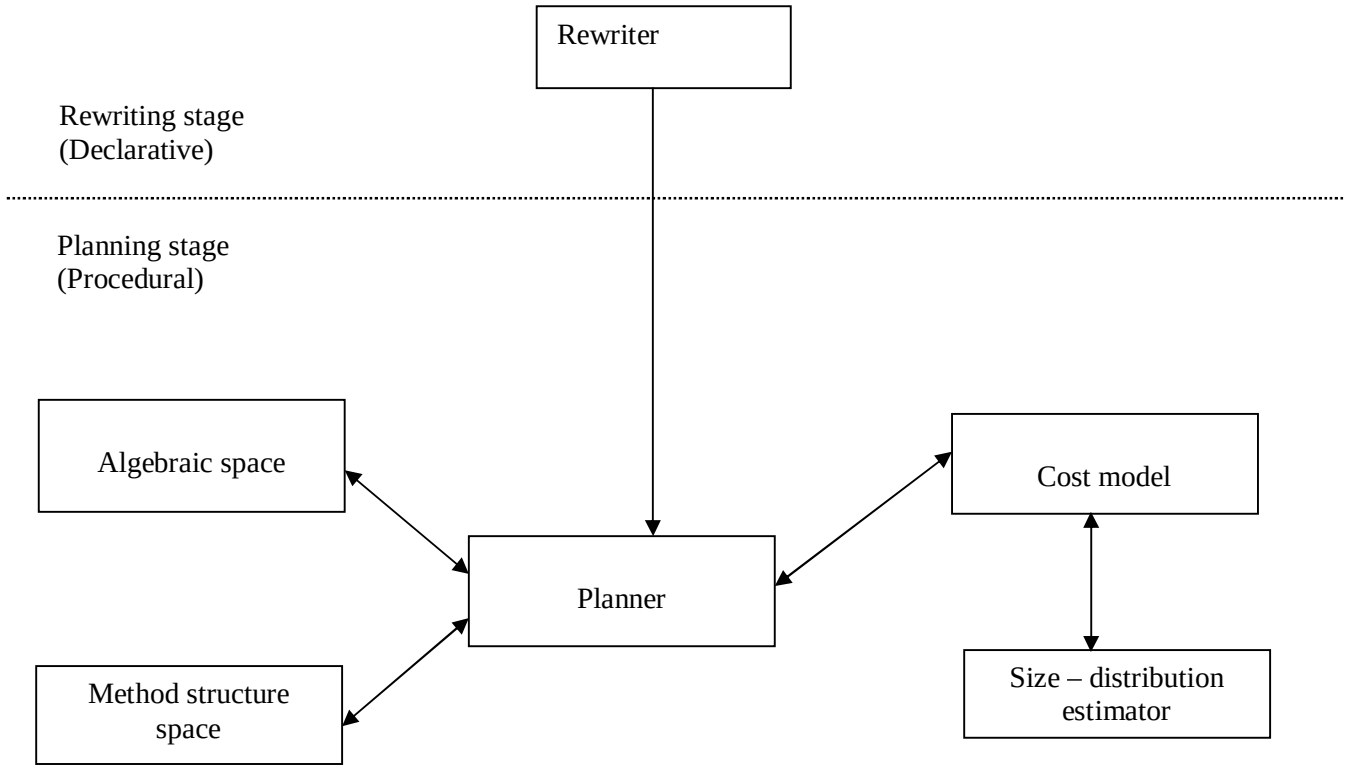
Taking SQL query for query Optimization, when it is given as input to the following system of processes, the Query undergoes to the following stages:
The DBMS begins by ***parsing*** the SQL statement. It breaks the statement into individual words, makes sure that the statement has a valid verb, legal clauses, and so on. Syntax errors and misspellings can be detected in this step. The DBMS ***validates*** the statement. It checks the statement against the system catalog. It checks whether all the tables referred exists in the database and their definition exists in the catalog.
The optimizer optimizes the statement. It explores various ways to carry out the statement. After exploring alternatives, it chooses one of them.
The optimizer then generates an ***execution plan*** for the statement. The plan is a binary representation of the steps that are required to carry out the statement; it is the DBMS equivalent of "executable code." It is carried out in Runtime Database Processor. Finally, the DBMS carries out the statement by executing the execution plan.

Query Optimizer Architecture

An abstraction of the process of generating and testing the different alternatives is essentially a modular architecture of a query optimizer.
The entire query optimization process involves two stages: Rewriting and Planning. There is only one module in the first stage, the **Rewriter**, whereas all other modules are in the second stage.

Rewriter

This module applies transformations to a given query and produces equivalent queries that are hopefully more efficient, e.g., replacement of views with their definition. The transformations performed by the Rewriter depend only on the declarative, i.e., static characteristics of queries do not take into account the actual query costs for the specific DBMS and database concerned. If the rewriting is known or assumed to always be beneficial, the original query is discarded; otherwise, it is sent to the next stage as well. By the nature of the rewriting transformations, this stage operates at the declarative level.



Modular architecture of a query optimizer

Planner

This is the main module of the ordering stage. It examines all possible execution plans for each query produced in the previous stage and selects the overall cheapest one to be used to generate the answer of the original query. It employs a search strategy, which examines the space of execution plans in a particular fashion. This space is determined by two other modules of the optimizer, the Algebraic Space and the Method-Structure Space. For the most part, these two modules and the search strategy determine the cost, i.e., running time, of the optimizer itself, which should be as low as possible.
The execution plans examined by the Planner are compared based on estimates of their cost so that the cheapest may be chosen. These costs are derived by the last two modules of the optimizer namely the Cost Model and the Size-Distribution Estimator.

Algebraic Space

This module determines the action execution orders that are to be considered by the Planner for each query sent to it. All such series of actions produce the same query answer, but usually differ in performance. They are usually represented in relational algebra as formulas or in tree form. Because of the algorithmic nature of the objects generated by this module and sent to the Planner, the overall planning stage is characterized as operating at the procedural level.

Method-Structure Space

This module determines the implementation choices that exist for the execution of each ordered series of actions specified by the Algebraic Space. This choice is related to the available join methods for each join (e.g., nested loops, merge scan, and hash join). This choice is also related to the available indices for accessing each relation, which is determined by the physical schema of each database stored in its

catalogs. Given an algebraic formula or tree from the Algebraic Space, this module produces all corresponding complete execution plans, which specify the implementation of each algebraic operator and the use of any indices.

Cost Model

This module specifies the arithmetic formulas that are used to estimate the cost of execution plans. For every different join method, for every different index type access, and in general for every distinct kind of step that can be found in an execution plan, there is a formula that gives its cost.
Given the complexity of many of these steps, most of these formulas are simple approximations of what the system actually does and are based on certain assumptions regarding issues like buffer management, disk–CPU overlap, sequential vs. random I/O, etc. The most important input parameters to a formula are the size of the buffer pool used by the corresponding step, the sizes of relations or indices accessed, and possibly various distributions of values in these relations. While the first one is determined by the DBMS for each query, the other two are estimated by the Size-Distribution Estimator.

Size-Distribution Estimator

This module specifies how the sizes (and possibly frequency distributions of attribute values) of database relations and indices as well as (sub) query results are estimated. As mentioned above, these estimates are needed by the Cost Model. The specific estimation approach adopted in this module also determines the form of statistics that need to be maintained in the catalogs of each database, if any.

Basic Algorithms for Executing Query Operations

A query basically consists of following operations. The query can be analyzed by analyzing these operations separately.

- Selection Operation
- Join Operation
- Projection Operation
- Set Operations

Select Operation

A query can have condition to select the required data. For that selection it may use several ways to search for the data. The following are some of the ways to search:

- File Scan
- Index Scan

File Scan

A number of search algorithms are possible for selecting the records from a file. The selection of records from a file is known as file scan, as the algorithm scans the records of a file to search for and retrieve records that satisfy the selection condition.

Index Scan

If the search algorithm uses an index to perform the search, then it is referred as Index Scan.

- Linear Search: This is also known as Brute Force method. In this method, every record in the file is retrieved and tested as to whether its attribute values satisfy the selection condition.
- Binary Search: If a selection condition involves an equality comparison on a key attribute on which the file is ordered, a binary search can be used. Binary search is more efficient than linear search.

- Primary Index Single Record Retrieval: If a selection condition involves an equality comparison on a primary key attribute with a primary index, we can use the primary index to retrieve that record.
- Secondary Index: This search method can be used to retrieve a single record if the indexing field has unique values (key field) or to retrieve multiple records if the indexing field is not a key. This can also be used for comparisons involving <, >, <= or >=.
- Primary Index Multiple Records Retrieval : A search condition uses comparison condition <, >, etc. on a key field with primary index is known as Primary index multiple records retrieval.
- Clustering Index: If the selection condition involves an equality comparison on a non-key attribute with a clustering index, we can use that index to retrieve all the records satisfying the condition.

Conjunctive Condition Selection

If the selection condition of the SELECT operation is a conjunctive condition (a condition that is made up of several simple conditions with the AND operator), then the DBMS can use the following additional methods to perform the selection.

Conjunctive selection: If an attribute in any single simple condition in the conjunctive condition has an access path that permits use of binary search or an index search, use that condition to retrieve the records and then check if that record satisfies the remaining condition.

Conjunctive selection using composite index: If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index exists on the combined fields we can use the index.

Query Optimization for Select Operation

Following method gives query optimization for selection operation:

i. If more than one of the attributes has an access path, then use the one that retrieves fewer disk blocks.

ii. Need to consider the selectivity of a condition: Selectivity of a condition is the number of tuples that satisfy the condition divided by total number of tuples. The smaller the selectivity the fewer the number of tuples retrieved and higher the desirability of using that condition to retrieve the records.

Join Operation

Join is one of the most time-consuming operations in query processing. Here we consider only equijoin or natural join. Two-way join is a join of two relations, and there are many ways to perform the join. Multi – way join is a join of more than two relations and number of ways to execute multi – way joins increases rapidly with number of relations.

We use methods for implementing two-way joins of form:

$$RJN\ (a = b)S$$

Where, R and S are relations which need to be joined, a and b are attributes used for conditions, and JN → Type of join.

Methods for Implementing Joins

Following are various methods to implement join operations.

Nested (Inner–Outer) Loop

For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B].

Access Structure Based Retrieval of Matching Records
If an index (or hash key) exists for one of the two join attributes – say, B of S – retrieve each record t in R, one at a time, and then use the access structure to retrieve directly all matching records from S that satisfy s[B] = t[A].

Sort-merge join
If the records of R and S are physically sorted (ordered) by value of the join attributes A and B, respectively, then both the relations are scanned in order to the join attributes, matching the records that have same values for A and B. In this case, the relations R and S are only scanned once.

Hash-join
The tuples of relations R and S are both hashed to the same hash file, using the same hashing function on the join attributes A of R and B of S as hash keys. A single pass through the relation with fewer records (say, R) hashes its tuples to the hash file buckets. A single pass through the other relation (S) then hashes each of its tuples to the appropriate bucket, where the record is combined with all matching records from R.
In practice all the above techniques are implemented by accessing whole disk blocks of a relation, rather than individual records. Depending on the availability of buffer space in memory the number of blocks read in from the file can be adjusted. It is advantageous to use the relation with fewer blocks as the outer loop relation in nested loop method.
For method using access structures to retrieve matching tuples, either the smaller relation or the file that has a match for every record (high join selection factor) should be used in the outer loop. In some cases an index may be created specifically for performing the join operation if one does not exist already.
Sort-Merge algorithm is the most efficient; sometimes the relations are sorted before merging. Hash join method is efficient if the hash file can be kept in the main memory.

Projection Operation
If the projected list of attributes has the key of the relation, then the whole relation has to be scanned. If the projected list does not have key then duplicate tuples need to be eliminated, this is done by sorting or hashing.

The Oracle Architecture
Oracle is based on the client–server architecture. The Oracle server consists of the *database* (the raw data, including log and control files) and the *instance* (the processes and system memory on the server that provide access to the database).
An instance can connect to only one database. The database consists of a *logical structure*, such as the database schema, and a *physical structure*, containing the files that make up an Oracle database.

Oracle's logical database structure
At the logical level, Oracle maintains *tablespaces*, *schemas*, and *data blocks* and *extents/segments*.

Data blocks, extents, and segments
The **data block** is the smallest unit of storage that Oracle can use or allocate. One data block corresponds to a specific number of bytes of physical disk space. The data block size can be set for each Oracle database when it is created. This data block size should be a multiple of the operating system's block size (within the system's maximum operating limit) to avoid unnecessary I/O. A data block has the following structure:

- *Header* Contains general information such as block address and type of segment.
- *Table directory* Contains information about the tables that have data in the data block.
- *Row directory* Contains information about the rows in the data block.

- *Row data* Contains the actual rows of table data. A row can span blocks.
- *Free space* Allocated for the insertion of new rows and updates to rows that require additional space.

The next level of logical database space is called an **extent**. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information. The level above an extent is called a **segment**. A segment is a set of extents allocated for a certain logical structure. For example, each table's data is stored in its own data segment, while each index's data is stored in its own index segment. Oracle dynamically allocates space when the existing extents of a segment become full. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

Tablespaces

An Oracle database is divided into logical storage units called **tablespaces**. A tablespace is used to group related logical structures together. For example, tablespaces commonly group all the application's objects to simplify some administrative operations.

Every Oracle database contains a tablespace named SYSTEM, which is created automatically when the database is created. The SYSTEM tablespace always contains the system catalog tables (called the *data dictionary* in Oracle) for the entire database. A small database might need only the SYSTEM tablespace; however, it is recommended that at least one additional tablespace is created to store user data separate from the data dictionary, thereby reducing contention among dictionary objects and schema objects for the same datafiles.

Users, schemas, and schema objects

A **user** (sometimes called a **username**) is a name defined in the database that can connect to, and access, objects. A **schema** is a named collection of schema objects, such as tables, views, indexes, clusters, and procedures, associated with a particular user. Schemas and users help DBAs manage database security.

To access a database, a user must run a database application (such as Oracle Forms or SQL*Plus) and connect using a username defined in the database. When a database user is created, a corresponding schema of the same name is created for the user. By default, once a user connects to a database, the user has access to all objects contained in the corresponding schema.

Oracle's physical database structure

The main physical database structures in Oracle are datafiles, redo log files, and control files.

Datafiles

Every Oracle database has one or more physical datafiles. The data of logical database structures (such as tables and indexes) is physically stored in these datafiles. The simplest Oracle database would have one tablespace and one datafile. A more complex database might have four tablespaces, each consisting of two datafiles, giving a total of eight datafiles.

Redo log files

Every Oracle database has a set of two or more redo log files that record all changes made to data for recovery purposes. Should a failure prevent modified data from being permanently written to the datafiles, the changes can be obtained from the redo log, thus preventing work from being lost.

Control files

Every Oracle database has a control file that contains a list of all the other files that make up the database, such as the datafiles and redo log files. For added protection, it is recommended that the control file should be multiplexed (multiple copies may be written to multiple devices). Similarly, it may be advisable to multiplex the redo log files as well.

The Oracle instance

The Oracle instance consists of the Oracle processes and shared memory required to access information in the database. The instance is made up of the Oracle background processes, the user processes, and the shared memory used by these processes. Among other things, Oracle uses shared memory for caching data and indexes as well as storing shared program code. Shared memory is broken into various *memory structures*, of which the basic ones are the System Global Area (SGA) and the Program Global Area (PGA).

- *System global area* The SGA is an area of shared memory that is used to store data and control information for one Oracle instance. The SGA is allocated when the Oracle instance starts and deallocated when the Oracle instance shuts down. The information in the SGA consists of the following memory structures, each of which has a fixed size and is created at instance startup:
  - *Database buffer cache* This contains the most recently used data blocks from the database. These blocks can contain modified data that has not yet been written to disk (*dirty blocks*), blocks that have not been modified, or blocks that have been written to disk since modification (*clean blocks*). By storing the most recently used blocks, the most active buffers stay in memory to reduce I/O and improve performance.
  - *Redo log buffer* This contains the redo log file entries, which are used for recovery purposes. The background process LGWR writes the redo log buffer to the active online redo log file on disk.
  - *Shared pool* This contains the shared memory structures such as shared SQL areas in the library cache and internal information in the data dictionary. The shared SQL areas contain parse trees and execution plans for the SQL queries. If multiple applications issue the same SQL statement, each can access the shared SQL area to reduce the amount of memory needed and to reduce the processing time used for parsing and execution
- *Program global area* The PGA is an area of shared memory that is used to store data and control information for the Oracle server processes. The size and content of the PGA depends on the Oracle server options installed.
- *User processes* Each user process represents the user's connection to the Oracle server (for example, through SQL*Plus or an Oracle Forms application). The user process manipulates the user's input, communicates with the Oracle server process, displays the information requested by the user and, if required, processes this information into a more useful form.
- *Oracle processes* Oracle (server) processes perform functions for users. Oracle processes can be split into two groups: *server processes* (which handle requests from connected user processes) and *background processes* (which perform asynchronous I/O and provide increased parallelism for improved performance and reliability).

The following are the background processes:

- *Database Writer (DBWR)* The DBWR process is responsible for writing the modified (dirty) blocks from the buffer cache in the SGA to datafiles on disk. An Oracle instance can have up to ten DBWR processes, named DBW0 to DBW9, to handle I/O to multiple datafiles. Oracle employs a technique known as write-ahead logging, which means that the DBWR process performs *batched writes* whenever the buffers need to be freed, not necessarily at the point the transaction commits.
- *Log Writer (LGWR)* The LGWR process is responsible for writing data from the log buffer to the redo log.
- *CHKPT*
  CKPT was responsible for signaling regular checkpoints.
- *SMON*
  SMON *mounts* a database by locating and validating the database controlfile. It then *opens* a database by locating and validating all the datafiles and online log files. Once the database is

opened and in use, SMON is responsible for various housekeeping tasks, such as collating free space in datafiles.

- *PMON*
  PMON monitors all the server processes and detects any problems with the sessions. If a session has terminated abnormally, PMON will destroy the server process, return its PGA memory to the operating system's free memory pool, and roll back any incomplete transaction that may have been in progress.

Database Security and Recovery

Database Security

Introduction

Database security issues are often lumped together with data integrity issues, but the two concepts are really quite distinct. Security refers to the protection of data against unauthorized disclosure, alteration, or destruction; integrity refers to the accuracy or validity of that data. To put it a little glibly:

- Security means protecting the data against unauthorized users.
- Integrity means protecting the data against authorized users.

In both cases, system needs to be aware of certain constraints that users must not violate; in both cases those constraints must be specified (typically by the DBA) in some suitable language, and must be maintained in the system catalog; and in both cases the DBMS must monitor user operations in order to ensure that the constraints are enforced.

Data are the most valuable resource for an organization. Security in a database involves mechanisms to protect the data and ensure that it is not accessed, altered, or deleted without proper authorization. The database in Defense Research Development Organization (DRDO), Atomic Research Centre, and Space Research Centre contains vital data and it should not be revealed to unauthorized persons. To protect the secret data there should be restriction to data access. This ensures the confidentiality of the data. Also the data should be protected from accidental destruction. Due to advancement in information technology, people share the data through World Wide Web. As a result the data become vulnerable to hackers. A database should not only provide the end user with the data needed to function, but also it should provide protection for the data.

Need for Database Security

The need for database security is given below:

- In the case of shared data, multiple users try to access the data at the same time. In order to maintain the consistency of the data in the database, database security is needed.
- Due to the advancement of internet, data are accessed through World Wide Web, to protect the data against hackers, database security is needed.
- The plastic money (Credit card) is more popular. The money transaction has to be safe. More specialized software both to enter the system illegally to extract data and to analyze the information obtained is available. Hence, it is necessary to protect the data/money.

General Considerations

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

- Legal, social, and ethical aspects
- Physical controls
- Policy questions
- Operational problems
- Hardware control

- Operating system support
- Issues that are the specific concern of the database system itself

There are two broad approaches to data security. The approaches are known as discretionary and mandatory control, respectively. In both cases, the unit of data or "data object" that might need to be protected can range all the way from an entire database on the one hand to a specific component within a specific tuple on the other. How the two approaches differ is indicated by the following brief outline.

In the case of discretionary control, a given user will typically have different access rights (also known as privileges) on different objects; further, there are very few – inherent limitations, that is – regarding which users can have which rights on which object (for example, user U1 might be able to see A but not B, while user U2 might be able to see B but not A). Discretionary schemes are thus very flexible.

In the case of mandatory control, each data object is labeled with a certain classification level, and each user is given a certain clearance level.
A given data object can then be accessed only by users with the appropriate clearance. Mandatory schemes thus tend to be hierarchic in nature and are hence comparatively rigid. (If user U1 can see A but not B, then the classification of B must be higher than that of A, and so no user U2 can see B but not A.)

Regardless of whether we are dealing with a discretionary scheme or a mandatory one, all decisions as to which users are allowed to perform which operations on which objects are policy decisions, not technical ones. As such, they are clearly outside the jurisdiction of the DBMS as such; all the DBMS can do is enforcing those decisions once they are made. It follows that:

- The results of policy decisions made must be known to the system (this is done by means of statement in some appropriate definitional language).
- There must be a means of checking a given access request against the applicable security constraint in the catalog. (By "access request" here we mean the combination of requested operation plus requested object plus requesting user, in general.) That checking is done by the DBMS's security subsystem, also known as the authorization subsystem.

In order to decide which security constraints are applicable to a given access request, the system must be able to recognize the source of that request, i.e., it must be able to recognize the requesting user. For that reason, when users sign on to the system, they are typically required to supply, not only their user ID (to say who they are), but also a password (to prove they are who they say they are). The password is supposedly known only to the system and to legitimate users of the user ID concerned.

Regarding the last point, incidentally, note that any number of distinct users might be able to share the same ID. In this way the system can support user groups, and can thus provide a way of allowing everyone in the accounting department to share the same privileges on the same objects. The operations of adding individual users to or removing individual users from a given group can then be performed independently of the operation of specifying which privileges on which objects apply to that group. Note, however, that the obvious place to keep a record of which users are in which groups is once again the catalog (or perhaps the database itself).

Database Security System

The person responsible for security of the database is usually database administrator (DBA). The database administrator must consider variety of potential threats to the system. Database administrators create authorization rules that define who can access what parts of database for what operations. Enforcement of authorization rules involves authenticating the user and ensuring that authorization rules are not violated by access requests. DBMS should support creation and storage of authorization rules and enforcement of authorization rules when users access a database.

The database security system stores authorization rules and enforces them for each database access. The authorization rules define authorized users, allowable operations, and accessible parts of a database. When a group of users access the data in the database, then privileges can be assigned to groups rather than individual users. Users are assigned to groups and given passwords.
In a nutshell, database security involves allowing and disallowing users from performing actions on the database and the objects within it. Database security is about controlling access to information. That is, some information should be available freely and other information should only be available to certain authorized people or groups.

Database Security Goals and Threats

Some of the goals and threats of database security are given below:

- Goal. Confidentiality (secrecy or privacy). Data are only accessible (read type) by authorized subjects (users or processes).
- ❖ Threat. Improper release of information caused by reading of data through intentional or accidental access by improper users. This includes inferring of unauthorized data from authorized observations from data.
- Goal. To ensure data integrity which means data can only be modified by authorized subjects.
- ❖ Threat. Improper handling or modification of data.

- Goal. Availability (denial of service). Data are accessible to authorized subjects.
- ❖ Threat. Action could prevent subjects from accessing data for which they are authorized.

Security Threat Classification

Security threat can be broadly classified into accidental, intentional according to the way they occur.
The accidental threats include human errors, errors in software, and natural or accidental disasters:

- Human errors include giving incorrect input, incorrect use of applications.
- Errors in software include incorrect application of security policies, denial of access to authorized users.
- Natural or accidental disasters include the damage of hardware or software.

The intentional threat includes authorized users who abuse their privileges and authority, hostile agents like improper users executing improper reading or writing of data, legal use of applications can mask fraudulent purpose.

Classification of Database Security

The database security can be broadly classified into physical and logical security.
Database recovery refers to the process of restoring database to a correct state in the event of a failure.

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

Database Security at Design Level

It is necessary to take care of the database security at the stage of database design. Few guidelines to build the most secure system are:

i. The database design should be simple. If the database is simple and easier to use, then the possibility that the data being corrupted by the authorized user is less.
ii. The database has to be normalized. The normalized database is almost free from update anomalies. It is harder to impose normalization on the relations after the database is in use. Hence, it is necessary to normalize the database at the design stage itself.
iii. The designer of the database should decide the privilege for each group of users. If no privileges are assumed by any user, there is less likelihood that a user will be able to gain illegal access.
iv. Create unique view for each user or group of users. Although "VIEW" promotes security by restricting user access to data, they are not adequate security measures, because unauthorized persons may gain knowledge of or access to a particular view.

Database Security at the Maintenance Level

Once the database is designed, the database administrator is playing a crucial role in the maintenance of the database. The security issues with respect to maintenance can be classified into:

- Operating system issues and availability
- Confidentiality and accountability through authorization rules
- Encryption
- Authentication schemes

Operating System Issues and Availability

The system administrator normally takes care of the operating system security. The database administrator is playing a key role in the physical security issues. The operating system should verify that users and application programs attempting to access the system are authorized. Accounts and passwords for the entire database system are handled by the database administrator.

Confidentiality and Accountability

Accountability means that the system does not allow illegal entry. Accountability is related to both prevention and detection of illegal actions. Accountability is assured by monitoring the authentication and authorization of users.
Authorization rules are controls incorporated in the data management system that restrict access to data and also restrict the actions that people may take when they access data.
Authentication can be carried out by the operating system level or by the relational database management system (RDBMS). In case, the system administrator or the database administrator creates for every user an individual account or username. In addition to these accounts, users are also assigned passwords.

Encryption

Encryption can be used for highly sensitive data like financial data, military data. Encryption is the coding of data so that they cannot be read and understood easily. Some DBMS products include encryption routines that automatically encode sensitive data when they are stored or transmitted over communication channel. Any system that provides encryption facilities must also provide complementary routines for decoding the data. These decoding routines must be protected by adequate security, or else the advantage of encryption is lost.

Authentication Schemes

Authentication schemes are the mechanisms that determine whether a user is who he or she claims to be. Authentication can be carried out at the operating system level or by the RDBMS. The database administrator creates for every user an individual account or user name. In addition to these accounts, users are also assigned passwords. A password is a sequence of characters, numbers, or a combination of both which is known only to the system and its legitimate user. Since the password is the first line of defense against unauthorized use by outsiders, it needs to be kept confidential by its legitimate user. It is

highly recommended that users change their password frequently. The password needs to be hard to guess, but easy for the user to remember. Passwords cannot, of themselves, ensure the security of a computer and its databases, because they give no indication of who is trying to gain access.
The password can also be tapped; hence mere password cannot ensure the security of the database. To circumvent this problem, the industry is developing devices and techniques to positively identify any prospective user. The most promising of these appear to be biometric devices, which measure or detect personal characteristics such as fingerprints, voice prints, retina prints, or signature dynamics. To implement this approach, several companies have developed a smart card which is a thin plastic card with an embedded microprocessor.
An individual's unique biometric data are stored permanently on the card. To access the database the user inserts the card and the biometric device reads the person's unique feature. The actual biometric data are then compared with the stored data, and the two must match for the user to gain computer access. A lost or stolen card would be useless to another person, since biometric data would not match.

Assignment: detailed security thro' access control

Database Recovery

Recovery brings the database from the temporary inconsistent state to a consistent state. Database recovery can also be defined as mechanisms for restoring a database quickly and accurately after loss or damage. Databases are damaged due to human error, hardware failure, incorrect or invalid data, program errors, computer viruses, or natural catastrophes. Since the organization depends on its database, the database management system must provide mechanisms for restoring a database quickly and accurately after loss or damage.

Different Types of Database Failures

A wide variety of failures can occur in processing a database, ranging from the input of an incorrect data value or complete loss or destruction of the database. Some of the types of failures are listed below:

- **i.** System crashes, resulting in loss of main memory
- **ii.** Media failures, resulting in loss of parts of secondary storage
- **iii.** Application software errors
- **iv.** Natural physical disasters
- **v.** Carelessness or unintentional destruction of data or facilities
- **vi.** Sabotage

Recovery Facilities

DBMS should provide following facilities to assist with recovery.

1) Backup mechanism, which makes periodic backup copies of database
2) Logging facilities, which keep track of current state of transactions and database changes
3) Checkpoint facility, which enables updates to database in progress to be made permanent
4) Recovery manager, which allows DBMS to restore the database to a consistent state following a failure

Backup Mechanism

The DBMS should provide backup facilities that produce a backup copy of the entire database. Typically, a backup copy is produced at least once per day. The copy should be stored in a secured location where it is protected from loss or damage. The backup copy is used to restore the database in the event of hardware failure, catastrophic loss, or damage. With large databases, regular backups may be impractical, as the time required to perform the backup may exceed that available. As a result, backups may be taken of dynamic data regularly but backups of static data, which do not change frequently, may be taken less often.

Logging Facilities
Basically there are two types of log, "transaction log" and "database change log." A transaction log is a record of the essential data for each transaction that is processed against the database. In database change log, there are before and after images of records that have been modified.
Transaction log: Transaction log contains a record of the essential data for each transaction that is processed against the database. Data that are typically recorded for each transaction include the transaction code or identification, action or type of transaction, time of the transaction, terminal number or user ID, input data values, table and records accessed, records modified, and possibly the old and new field values.
Database change log: The database change log contains before and after images of records that have been modified by transactions. A before-image is a copy of a record before it has been modified, and an after-image is a copy of the same record after it has been modified.

Checkpoint Facility
A checkpoint facility in a DBMS periodically refuses to accept any new transactions. All transactions in progress are completed, and the journal files are brought up to date. At this point, the system is in a quiet state, and the database and transaction logs are synchronized. The DBMS writes a special record (called a checkpoint record) to the log file, which is like a snapshot of the state of the database. The checkpoint record contains information necessary to restart the system. Any dirty data blocks are written from memory to disk storage, thus ensuring that all changes made prior to taking the checkpoint have been written to long-term storage. A DBMS may perform checkpoints automatically or in response to commands in user application programs. Checkpoints should be taken frequently.

Recovery Manager
The recovery manager is a module of the DBMS which restores the database to a correct condition when a failure occurs and which resumes processing user requests. The recovery manager uses the logs to restore the database.

Main Recovery Techniques

Three main recovery techniques that are commonly employed are:

1) Deferred update
2) Immediate update
3) Shadow paging

Deferred update: Deferred updates are not written to the database until after a transaction has reached its commit point. If transaction fails before commit, it will not have modified database and so no undoing of changes are required. Deferred update may be necessary to redo updates of committed transactions as their effect may not have reached database.

Immediate update: In the case of immediate update, updates are applied to database as they occur. There is a need to redo updates of committed transactions following a failure. Also there may be need to undo effects of transactions that had not committed at time of failure. It is essential that log records are written before write to database. If no "transaction commit" record in log, then that transaction was active at failure and must be undone. Undo operations are performed in reverse order in which they were written to log.

Shadow paging: Shadow paging maintains two page tables during life of a transaction, current page and shadow page table. When transaction starts, two pages are the same. Shadow page table is never changed thereafter and is used to restore database in the event of failure. During transaction, current page table

records all updates to database. When transaction completes, current page table becomes shadow page table.

Crash Recovery

Crash recovery is the process of protecting the database from catastrophic system failures and media failures. Recovery manager of a DBMS is responsible for ensuring transaction atomicity and durability. Atomicity is attained by undoing the actions of transactions that do not commit. Durability is attained by making sure that all actions of committed transactions survive system crashes.

Recovery Manager

Recovery manager is responsible for ensuring transaction atomicity and durability. To save the state of database for the period of times it performs few operations. They are:

- Saving checkpoints
- Stealing frames
- Forcing pages

Saving checkpoints: It will save the status of the database in the period of time duration. So if any crashes occur, then database can be restored into last saved check point.

Steal approach: In this case, the object can be written into disk before the transaction which holds the object is committed. This is happening when the buffer manager chooses the same place to replace by some other page and at the same time another transaction require the same page. This method is called as stealing frames.

Forcing page: In this case, once if the transaction completed the entire objects associated with it should be forced to disk or written to the disk. But it will result in more I/O cost. So normally we use only no-force approach.

ARIES Algorithm

ARIES is a recovery algorithm designed to work with a steal, no-force approach. It is more simple and flexible than other algorithms.
ARIES algorithm is used by the recovery manager which is invoked after a crash. Recovery manager will perform restart operations.

Main Key Terms Used in ARIES

Log. These are all file which contain several records. These records contain the information about the state of database at any time. These records are written by the DBMS while any changes done in the database. Normally copy of the log file placed in the different parts of the disk for safety.
LSN: The abbreviation of LSN is log sequence number. It is the ID given to each record in the log file. It will be in monotonically ascending order.
Page LSN: For recovery purpose, every page in the database contains the LSN of the most recent log record that describes a change to this page. This LSN is called the page LSN.
CLR: The abbreviation of CLR is compensation log record. It is written just before the change recorded in an update log record U is undone.
WAL: The abbreviation of WAL is write-ahead log. Before updating a page to disk, every update log record that describes a change to this page must be forced to stable storage. This is accomplished by forcing all log records up to and including the one with LSN equal to the page LSN to stable storage before writing the page to disk.
There are three phases in restarting process, they are:

1) Analysis

2) Redo
3) Undo

Analysis: In this phase, it will identify whether any page present in the buffer pool is not written into disk and activate the transactions which are in active at the time of crash.
Redo: In this phase, all the operations are restarted and the state of database at the time of crash is obtained. This is done by the help of log files.
Undo: In this phase, the actions of uncommitted transactions are undone.
So only committed are taken into account.

ARIES algorithm has three main principles:

- Write-ahead logging
- Repeating history during redo
- Logging changes during undo

Write-ahead logging: This states that any change to a database object is first recorded in the log; the record in the log must be written to stable storage before the change to the database object is written to disk.

Repeating history during redo: On restart after a crash, ARIES retraces all actions of the DBMS before the crash and brings the system back to the exact state that it was in at the time of crash. Then, it undoes the actions of transactions still active at the time of crash.

Logging changes during undo: Changes made to the database while undoing a transaction are logged to ensure such an action is not repeated in the event of repeated restarts.

Elements of ARIES

i. The log
ii. Tables
iii. The write-ahead log protocol
iv. Checkpointing

The Log
It records a history of actions that are executed by DBMS. The most recent portion of the log is called as log tail. This page will be kept at main memory and periodically it will be forced to disk. It contains several records. Each record is uniquely identified by LSN.
A log record is written for each of the following actions:

- Updating a page
- Commit
- Abort
- End
- Undoing an update

Updating a page: After modifying a page, an update type record is appended to the log tail. The page LSN of this page is then set to the update log record.
Before-image is the value of the changed bytes before the change. Afterimage is the value of the changed bytes after the change.

Tables
In addition of log ARIES, it maintains the following two tables to maintain recovery related information:

- Transaction table
- Dirty page table

Transaction table: This table contains one entry for each active transaction.
The entry contains the transaction ID, the status and a field called last LSN, which is the LSN of the most recent log record for this transaction. The status of a transaction can be that it is in progress, committed, or aborted.

Dirty page table. This table contains one entry for each dirty page in the buffer pool, i.e., each page with changes not yet reflected on disk. The entry contains a field record LSN, which is the LSN of the first log record that caused the page to become dirty.
Record LSN in the dirty page table and last LSN in the transaction table are pointing to the corresponding records in the log table.

Write-Ahead Log Protocol
WAL is the fundamental rule that ensures that a record of every change to the database is available while attempting to recover from crash. If a transaction made a change and committed, the no-force approach means that some of these changes may not have been written to disk at the time of a subsequent crash. Without a record of these changes, there would be no way to ensure that the changes of a committed transaction survive crashes.
According to its rules when a transaction is completed its log tail is forced to disk, even a no-force approach is used.

Checkpointing
A checkpoint is used to reduce amount of work to be done during restart in the event of a subsequent crash.
Checkpointing in ARIES has three steps:

- Begin checkpoint
- End checkpoint
- Fuzzy checkpoint

Begin checkpoint. It is written to indicate the checkpoint is starts.
End checkpoint. It is written after begin checkpoint. It contains current contents of transaction table and the dirty page table, and appended to the log.
Fuzzy checkpoint: It is written after end checkpoint is forced to the disk.
While the end checkpoint is being constructed, the DBMS continues executing transactions and writing other log records; the only guarantee we have is that the transaction table and dirty page table are accurate as the time of the begin checkpoint.

Physical Database Design

Introduction

Physical database design describes the storage structures and access methods used in system. The goal of physical database design is to specify all identifying and operational characteristics of the data that will be recorded in the information system. The physical database design specifies how database records are stored, accessed, and related to ensure adequate performance. The physical database design specifies the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures. The physical organization of data has a major impact on database system performance because it is the level at which actual implementation takes place in physical storage.

Goals of Physical Database Design

The goal of physical database design is to create a design providing the best response time at the lowest cost. Here response time refers to the time required to access the data, and the cost associated with CPU, memory disk input/output. The main goals of good physical database design are summarized as:

- A good physical database design should achieve high packing density, which implies minimum wastage space.
- A good physical database design should achieve fast response time.
- The physical database design should also support a high volume of transactions.

Physical Design Steps

The various steps in physical database design are:

1. Stored record format design
2. Stored record clustering
3. Access method design
4. Program design

Step 1: Stored Record Format Design

The visible component of the physical database structure is the stored record format design. Stored record format design addresses the problem of formatting stored data by analysis of the characteristics of data item types, distribution of their values, and their usage by various applications. Decisions on redundancy of data, derived vs. explicitly stored values of data, and data compression are made here. Certain data items are often accessed far more frequently than others, but each time a particular piece of data is needed, the entire stored record, and all stored records in a physical block as well, must be accessed. Record partitioning defines an allocation of individual data items to separate physical devices of the same or different type, or separate extents on the same device, so that total cost of accessing data for a given set of user applications is minimized. Logically, data items related to a single entity are still considered to be connected, and physically they can still be retrieved together when necessary. An extent is a contiguous area of physical storage on a particular device.

Step 2: Stored Record Clustering

One of the most important physical design considerations is the physical allocations of stored records, as a whole, to physical extents. Record clustering refers to the allocation of records of different types into physical clusters to take advantage of physical sequentiality whenever possible. Analysis of record clustering must take access path configuration into account to avoid access time degradation due to new placement of records. Associated with both record clustering and record partitioning is the selection of physical block size. Blocks in a given clustered extent are influenced to some extent by stored record size, storage characteristics of the physical devices. Larger blocks are typically associated with sequential processing and smaller blocks with random processing.

Step 3: Access Method Design

The critical components of an access method are storage structure and search mechanisms. Storage structure defines the limits of possible access paths through indexes and stored records, and the search mechanisms define which paths are to be taken for a given applications. Access method design is often defined in terms of primary and secondary access path structure. The primary access paths are associated with initial record loading, or placement, and usually involve retrieval via the primary key. Individual files are first designed in this manner to process the dominant application most efficiently. Access time

can be greatly reduced through secondary indexes, but at the expense of increased storage space overhead and index maintenance.

Step 4: Program Design
Standard DBMS routines should be used for all accessing, and query or update transaction optimization should be performed at the systems software level.
Consequently, application program design should be completed when the logical database structure is known.

Implementation of Physical Model

The implementation of the physical model is dependent on the hardware and software being used by the company. The hardware can determine what type of software can be used because software is normally developed according to common hardware and operating system platforms. Some database software might only be available for Windows NT systems, whereas other software products such as Oracle are available on a wider range of operating system platforms, such as UNIX. The available hardware is also important during the implementation of the physical model because data are physically distributed into one or more physical disk drives. Normally, the more physical drives available, the better the performance of the database after the implementation.

File Organization

A database is stored as collection of files. A file is a set of records or relations of same type. This definition includes files of text, which can be regarded as files of 1-byte records. A record can be seen as a collection of related fields containing elementary data. Data files can exist in both primary and secondary memory, they are almost always held in secondary memory because data files are often voluminous, and the capacity of a data file can easily exceed the capacity of primary memory. Consequently, only portions of large files can be held in main memory at one time. File organization is a technique for arranging records of a file in secondary storage.

Factors to be Considered in File Organization

The problem in selecting a particular file organization is to choose a structure that will satisfy certain requirements. For example, a user may need to retrieve records in sequence and may also need fast access to a particular record. In this case, suitable organizations include hash files in which the key order is preserved, B-trees and indexed sequential access method (ISAM) files.

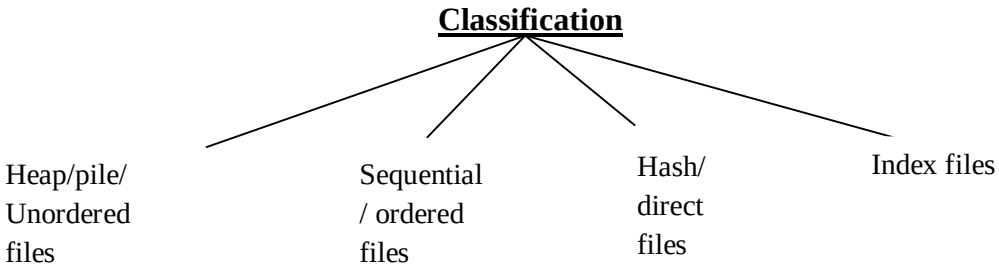
Some of the factors which are preferred in choosing the file organization are given below:

- Access should be fast. Here access refers to data access
- Storage space has to be efficiently used
- Minimizing the need for reorganization
- Accommodating growth

It is difficult to determine the best file organization and the most efficient access technique for a given situation. A good approach is to simulate the behavior of a number of candidate organizations. The simulator requires three sets of input parameters: file characteristics, user requirements, and hardware characteristics. File characteristics are logical properties of the file.
File characteristics include the number of records in the file and the average attribute length. User requirements are concerned with the accesses and changes to the file. This includes the number of deletions per day and the number of times a month the whole file is read serially. Hardware characteristics are parameters of the available storage devices. These characteristics include block size, tracks per cylinder, and the storage cost per megabyte.

File Organization Classification

File organization can be broadly classified into two types (1) primary file organization and (2) secondary file organization



Heap File Organization

Heap file is otherwise known as random file or pile file. In heap file organization, file records are inserted at the end of the file or in any file block with free space, hence insertion of record is efficient. Data in a file are collected in the order that it arrives. It is not analyzed, categorized, or forced to fit field definitions or field sizes. At best, the order of the records may be chronological.
Records may be of variable length and need not have similar sets of data elements.

Uses of Heap File Organization

Heap files are used in situations where data are collected prior to processing, where data are not easy to organize, and in some research on file structures.
Since much of the data collected in real-world situations are in the form of piles, this file organization is considered as the base for other evaluations.

Drawback of Heap File Organization

In heap file organization, data analysis can become very expensive because of the time required for retrieval of a statistically adequate number of sample records:

- Searching of record is difficult. Normally linear search is used to locate the record.
- Deletion of record is difficult. Because if we want to delete a particular record, first we have to locate the file and delete.

Good in the Following Situations

Heap file is good under the following situations:

- Bulk data have to be loaded.
- Relations which are always have few pages long because search is easy.

Bad under Situations

Heap file is bad under the following situation:

- Output is in sorted order.

Sequential File Organization

Sequential files are also called as ordered files. In sequential file, the file records are kept sorted by the value of an ordering key. The ordering field has unique value. A sequential file is a set of contiguously stored records on a physical device such as a disk, tape, or CD-ROM.
A block is the basic unit of input/output from disk to RAM. It can range in size from a fraction of a record to hundreds or thousands of records. Typically a block size ranges from one to hundred records. If a database has normalized records, i.e., records of constant size, then the number of records in a block is called blocking factor. For consistency and ease of programming, block sizes are usually constant for a whole system. On the other hand, almost all relational systems allow variable-size records, hence average record size can be used for simplicity.

Sequential Processing of File
If we have a file of "n" records then the basic performance measures for sequential search of the entire file is given below:

- Logical record accesses=n, where n is the number of records in a file.
- Sequential block accesses=ceil (n/blocking factor).
- Random block accesses=0.
- Once a sequential file is created, records can be added at the end of the file.
- It is not possible to insert records in the middle of the file without rewriting the file.
- Searching method used in sequential file organization is binary search.

Draw Back
If new record is inserted or some record is deleted the file has to be reorganized, which is time consuming.

Hash File Organization

Hash files are also called direct files. Hashing is nothing but a method of distributing data evenly to different areas of memory. In hash file organization, the records are organized using hashing algorithm. Hash file organization is suitable for random access. In hash file organization, a hash function is applied to each record key, which returns a number which is used to indicate the position of the record in the file. The hash function must be used for both reading and writing.

Hashing Function

Hash function is used to locate record for access, insertion, and deletion. The hash function is given by:
Hashing function = K mod B.
K→Key value.
B→No of buckets.

Bucket

A bucket is a unit of storage containing one or more records (a bucket is typically a disk block). In a hash file organization, the bucket of a record is obtained directly from its search key value using a hash function.
Hashing function = K mod B.
K→Key value.
B→No of buckets.
If B=3,

Choice of Bucket
Here the choice of the bucket refers to the fact that whether the bucket size is large or small. If the bucket size is large then, storage space required is large hence waste; on the other hand if the bucket size is small, then there is a chance of overflow.

As the bucket size increases, the probability of overflow decreases but the time taken to search for a record in the bucket may increase.

Collision
There is a collision during an insertion when two records hash to the same bucket address. Two keys that hash to the same address are sometimes termed synonyms. Collisions are not a problem if the bucket is not full, because the records are simply stored in available spaces in the bucket. Overflow occurs during insertion when a record is hashed to a bucket that is already full.

Methods to Avoid Overflow
Overflow occurs when a record hashes to a full bucket. In open addressing, the addresses of buckets to search are calculated dynamically. In chaining, chains of overflow records are rooted in the home buckets.

Open Addressing

Open addressing is a technique to avoid overflow. From the name open addressing it is clear that we have to generate a list of bucket addresses. The list of bucket addresses is generated from the key of the record. The processes of generation of list of bucket addresses is denoted by
Ai = f (i, key) i = 0, 1, 2, 3, . . .
Where Ai is the list of bucket address and i is an integer.

If the bucket Ai is full, then the bucket Ai+1 are examined. When retrieving a record, buckets are examined until one is found that either contains the required record or has an empty space. An empty space indicates that the record being searched for is not in the file. This method of resolving overflow was proposed by Peterson who termed it open addressing. A good function “f” should ensure that each integer 0 to N−1 (where N is the number of home buckets) appears in the generated list of bucket addresses.

Chaining

A second solution to the problem of overflows is called chaining or closed addressing. In this method, lists of records that have overflowed are rooted in appropriate home buckets. The overflow records are put in any convenient place, perhaps in a separate overflow area or in an arbitrary bucket that is not full. In contrast to open addressing, the location used does not typically depend on the contents of the record. Let us consider three variations of the basic idea outlined above (1) separate lists, (2) coalescing lists, and (3) Directory methods.

Separate lists. In the separate lists method we link together records overflowing from the same home bucket. The list is rooted in that bucket. Only the records on the list need to be examined when searching, and because any one might be the one looked for, comparisons are minimized. Deletions are straightforward. If we delete a record from a home bucket, we can replace it by one on the overflow list. If we delete a record on a list, it is removed from the list in the conventional way.

Coalescing lists. The separate list method requires a comparatively large amount of space for pointers. A second possibility is to store records in spare space in home buckets. Each bucket has a single pointer pointing to the next bucket to try when searching. Pointers are established as records overflow.
This method reduces pointer overhead, but many more records may have to be examined when searching.

Directory methods. In methods involving directories, room is allocated in a home bucket beyond that needed to store records. The extra space is used to hold pointers to records overflowing from the bucket and their keys. As long as all overflows can be pointed to in this way, this method, is fast.

Index File Organization

Index is a collection of data entries plus a way to quickly find entries with given key values. Index is a mechanism for efficiently locating row(s) without having to scan entire table.

Advantage of Indexing

Index speeds up selections on the search key field. Search key is an attribute or set of attributes used to look up records in a file. It is possible to build more than one index for the same table:

- Index in the book allows us to locate specific page in the book.
- Index in the record allows us to find specific record in the file.

Classification of Index

Indexes can be broadly classified into primary index, secondary index, dense index, sparse index, bitmap, and single and multilevel as illustrated below:
Primary index: Primary index is one whose search key specifies the sequential order of the file.
Secondary index: Secondary index improves the performance of queries that use keys other than primary search key.
Dense index: Dense index has index entry for each data record.
Sparse index: Sparse index has index entry for each page of data file.

Search Key

Search key is an attribute or set attributes used to look up records in a file.

Dense	**Vs**	**Sparse Index**
1. Fast search.		1. Access time is increase.
2. Space is more.		2. Less storage space.

Tree-Structured Indexes

A tree is a structure in which each node has at most one parent except for the root or top node. Tree-structured indexes are ideal for range-searches, and also good for equality searches. These tree-structured indexes can be classified into **(1)** ISAM, **(2)** B-tree and, **(3)** B+ tree.

ISAM

ISAM stands for indexed sequential access method. ISAM is a static index structure that is effective when the file is not frequently updated. In ISAM new entries are inserted in overflow pages.

B-Tree

The B-tree is a very popular structure for organizing and maintaining large indexes. B-trees were studied in early 1970s by Bayer, McCreight, and Comer.
B-tree is a generalization of binary tree in which two or more branches may be taken from each node. B-tree is called balanced tree because the access paths to different records of equal length. B-tree has the ability to quickly search huge quantities of data. B-tree adapts well to insertions and deletions.
One of the earliest B-tree search mechanisms was used at Boeing Labs. Later, the original B-tree spawned several variants, including the B+ developed by Prof. Donald Knuth. An index provides fast access to data when the data can be searched by the value that is the index key.

B-Tree Properties

A B-tree is a generalization of binary tree in which two or more branches may be taken from each node.
A B-tree of order k has the following properties:

- Each path from the root node to a leaf node has the same length, h, also called the height of the B-tree (i.e., h is the number of nodes from the root to the leaf, inclusive).
- Each node, except the root and leaves, has at least k+1 child nodes and no more than 2k+1 child nodes.
- The root node may have as few as two child nodes, but no more than 2k+1 child nodes.
- Each node, except the root, has at least k keys and no more than 2k keys.

The root may have as few as one key. In general, any nonleaf (branch) node with j keys must have j+1 child nodes.

Differences between B and B+ Tree

B-tree	**B+ tree**
In B-tree, nonleaf nodes are larger than leaf nodes	In B+ tree leaf and nonleaf nodes are of same size
Deletion in B-tree is complicated In	B+ tree, deleted entry always appears in a leaf, and hence it is easy to delete an entry
Pointers to data records exist at all levels of the tree	Pointers to data records exist only at the leaves

Advantages of B-Trees
The major advantages of B-trees are summarized below:

- Secondary storage space utilization is better than 50% at all times. Storage space is dynamically allocated and reclaimed, and no service degradation occurs when storage utilization becomes very high.
- Random access requires very few steps and is comparable to hashing and multiple index methods.
- Record insertions and deletions are efficiently handled on the average, allowing maintenance of the natural order of keys for sequential processing and proper tree balance to maintain fast random retrieval.
- Allows efficient batch processing by maintaining key order.

Bitmap Index
Bitmap index is optimal for indexing a column containing few unique values.

Benefits of Bitmap Index
The benefits of bitmap index are summarized as:

- Reduced response time for large classes of ad hoc queries.
- A substantial reduction of space usage compared to other indexing techniques.

Fully indexing a large table with a normal index can be expensive in terms of space since the index can be several times larger than the data in the table.
Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

Data Storage Devices

The data stored by an organization double in every 3 or 4 years. Hence the selection of data storage devices is a key consideration for data managers.

Factors to be Considered in Selecting Data Storage Devices

The following factors have to be considered while evaluating data storage options:

- Online storage
- Backup files

- Archival storage

When the device is used to store online data, then one has to give importance to access speed and capacity, because many firms require rapid response to large volumes of data.
Backup files are required to provide security against data loss. Ideally, backup storage is a high volume capacity at low cost.
Archived data may need to be stored for many years; so archival medium should be highly reliable, with no data decay over extended periods, and low cost.
The following factors have to be considered in storing the data in a particular medium:

- Volume of data
- Volatility of data
- Required speed of access to data
- Cost of data storage
- Reliability of data storage medium

Magnetic Technology

Magnetic technology is based on magnetization and demagnetization of spots on a magnetic recording surface. The same spot can be magnetized and demagnetized repeatedly. Magnetic recording materials may be coated on rigid plotters (hard disks), flexible circular substrates (floppy disks), thin ribbons of material (magnetic tapes), or rectangular sheets (magnetic cards).
The main advantages of magnetic technology are its relative maturity and widespread use. A major disadvantage is susceptibility to strong magnetic fields that can corrupt data stored on a disk. Also magnetization decays with time. Hence it is not a preferable medium to store legal documents, archival data.

Fixed Magnetic Disk

A fixed magnetic disk contains one or more recording surfaces that are permanently mounted in the disk drive and cannot be removed. Fixed disk is the medium of choice from personal computers to super computers. Fixed disks gives rapid, direct access to large volumes of data, and is ideal for highly volatile files. The major disadvantage of magnetic disk is the possibility of head crash that destroys the disk surface and data hence it is necessary to regularly make backup copies of hard disk files.

Removable Magnetic Disk

A removable disk comes in two formats: single disk and disk pack. Disk packs consist of multiple disks mounted together on a common spindle in a stack, usually on a disk drive with a retractable read/write heads. The disk's removability is its primary advantage making it ideal for backup.

Floppy Disk

Floppy low cost makes them ideal for storing and transporting small files and programs. But the reliability is not so good. A speck of dust can cause read error.

Magnetic Tape

In magnetic tape, the data storage and retrieval are in sequential manner.
Hence the access time, which refers to data access, is high. Magnetic tape was used extensively for archiving and backup in early database systems.

Redundant Array of Inexpensive Disk

RAID stands for Redundant Array of Inexpensive (Independent) Disk. A disk array comprises of several disks managed by a controller. Disks and controllers can be joined together in RAID combinations. First, they can provide fault tolerance by introducing redundancy across multiple disks. Second, they can

provide increased throughput because disk array controller supports parallel access to multiple disks. Instead of using one massive drive, RAID technology stores several smaller drives in one container.

Stripping: Stripping is an important concept for RAID storage. Stripping involves the allocation of physical records to different disks. A stripe is the set of physical records that can be read or written in parallel. Normally, a stripe contains a set of adjacent physical records.
The different types of disk arrays are known by their RAID Levels. Some of the RAID Levels are:

1. RAID Level 0+1
2. RAID Level 0
3. RAID Level 1
4. RAID Level 2
5. RAID Level 3
6. RAID Level 4
7. RAID Level 5
8. RAID Level 6
9. RAID Level 10
10. RAID Level 50

RAID Level 0+1

RAID Level 0+1 requires a minimum of four drives to implement. RAID 0+1 is implemented as mirrored arrays whose segments are
RAID 0 arrays. High input/output rates are achieved due to multiple stripe segments.

Disadvantages of RAID Level 0+1

The disadvantages of RAID Level 0+1 are:

- Limited scalability at a very high inherent cost.
- Very expensive/high overhead.
- A single drive failure will cause the whole array to become RAID Level 0 array.
- All drives must move in parallel to proper track lowering sustained performance.

Recommended Applications of RAID Level 0+1

The recommended applications of RAID Level 0+1 are:

- Imaging applications
- File server

RAID Level 0

RAID Level 0 requires a minimum of two drives to implement. RAID Level 0 implements a stripped disk array, the data are broken down into blocks and each block is written to a separate disk drive.

Advantages of RAID Level 0

The main advantages of RAID Level 0 are:

- Very simple design and easy to implement.
- No parity calculation overhead is involved.
- Best performance is achieved when data are striped across multiple controllers with only one drive per controller.
- Input/output performance is greatly improved by spreading the input/output load across many channels and drives.

Drawbacks of RAID Level 0

- Some of the drawbacks of RAID Level 0 are:

- RAID Level 0 is not a “true” RAID because it is not fault-tolerant.
- The failure of just one drive will result in all data in an array being lost.
- RAID Level 0 should never be used in mission critical environments.

Recommended Applications of RAID Level 0

- Image, video editing
- Prepress applications
- Applications that require high bandwidth

RAID Level 1

RAID Level 1 requires a minimum of two drives to implement. The characteristics of RAID Level 1 are mirroring and duplexing.

Advantages of RAID Level 1

The main advantages of RAID Level 1 are:

- Simplest RAID storage subsystem design.
- Under certain circumstances, RAID 1 can sustain multiple simultaneous drive failures.
- One hundred percent redundancy of data means no rebuild is necessary in case of a disk failure, just a copy to the replacement disk.

Disadvantages of RAID Level 1

Some of the drawbacks of RAID Level 1 are:

- Highest disk overhead.
- May not support hot swap of failed disk when implemented in “software.”
- Hardware implementation is strongly recommended.

Recommended Applications of RAID Level 1

- Accounting
- Payroll
- Financial
- Any application requiring high availability

RAID Level 2

A RAID Level 2 system would normally have as many data disks as the word size of the computer, typically 32. In addition, RAID 2 requires the use of extra disks to store an error-correcting code for redundancy. With 32 data disks, a RAID 2 system would require seven additional disks for a Hamming-code ECC.

For a number of reasons, including the fact that modern disk drives contain their own internal ECC, RAID 2 is not a practical disk array scheme.

Advantages of RAID Level 2

The main advantages of RAID Level 2 are:

- Extremely high data transfer rates possible.
- Relatively simple controller design compared to RAID Levels 3–5.

Disadvantages of RAID Level 2

Some of the disadvantages of RAID Level 2 are:

- Entry level cost is very high.

- No practical use; same performance can be achieved by RAID 3 at lower cost.

RAID Level 3

RAID Level 3 is characterized by parallel transfer with parity. The idea of parallel transfer with parity .In RAID Level 1, data are stripped (subdivided) and written on the data disks. Stripe parity is generated on Writes, recorded on the parity disk, and checked on Reads. RAID Level 3 requires a minimum of three drives to implement.

Advantages of RAID Level 3

The advantages of RAID Level 3 are:

- Very high data transfer rate.
- Disk failure has an insignificant impact on throughput.
- High efficiency because of low ratio of parity disks to data disks.

Disadvantages of RAID Level 3

- Controller design is fairly complex.
- Transaction rate is equal to that of a single disk drive at best.
- Very difficult and resource intensive to do as a “software” RAID.

Recommended Applications

- Video production and live streaming
- Image editing, video editing
- Any application requiring high throughput

RAID Level 4

RAID Level 4 is characterized by independent data disks with shared parity disks. Each entire block is written onto a data disk.

Parity for same rank blocks is generated on Writes, recorded on the parity disk, and checked on Reads. RAID Level 4 requires a minimum of three drives to implement.

Advantages of RAID Level 4

Some of the advantages of RAID Level 4 are:

- Very high Read data transaction rate.
- Low ratio of parity disks to data disks means high efficiency.
- High aggregate Read transfer rate.

Disadvantages of RAID Level 4

Some of the disadvantages of RAID Level 4 are:

- Quite complex controller design.
- Worst write transaction rate and Write aggregate transfer rate.
- Difficult and inefficient data rebuild in the event of disk failure.

RAID Level 5

In RAID Level 5, each entire data block is written on a data disk, parity for blocks in the same rank is generated on Writes, recorded in a distributed location and checked on Reads. RAID Level 5 requires a minimum of three drives to implement. RAID Level 5 is characterized by independent data disks with distributed parity blocks.

Advantages of RAID Level 5

The main advantages of RAID Level 5 are:

- Highest Read transaction rate

- Medium Write data transaction rate
- Good aggregate transfer rate

Disadvantages of RAID Level 5

Some of the disadvantages of RAID Level 5 are:

- Most complex controller design
- Difficult to rebuild in the event of disk failure
- Individual block transfer rate is same as single disk

Recommended Applications

- File and application servers
- Database servers
- Intranet servers

RAID Level 6

RAID Level 6 is characterized by independent data disks with two independent distributed parity schemes. Two independent parity computations must be used in order to provide protection against double disk failure. Two different algorithms are employed to achieve this purpose. RAID Level 6 requires a minimum of four drives to implement.

Advantages of RAID Level 6

The main advantages of RAID Level 6 are:

- RAID Level 6 provides high fault tolerance and can sustain multiple simultaneous drive failures.
- Perfect solution for critical applications.

Drawbacks of RAID Level 6

Some of the drawbacks of RAID Level 6 are:

- More complex controller design.
- Controller overhead to compute parity addresses is extremely high.

Recommended Applications

- File and application servers
- Web and E-mail servers
- Intranet servers

RAID Level 10

RAID Level 10 has very high reliability combined with high performance.
RAID Level 10 is implemented as a stripped array whose segments are RAID 1 arrays.

Advantages of RAID Level 10

The main advantages of RAID Level 10 are:

- High input/output rates are achieved by striping RAID 1 segments.

Drawbacks of RAID Level 10

Some of the drawbacks of RAID Level 10 are:

- Very expensive/high overhead
- Very limited scalability at a very high inherent cost

Recommended Application

- Database server requiring high performance and fault tolerance

Software-Based RAID

Primarily used with entry-level servers, software-based arrays rely on a standard host adapter and execute all I/O commands and mathematically intensive RAID algorithms in the host server CPU. This can slow system performance by increasing host PCI bus traffic, CPU utilization, and CPU interrupts. Some network operating system (NOS) such as NetWare and Windows NT include embedded RAID software. The chief advantage of this embedded RAID software has been its lower cost compared to higher-priced RAID alternatives. However, this advantage is disappearing with the advent of lower-cost, bus-based array adapters. The major advantages are low cost and it requires only a standard controller.

Hardware-Based RAID

Unlike software-based arrays, bus-based array adapters/controllers plug into a host bus slot (typically a 133 MByte (MB) s−1 PCI bus) and offload some or all of the I/O commands and RAID operations to one or more secondary processors. Originally used only with mid- to high-end servers due to cost, lower-cost bus-based array adapters are now available specifically for entry-level server network applications.

RAID Controller

The RAID controller is a device in which servers and storage intersect. The controller can be internal to the server, in which case it is a card or chip, or external, in which case it is an independent enclosure, such as a network – attached storage (NAS). In either case, the RAID controller manages the physical storage units in a RAID system and delivers them to the server in logical units.

While a RAID controller is almost never purchased separately from the RAID itself, the controller is a vital piece of the puzzle and therefore not as much a commodity purchase as the array.

In addition to offering the fault-tolerant benefits of RAID, bus-based array adapters/controllers perform connectivity functions that are similar to standard host adapters. By residing directly on a host PCI bus, they provide the highest performance of all array types. Bus-based arrays also deliver more robust fault-tolerant features than embedded NOS RAID software.

Types of Hardware RAID

There are two main types of hardware RAID, differing primarily in how they interface the array to the system.

Bus-Based or Controller Card Hardware RAID

This is the more conventional type of hardware RAID, and the type most commonly used, particularly for lower-end systems. A specialized RAID controller is installed into the PC or server, and the array drives are connected to it. It essentially takes the place of the small computer system interface (SCSI) host adapter or integrated development environment (IDE) controller that would normally be used for interfacing between the system and the hard disks; it interfaces to the drives using SCSI or IDE/ATA, and sends data to the rest of the PC over the system bus. Some motherboards, particularly those intended for server systems, come with some variant of integrated RAID controller.

These are built into the motherboard, but function is precisely the same manner as an add-in bus-based card. (This is analogous to the way that the integrated IDE/ATA controllers on all modern motherboards function the same way that add-in IDE/ATA controllers once did on older systems.) The only difference is that integrated controllers can reduce overall cost at the price of flexibility.

Intelligent, External RAID Controller

In this higher-end design, the RAID controller is removed completely from the system to a separate box. Within the box the RAID controller manages the drives in the array, typically using SCSI, and then presents the logical drives of the array over a standard interface (again, typically a variant of SCSI) to the

server using the array. The server sees the array or arrays as just one or more very fast hard disks; the RAID is completely hidden from the machine.
In essence, one of these units really is an entire computer unto itself, with a dedicated processor that manages the RAID array and acts as a conduit between the server and the array.
Advantages are data protection and performance benefits of RAID and more robust fault-tolerant features and increased performance vs. software – based RAID.

Optical Technology

Optical storage systems work by reflecting beams of laser light off a rotating disk with a minutely pitted surface. As the disk rotates, the amount of light reflected back to a sensor varies, generating a stream of ones and zeros. The advantages of optical technology are high storage densities, low cost media, and direct access.
There are four storage media based on optical technology. They are
CD-ROM, WORM, magneto-optical, and DVD. Optical technology is highly reliable because it is not susceptible to head crashes.
CD-ROM. CD-ROM stands for compact disk-read only memory. CD-ROM is a compact, robust, high capacity medium for the storage of permanent data.
Once the data are written, it cannot be altered.
CD-R: CD-R stands for CD-recordable. CD-R writers are used to prepare disks for CD mastering, test prototype applications, and backup systems.
CD-RW: CD-RW stands for CD-rewritable. This format allows erasing and rewriting to a disk many times.
WORM: WORM stands for write once read many. WORM is the major storage device for images. Information once written to a blank disk cannot be altered. WORM jukeboxes are used to store high volumes of data. A jukebox may contain up to 2,000 WORM disks. A WORM jukebox makes terabytes of data available in about 10 s.

Advantages of Optical Disks
The main advantages of optical disk are given below:

- Physical. An optical disk is much sturdier than tape or a floppy disk. It is physically harder to break, melt, or warp.
- Delicacy. It is not sensitive to being touched, though it can get too dirty or scratched to be read. It can be cleaned.
- Magnetic. It is entirely unaffected by magnetic fields.
- Capacity. Optical disks hold much more data than floppy disks.

Disadvantages of Optical Disks
Some of the disadvantages of optical disks are:

- Cost. The cost of the optical disk is high. But due to the advancement in technology, the price has come down drastically. Hence cost cannot be considered as drawback.
- Duplication. It is not easy to copy an optical disk as it is a floppy disk.
  Software and hardware is necessary for writing disks. This is balanced by the fact that it is not as necessary to have extra copies since the disk is so much sturdier than other media.

DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Introduction

A major motivation behind the development of database systems is the desire to integrate the operational data of an organization and to provide controlled access to the data. Although integration and controlled access may imply centralization, this is not the intention. In fact, the development of computer networks promotes a decentralized mode of work. This decentralized approach mirrors the organizational structure of many companies, which are logically distributed into divisions, departments, projects, and so on, and physically distributed into offices, plants, factories, where each unit maintains its own operational data. The shareability of the data and the efficiency of data access should be improved by the development of a distributed database system that reflects this organizational structure, makes the data in all units accessible, and stores data proximate to the location where it is most frequently used.

Concepts

i. **Distributed database** is a logically interrelated collection of shared data (and a description of this data) physically distributed over a computer network.

ii. **Distributed DBMS** is the software system that permits the management of the distributed database and makes the distribution transparent to users.

A Distributed Database Management System (DDBMS) consists of a single logical database that is split into a number of fragments. Each fragment is stored on one or more computers under the control of a separate DBMS, with the computers connected by a communications network. Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing data stored on other computers in the network. Users access the distributed database via applications, which are classified as those that do not require data from other sites (local applications) and those that do require data from other sites (global applications). We require a DDBMS to have at least one global application. A DDBMS therefore has the following characteristics:

- a collection of logically related shared data;
- the data is split into a number of fragments;
- fragments may be replicated;
- fragments/replicas are allocated to sites;
- the sites are linked by a communications network;
- the data at each site is under the control of a DBMS;
- the DBMS at each site can handle local applications, autonomously;
- Each DBMS participates in at least one global application.

From the definition of the DDBMS, the system is expected to make the distribution **transparent** (invisible) to the user. Thus, the fact that a distributed database is split into fragments that can be stored on different computers and perhaps replicated should be hidden from the user. The objective of transparency is to make the distributed system appear like a centralized system. This is sometimes referred to as the **fundamental principle** of distributed DBMSs.

iii. Distributed processing

It is important to make a distinction between a distributed DBMS and distributed processing.
Distributed processing is a centralized database that can be accessed over a computer network.
The key point with the definition of a distributed DBMS is that the system consists of data that is physically distributed across a number of sites in the network. If the data is centralized, even though other users may be accessing the data over the network, we do not consider this to be a distributed DBMS, simply distributed processing.

iv. Parallel DBMSs

We also make a distinction between a distributed DBMS and a parallel DBMS.
Parallel DBMS is a DBMS running across multiple processors and disks that is designed to execute operations in parallel, whenever possible, in order to improve performance.
Parallel DBMSs are again based on the premise that single processor systems can no longer meet the growing requirements for cost-effective scalability, reliability, and performance. A powerful and financially attractive alternative to a single-processor-driven DBMS is a parallel DBMS driven by multiple processors. Parallel DBMSs link multiple, smaller machines to achieve the same throughput as a single, larger machine, often with greater scalability and reliability than single-processor DBMSs.

To provide multiple processors with common access to a single database, a parallel DBMS must provide for shared resource management. Which resources are shared and how those shared resources are implemented, directly affects the performance and scalability of the system which, in turn, determines its appropriateness for a given application/environment. The three main architectures for parallel DBMSs are:

- shared memory;
- shared disk;
- Shared nothing.

Shared memory is a tightly coupled architecture in which multiple processors within a single system share system memory. Known as symmetric multiprocessing (SMP), this approach has become popular on platforms ranging from personal workstations that support a few microprocessors in parallel, to large RISC (Reduced Instruction Set Computer) - based machines, all the way up to the largest mainframes. This architecture provides high-speed data access for a limited number of processors, but it is not scalable beyond about 64 processors when the interconnection network becomes a bottleneck.

Shared disk is a loosely-coupled architecture optimized for applications that are inherently centralized and require high availability and performance. Each processor can access all disks directly, but each has its own private memory. Like the shared nothing architecture, the shared disk architecture eliminates the shared memory performance bottleneck.
Unlike the shared nothing architecture, however, the shared disk architecture eliminates this bottleneck without introducing the overhead associated with physically partitioned data. Shared disk systems are sometimes referred to as **clusters**.

Shared nothing, often known as massively parallel processing (MPP), is a multiple processor architecture in which each processor is part of a complete system, with its own memory and disk storage. The database is partitioned among all the disks on each system associated with the database, and data is transparently available to users on all systems.
This architecture is more scalable than shared memory and can easily support a large number of processors. However, performance is optimal only when requested data is stored locally.
While the shared nothing definition sometimes includes distributed DBMSs, the distribution of data in a parallel DBMS is based solely on performance considerations.

Advantages and Disadvantages of DDBMSs

The distribution of data and applications has potential advantages over traditional centralized database systems. Unfortunately, there are also disadvantages.

Advantages

i. Reflects organizational structure

Many organizations are naturally distributed over several locations. For example,
KCB has many offices in different cities. It is natural for databases used in such an application to be distributed over these locations. *KCB* may keep a database at each branch office containing details of such

things as the staff who work at that location, the accounts for clients, and the clients. The staff at a branch office will make local inquiries of the database. The company headquarters may wish to make global inquiries involving the access of data at all or a number of branches.

ii. Improved shareability and local autonomy

The geographical distribution of an organization can be reflected in the distribution of the data; users at one site can access data stored at other sites. Data can be placed at the site close to the users who normally use that data. In this way, users have local control of the data and they can consequently establish and enforce local policies regarding the use of this data. A global database administrator (DBA) is responsible for the entire system.

Generally, part of this responsibility is devolved to the local level, so that the local DBA can manage the local DBMS.

iii. Improved availability

In a centralized DBMS, a computer failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS, or a failure of a communication link making some sites inaccessible, does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures. If a single node fails, the system may be able to reroute the failed node's requests to another site.

iv. Improved reliability

As data may be replicated so that it exists at more than one site, the failure of a node or a communication link does not necessarily make the data inaccessible.

v. Improved performance

As the data is located near the site of 'greatest demand', and given the inherent parallelism of distributed DBMSs, speed of database access may be better than that achievable from a remote centralized database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralized DBMS.

vi. Economics

In the 1960s, computing power was calculated according to the square of the costs of the equipment: three times the cost would provide nine times the power. This was known as *Grosch's Law*. However, it is now generally accepted that it costs much less to create a system of smaller computers with the equivalent power of a single large computer. This makes it more cost-effective for corporate divisions and departments to obtain separate computers. It is also much more cost-effective to add workstations to a network than to update a mainframe system.

The second potential cost saving occurs where databases are geographically remote and the applications require access to distributed data. In such cases, owing to the relative expense of data being transmitted across the network as opposed to the cost of local access, it may be much more economical to partition the application and perform the processing locally at each site.

vii. Modular growth

In a distributed environment, it is much easier to handle expansion. New sites can be added to the network without affecting the operations of other sites. This flexibility allows an organization to expand relatively easily. Increasing database size can usually be handled by adding processing and storage power to the network. In a centralized DBMS, growth may entail changes to both hardware (the procurement of a more powerful system) and software (the procurement of a more powerful or more configurable DBMS).

viii. Integration

At the start of this section we noted that integration was a key advantage of the DBMS approach, not centralization. The integration of legacy systems is one particular example that demonstrates how some

organizations are forced to rely on distributed data processing to allow their legacy systems to coexist with their more modern systems. At the same time, no one package can provide all the functionality that an organization requires nowadays.
Thus, it is important for organizations to be able to integrate software components from different vendors to meet their specific requirements.

ix. Remaining competitive

There are a number of relatively recent developments that rely heavily on distributed database technology such as e-Business, computer-supported collaborative work, and workflow management. Many enterprises have had to reorganize their businesses and use distributed database technology to remain competitive. For example, while more people open accounts just because the Internet exists, *KCB* may lose some of its market share if it does not allow clients to access their accounts online now.

Disadvantages

1) Complexity

A distributed DBMS that hides the distributed nature from the user and provides an acceptable level of performance, reliability, and availability is inherently more complex than a centralized DBMS. The fact that data can be replicated also adds an extra level of complexity to the distributed DBMS. If the software does not handle data replication adequately, there will be degradation in availability, reliability, and performance compared with the centralized system, and the advantages we cited above will become disadvantages.

2) Cost

Increased complexity means that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralized DBMS. Furthermore, a distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional labor costs to manage and maintain the local DBMSs and the underlying network.

3) Security

In a centralized system, access to the data can be easily controlled. However, in a distributed DBMS not only does access to replicated data have to be controlled in multiple locations, but the network itself has to be made secure. In the past, networks were regarded as an insecure communication medium. Although this is still partially true, significant developments have been made to make networks more secure.

4) Integrity control more difficult

Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Enforcing integrity constraints generally requires access to a large amount of data that defines the constraint but which is not involved in the actual update operation itself. In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be prohibitive.

5) Lack of standards

Although distributed DBMSs depend on effective communication, we are only now starting to see the appearance of standard communication and data access protocols. This lack of standards has significantly limited the potential of distributed DBMSs. There are also no tools or methodologies to help users convert a centralized DBMS into a distributed DBMS.

6) Lack of experience

General-purpose distributed DBMSs have not been widely accepted, although many of the protocols and problems are well understood. Consequently, we do not yet have the same level of experience in industry

as we have with centralized DBMSs. For a prospective adopter of this technology, this may be a significant deterrent.

7) Database design more complex

Besides the normal difficulties of designing a centralized database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication.

Homogeneous and Heterogeneous DDBMSs

A DDBMS may be classified as homogeneous or heterogeneous. In a homogeneous system, all sites use the same DBMS product. In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical, and object-oriented DBMSs.

Homogeneous systems are much easier to design and manage. This approach provides incremental growth, making the addition of a new site to the DDBMS easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.

Heterogeneous systems usually result when individual sites have implemented their own databases and integration is considered at a later stage. In a heterogeneous system, translations are required to allow communication between different DBMSs. To provide DBMS transparency, users must be able to make requests in the language of the DBMS at their local site. The system then has the task of locating the data and performing any necessary translation. Data may be required from another site that may have:

- different hardware;
- different DBMS products;
- Different hardware and different DBMS products.

If the hardware is different but the DBMS products are the same, the translation is straightforward, involving the change of codes and word lengths. If the DBMS products are different, the translation is complicated involving the mapping of data structures in one data model to the equivalent data structures in another data model. For example, relations in the relational data model are mapped to records and sets in the network model.

The typical solution used by some relational systems that are part of a heterogeneous DDBMS is to use **gateways**, which convert the language and model of each different DBMS into the language and model of the relational system. However, the gateway approach has some serious limitations. First, it may not support transaction management, even for a pair of systems; in other words, the gateway between two systems may be only a query translator. For example, a system may not coordinate concurrency control and recovery of transactions that involve updates to the pair of databases. Second, the gateway approach is concerned only with the problem of translating a query expressed in one language into an equivalent expression in another language. As such, generally it does not address the issues of homogenizing the structural and representational differences between different schemas.

Functions of a DDBMS

We expect a DDBMS to have at least the functionality for a centralized. In addition, we expect a DDBMS to have the following functionality:

- extended communication services to provide access to remote sites and allow the transfer of queries and data among the sites using a network;
- extended system catalog to store data distribution details;
- distributed query processing, including query optimization and remote data access;
- extended security control to maintain appropriate authorization/access privileges to the distributed data;
- extended concurrency control to maintain consistency of distributed and possibly replicated data;
- Extended recovery services to take account of failures of individual sites and the failures of communication links.

Component Architecture for a DDBMS

Independent of the reference architecture, we can identify a component architecture for a DDBMS consisting of four major components:

- local DBMS (LDBMS) component;
- data communications (DC) component;
- global system catalog (GSC);
- Distributed DBMS (DDBMS) component.

Local DBMS component

The LDBMS component is a standard DBMS, responsible for controlling the local data at each site that has a database. It has its own local system catalog that stores information about the data held at that site. In a homogeneous system, the LDBMS component is the same product, replicated at each site. In a heterogeneous system, there would be at least two sites with different DBMS products and/or platforms.

Data communications component

The DC component is the software that enables all sites to communicate with each other.
The DC component contains information about the sites and the links.

Global system catalog

The GSC has the same functionality as the system catalog of a centralized system. The GSC holds information specific to the distributed nature of the system, such as the fragmentation, replication, and allocation schemas. It can itself be managed as a distributed database and so it can be fragmented and distributed, fully replicated, or centralized, like any other relation, as we discuss below. A fully replicated GSC compromises site autonomy as every modification to the GSC has to be communicated to all other sites. A centralized GSC also compromises site autonomy and is vulnerable to failure of the central site.

More terms:

- *Fragmentation* A relation may be divided into a number of subrelations, called **fragments**, which are then distributed. There are two main types of fragmentation: **horizontal** and **vertical**. Horizontal fragments are subsets of tuples and vertical fragments are subsets of attributes.
- *Allocation* Each fragment is stored at the site with 'optimal' distribution.
- *Replication* The DDBMS may maintain a copy of a fragment at several different sites.

Date's Twelve Rules for a DDBMS

Fundamental principle

To the user, a distributed system should look exactly like a non-distributed system.

1) Local autonomy

The sites in a distributed system should be autonomous. In this context, autonomy means that:

- local data is locally owned and managed;
- local operations remain purely local;
- All operations at a given site are controlled by that site.

2) No reliance on a central site

There should be no one site without which the system cannot operate. This implies that there should be no central servers for services such as transaction management, deadlock detection, query optimization, and management of the global system catalog.

3) Continuous operation

Ideally, there should never be a need for a planned system shutdown, for operations such as:

- adding or removing a site from the system;

- the dynamic creation and deletion of fragments at one or more sites.

4) Location independence

Location independence is equivalent to location transparency. The user should be able to access the database from any site. Furthermore, the user should be able to access all data as if it were stored at the user's site, no matter where it is physically stored.

5) Fragmentation independence

The user should be able to access the data, no matter how it is fragmented.

6) Replication independence

The user should be unaware that data has been replicated. Thus, the user should not be able to access a particular copy of a data item directly, nor should the user have to specifically update all copies of a data item.

7) Distributed query processing

The system should be capable of processing queries that reference data at more than one site.

8) Distributed transaction processing

The system should support the transaction as the unit of recovery. The system should ensure that both global and local transactions conform to the ACID rules for transactions, namely: atomicity, consistency, isolation, and durability.

9) Hardware independence

It should be possible to run the DDBMS on a variety of hardware platforms.

10) Operating system independence

As a corollary to the previous rule, it should be possible to run the DDBMS on a variety of operating systems.

11) Network independence

Again, it should be possible to run the DDBMS on a variety of disparate communication networks.

12) Database independence

It should be possible to have a DDBMS made up of different local DBMSs, perhaps supporting different underlying data models. In other words, the system should support heterogeneity.

Data Mining and Data Warehousing

Data Mining

Introduction

Data mining refers to extracting or "mining" knowledge from large amounts of data. The data mining is appropriately named as "Knowledge Mining." There are many other terms carrying a similar or slightly different meaning to data mining, such as Knowledge Mining from Databases, Knowledge Extraction, Data/Pattern Analysis, Data Archaeology, and Data dredge.

Data mining is an essential process where intelligent methods are applied in order to extract data patterns. Data mining is the process of discovering interesting knowledge from large amounts of data stored in databases, data warehouses, or other information repositories.

Architecture of Data Mining Systems

Data mining is an essential process where intelligent methods are applied in order to extract data patterns. The architecture of data mining and the major components are described as follows.

Data Warehouse: This is one or a set of databases, spreadsheets, or other kinds of information repositories. Data cleaning and Data integration techniques may be performed on the data.
Database: The database server is responsible for fetching the relevant data based on the user data-mining request.
Knowledge Base: This can be used to guide the search, or evaluate the interestingness of the resulting patterns. Such knowledge include concept hierarchies, used to organize attributes or attribute values into different levels of abstraction, knowledge such as user beliefs, which can be used to assess the pattern interestingness based on its unexpectedness may also be included.
Data Mining Engine: This is essential to the data mining system and ideally consists of set of functional modules for tasks such as characterization, association, classification, cluster analysis, evaluation, and deviation analysis.
Pattern Evaluation Modules: This component typically employs interestingness measures and interacts with the data mining modules so as to focus the search toward interesting patterns. It may use interestingness thresholds to filter out discover patterns. Alternatively, this module may be integrated with the mining module depending on the implementation of the data mining method used.

Graphical User Interface: This module communicates between the user and the data mining system, allowing the user to interact with the system by specifying the data mining query or task, providing the information to help focus the search, and performing exploratory data mining based on the intermediate data mining results.

Data Mining Functionalities
Functionalities of data mining are used to specify the kind of patterns to be found in data mining tasks. It can be classified into two categories such as Descriptive and Predictive. Descriptive mining task characterize the general properties of data in the database, whereas predictive mining task perform inference on the current data in order to make predictions.
These functionalities are classified as follows:

- Characterization and discrimination
- Association analysis
- Classification and prediction
- Cluster analysis
- Outlier analysis
- Evolution analysis

Classification of Data Mining Systems
Data mining is an interdisciplinary field, the confluence of set of disciplines, including database system statistics, machine learning, visualization, and information science. Moreover, depending on the data mining approach used, techniques from other disciplines may be applied. Data mining research is expected to generate a large variety of data mining systems. It can be described as follows.

Classification According to the Kinds of Database Mined
Database system themselves can be classified according to different criteria such as data models, each of which may require its own data mining techniques.
If classifying according to the special types of data handled, we may have a spatial, time series, text, or worldwide mining system.

Classification According to the Kinds of Knowledge Mined
It can be categorized according to the kinds of knowledge they mine, i.e., based on data mining functionalities, such as characterization, discrimination, association, classification, clustering, cluster

outlier analysis, and evolution analysis. It can be based on granularity or levels of abstraction of the knowledge mined.

Classification According to the Kinds of Techniques Utilized
Data mining techniques can be categorized according to the degree of user interaction involved or methods of data analysis employed. A sophisticated data mining system will often adopt multiple data mining techniques or work out an effective integrated technique that combines the merits of a few individual approaches.

Major Issues in Data Mining
Major issues in data mining are mining methodology, user interaction, performance, and diverse data types. These are described as follows.

Mining Methodologies and User Interaction Issues
These reflect the kind of knowledge mine, the ability to mined knowledge at multiple granularities, the user domain knowledge, and knowledge visualization.

Mining Different Kind of Knowledge in Database
Since different users can be interested in different kinds of knowledge, data mining should cover a wide spectrum of data analysis and knowledge discovery task, including data characterization discrimination, association, classification, clustering, trend and deviation analysis, and similarity analysis. These tasks may be used in the same database in different ways and require the development of numerous data mining techniques.

Incorporation of Background Knowledge
Background knowledge or information regarding the domain under study may be used to guide the discovery process and allows discovered patterns to be expressed in concise terms and at different levels of abstraction.

Presentation and Visualization of Data Mining Results
Discover knowledge should be expressed in high-level languages, visual representations, or other expressive forms so that knowledge can be easily understood and directly used by humans. This is especially crucial if the data mining system is to be interactive.

Handling Noisy or Incomplete Data
The data stored in database may reflect noise, exceptional cases, or incomplete data objects. When mining data regularities, these objects may confuse the process, causing knowledge model constructed to over fit the data. As a result, the accuracy of the discovered pattern can be poor.

Performance Issues
The performance issues in data mining include efficiency, scalability, and parallelization of data mining algorithms.

Efficiency and Scalability of Data Mining Algorithms
To effectively extract information from a huge amount of data in databases, data mining algorithm must be efficient and scalable. Many of the issues are followed under mining methodology, and user interaction must consider efficiency and scalability.

Parallel, Distributed, and Incremental Mining Algorithms

The huge size of many databases, the wide distribution of data, and computational complexity of some data mining methods are factors motivating the development of parallel and distributed data mining algorithm. Such algorithms divide the data into partitions, which are processed in parallel. The results from the partitions are then merged. Therefore, this algorithm performs the knowledge modification incrementally to amend and strengthened.

Issues Relating to the Diversity of Database Types
The main issues related to the diversity of the database types are handling of relational and complex types of data and mining information from heterogeneous database.

Handling of Relational and Complex Types of Data
Relational databases are widely used, the development of efficient and effective data mining systems for such data is important. However, other database may contain complex data object, hypertext and multimedia data, spatial data, temporal data, or transaction data. It is unrealistic to expect one system to mine all kinds of data, given the diversity of data types and different goals of data mining.

Mining Information from Heterogeneous Database and Global Information Systems
LAN connects many sources of data, forming huge, distributed, and heterogeneous databases. The discovery of knowledge from different sources of structure and semi structured or unstructured data with diverse data semantics poses great challenges to data mining. Web mining, which uncovers interesting knowledge about Web contents, Web usage became a very challenging and highly dynamic field in data mining.

Data Preprocessing
Today's real world databases are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size, often several giga bytes or more. To improve the quality of the data and efficiency, data preprocessing is introduced.
Real world data tends to be dirty incomplete and inconsistent. This technique can improve the quality of data, thereby improving accuracy and efficiency of the subsequent data mining process. It is an important step in the knowledge discovery process. Since quality decisions must be based on quality data. Detecting data anomalies, rectifying them early, and reducing the data to be analyzed can lead to huge payoffs for decision making.
There are a number of data preprocessing techniques. They are:

- Data Cleaning
- Data Integration
- Data Transformation
- Data Reduction

Data Cleaning
Data cleaning routines attempt to fill in the missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data.
Noisy Data
Noise is a random error or variance in a measured variable. Smooth out the data to remove the noise. The smoothing techniques are as follows:

- *Binning*: This method smoothen a sorted data value by consulting its "neighborhood," that is, the values around it. The sorted values are distributed into a number of "buckets," or bins:
  A. Smoothing by bin means each value in a bin is replaced by the mean value of the bin.
  B. Smoothing by bin medians means each bin value is replaced by bin median.

- *Clustering*: Outliers may be detected by clustering, where similar values are organized into groups or clusters. The values outside the set of clusters are outliers.
- *Combined Computer and Human Inspection*. Outliers may be identified through a combination of computer and human inspection. A human can sort through the patterns in the list to identify the actual garbage ones (e.g., Miss Labeled Character). This is much faster than manually searching through the entire database.
- *Regressions*. Data can be smoothed by fitting the data to a function such as with regression.
    - **i.** *Linear Regression*: This involves finding the "best" line to fit two variables so that one variable can be used to predict the other.
    - **ii.** Multiple Linear Regression. It is an extension of linear regressions, where more than two variables are involved and the data are fit to a multidimensional surface.

Inconsistent Data

There may be inconsistencies in the data recorded for some transactions. Some data inconsistencies may be corrected manually using external references.

Data Integration

Data mining often requires data integration, the merging of data from multiple data stores. There are a number of issues to consider during data integration, which combines data from multiple sources into a coherent data store. These sources may include multiple databases, data cubes, or flat files. The issues in data integration are:

- **A.** *Schema Integration*: It is referred to as entity identification problem. So the databases typically have metadata, that is, data about the data. The metadata can be used to help avoid errors in Schema integration.
- **B.** *Redundancy*: An attribute may be redundant if it can be derived from another table. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set. Some redundancies can be detected by correlation analysis. The correlation between attributes A and B can be measured by:

$$r_{A,B} = \sum \frac{(A - \bar{A})(B - \bar{B})}{n - 1\sigma_A\sigma_B}$$

Where n is the number of tuples, A and B are the respective mean values of A and B, and σ

Careful integration of the data from multiple sources can help reduce and avoid redundancies and inconsistencies in the resulting data set.

Data Transformation

In data transformation, the data are transformed or consolidated into forms appropriate for mining. Data transformations can involve the following:

1. Smoothing. Smoothing works to remove the noise from data, such technique include binning, clustering, and regression.
2. Aggregation. Aggregation operations are applied to the data, this step is typically used in constructing a data cube for analysis of the data at multiple granularities.
3. Generalization. Generalization of the data, where low-level or "primitive" data are replaced by higher-level concepts through the use of concept hierarchies.
4. Normalization. Normalization implies that the attribute data are scaled so as to fall within a small specified range, such as –1.0–1.0 or 0.0–1.0
5. Attribute Construction. In attribute construction, new attributes are constructed and added from the given set of attributes to help the mining process.

Data Reduction

Data reduction technique can be applied to obtain a reduced representation of the data set that is much smaller in volume. It maintains the integration of original data that is mining on the reduced data set more efficiently.
There are several types of data reduction, which are given as follows:

- **a)** Data Cube Aggregation. The aggregation operations are applied to the data in the construction of a data cube.
- **b)** Dimension Reduction. In dimension reduction, the irrelevant, weakly relevant or redundant attributes or dimensions may be detected and removed.
- **c)** Data Compression. In data compression, the encoding mechanisms are used to reduce the data set size.
- **d)** Numerosity Reduction. In numerosity reduction, the data are replaced or estimated by alternative, smaller data representations such as parametric models or nonparametric methods such as clustering, sampling, and use of histograms.
- **e)** Discretization and Concept Hierarchy Generation. The raw data values for attributes are replaced by ranges or higher conceptual levels. Concept hierarchies allow the mining of data at multiple levels of abstraction and are a powerful tool for data mining.

Data Mining Primitives
A popular misconception about data mining is to expect that data mining systems can autonomously dig out all of the valuable knowledge that is embedded in a given large database, without human intervention or guidance. It may be the first sound appealing to have an autonomous data mining system. A more realistic scenario is to expect that users can communicate with the data mining system using a set of data mining primitives designed in order to facilitate efficient and fruitful knowledge discovery. Such primitives include the specification of the portions of the database.
These primitives allow the user to interactively communicate with the data mining system during discovery in order to examine the findings from different angles or depths, and direct the mining process. A data mining query language can be designed to incorporate these primitives, allowing users to flexibly interact with data mining systems. Data mining query language provides a foundation on which user-friendly graphical interfaces can be built.

Data Mining Task
Data mining task can be specified in the form of a data mining query, which is input to the data mining system. A data mining query is defined in terms of the following primitives.

The Kinds of Knowledge to be mined
This specifies the data mining function to be performed, such as characterization, discrimination, association, classification, and clustering or evaluation analysis.
In addition to specifying the kind of knowledge to be mined for a given data mining task, the user can be more specific and provide pattern templates that all discovered patterns must match. These templates are Meta patterns (Meta rules or Meta queries) and can be used to guide the discovery process.

Background Knowledge
User can specify background knowledge or knowledge about the domain to be mined. This knowledge is useful for guiding the knowledge discovery process and evaluating the patterns found. There are several kinds of background knowledge such as concept hierarchies, schema hierarchies, and operation derived hierarchies.
Concept hierarchy defines a sequence of mapping from a set of low-level concepts to higher-level, more general concepts. It is useful to allow the data to be mined with multiple levels of abstraction.

Applications and Trends in Data Mining
Data mining is a young discipline with wide and diverse applications, there is still a nontrivial gap between general principles of data mining and domain specific for effective data mining tools for particular applications.

Data Mining for Biomedical and DNA Data Analysis
Biomedical research, ranges from the development of pharmaceutical and advances in cancer therapies, identification, and study of the human genome by discovering large-scale sequencing patterns. A gene is usually comprised of hundreds of individual nucleotides arranged in a particular order. There are almost an unlimited number of ways that the nucleotides can be ordered and sequenced to form distinct genes. Data Mining has become a powerful tool and contributes substantially to DNA analysis in the following ways.

Semantic Integration of Heterogeneous, Distributed Genome Databases
Due to the highly distributed, uncontrolled generation and use of a wide variety of DNA data, the semantic integration of such heterogeneous and widely distributed genome databases becomes an important task for systematic and coordinated analysis of DNA databases. Data cleaning and data integration methods will help the integration of genetic data and the construction of data warehouses for genetic data analysis.
Similarity Search and Comparison among DNA Sequences
One of the most important search problems in genetic analysis is similarity search and comparison among DNA sequences. The techniques needed here is quite different from that used for time series data: For example, data transformation methods such as scaling, normalization, and window stitching, which are popularly used in the analysis of time series data, are ineffective for genetic data.

Association Analysis: Identification of Co-occurring Gene Sequences
Association analysis methods can be used to help determine the kinds of genes that are likely to co-occur in target samples. Such analysis would facilitate the discovery of groups of genes and the study of interactions and relationships between them.

Path Analysis: Linking Genes to Different Stages of Disease Development
While a group of genes may contribute to a disease process, different genes may become active at different stages of the disease, therefore achieving more effective treatment of the disease. Such path analysis is expected to play an important role in genetic studies.

Data Mining for Financial Data Analysis
Financial data collected in the banking and financial industry is often relatively complete, reliable, and of high quality, which facilitates systematic data analysis and data mining. Here we present a few typical cases:
Loan payment prediction and customer credit policy analysis
Loan payment prediction and customer credit analysis are critical to the business of a bank. Many factors can strongly or weakly influence loan payment performance and customer credit rating. Data mining methods, such as feature selection and attribute relevance ranking, may help identify important factors and eliminate irrelevant ones.
Classification and clustering of customers for customer group identification targeted marketing
Customers with similar behaviors regarding banking and loan payments may be grouped together by multidimensional clustering techniques. Effective clustering and collaborative filtering methods can help identify customer groups, associate a new customer with an appropriate customer group, and facilitate targeted marketing.

Data Mining for the Retail Industry

The retail industry is a major application area for data mining since it collects huge amounts of data on sales, customer shopping history, goods transportation, consumption and service records, and so on. The quantity of data collected continues to expand rapidly, especially due to the increasing ease, availability, and popularity of business conducted on the Web; or e-commerce.

Retail data mining can help identify customer buying behaviors, discover customer shopping patterns and trend, improve the quality of customer service, achieve better customer retention and satisfaction, enhance goods consumption ratios, design more effective goods transportation and distribution policies, and reduce the cost of business.

A few examples of data mining in the retail industry are:

- Design and construction of data warehouses based on the benefits of data mining
- Multidimensional analysis of sales, customers, products, time, and religion
- Analysis of the effectiveness of sales campaigns
- Customer retention – analysis of customer loyalty
- Purchase recommendation and cross reference of items

Data Mining for the Telecommunication Industry

The telecommunication industry has quickly evolved from offering local and long distance telephone services for providing many other comprehensive communication services including voice, fax, pager, cellular phone, images, e-mail, computer and Web data transmission, and other data traffic. The integration of telecommunication, computer network, Internet, and numerous other means of communication and computing is also underway.

The following are a few scenarios where data mining may improve telecommunication services.

- Multidimensional analysis of telecommunication data
  Telecommunication data are intrinsically multidimensional with dimensions such as calling time, duration, and location of caller, location of called, and type of call. The multidimensional analysis of such data can be used to identify and compare the data traffic, system workload, resource usage, user group behavior, profit, and so on.
- Multidimensional association and sequential pattern analysis
  The discovery of association and sequential patterns in multidimensional analysis can be used to promote telecommunication services. The calling records may be grouped by customer in the following form:
  (Customer id, residence, office, time, date, service 1, service 2, . . .)
  A sequential pattern can help to promote the sales of specific long distance and cellular phone combinations, and improve the availability of particular services in the region.

Use of visualization tools in telecommunication data analysis

Tools for OLAP, linkage, association, clustering, and outlier visualization have been shown to be very useful for telecommunication and data analysis.

Data Warehousing

A Data Warehouse (DW) is a database that stores information oriented to satisfy decision-making requests. It is a database with some particular features concerning the data it contains and its utilization. A very frequent problem in enterprises is the impossibility for accessing to corporate, complete, and integrated information of the enterprise that can satisfy decision-making requests. A paradox occurs: data exists but information cannot be obtained.

In general, a DW is constructed with the goal of storing and providing all the relevant information that is generated along the different databases of an enterprise. A data warehouse helps turn data into information. In today's business world, data warehouses are increasingly being used to make strategic business decisions.

Goals of Data Warehousing

Data warehousing technology comprises a set of new concepts and tools which support the knowledge worker like executive, manager, and analyst with information material for decision making. The fundamental reason for building a data warehouse is to improve the quality of information in the organization.

The key issues are the provision of access to a company-wide view of data whenever it resides. Data coming from internal and external sources, existing in a variety of forms form traditional structural data to unstructured data like text files or multimedia is cleaned and integrated into a single repository. A data warehouse is the consistent store of this data which is made available to end users in a way they can understand and use in a business context.

The need for data warehousing originated in the mid-to-late 1980s with the fundamental recognition that information systems must be distinguished into operational and informational systems. Operational systems support the day-to-day conduct of the business, and are optimized for fast response time of predefined transactions, with a focus on update transactions. Operational data are a current and real-time representation of the business state. In contrast, informational systems are used to manage and control the business. They support the analysis of data for decision making about how the enterprise will operate now and in the future. They are designed mainly for ad hoc, complex, and mostly read-only queries over data obtained from a variety of sources. Information data are historical, i.e., they represent a stable view of the business over a period of time.

Limitations of current technology to bring together information from many disparate systems hinder the development of informational systems. Data warehousing technology aims at providing a solution for these problems.

Characteristics of Data in Data Warehouse

Data in the Data Warehouse is integrated from various, heterogeneous operational systems like database systems, flat files, etc. Before the integration, structural and semantic differences have to be reconciled, i.e., data have to be "homogenized" according to a uniform data model. Furthermore, data values from operational systems have to be cleaned in order to get correct data into the data warehouse. Since a data warehouse is used for decision making, it is important that the data in the warehouse be correct. However, large volumes of data from multiple sources are involved; there is a high probability of errors and anomalies in the data. Therefore, tools that help to detect data anomalies and correct them can have a high payoff. Some examples where data cleaning becomes necessary are: inconsistent field lengths, inconsistent descriptions, inconsistent value assignments, missing entries, and violation of integrity constraints.

The need to access historical data is one of the primary incentives for adopting the data warehouse approach. Historical data are necessary for business trend analysis which can be expressed in terms of understanding the differences between several views of the real-time data. Maintaining historical data means that periodical snapshots of the corresponding operational data are propagated and stored in the warehouse without overriding previous warehouse states. However, the potential volume of historical data and the associated storage costs must always be considered in relation to their business benefits.

Data warehouse contains usually additional data, not explicitly stored in the operational sources, but derived through some process from operational data. For example, operational sales data could be stored in several aggregation levels in the warehouse.

Data Warehouse Architectures

Data warehouses and their architectures vary depending upon the specifics of an organization's situation. Three common data warehouse architectures which are discussed in this section are:

- Basic Data Warehouse Architecture

- Data Warehouse Architecture with a Staging Area
- Data Warehouse Architecture with a Staging Area and Data Marts

Basic Data Warehouse Architecture
In basic data warehouse architecture end users directly access data derived from several source systems through data warehouse.
The data obtained from the warehouse can be used for analysis, reporting, and mining information. The data sources include operational system and flat files. Here a flat file is one in which the fields of records are simple atomic values.
Data Warehouse Architecture with a Staging Area
In this architecture, the operational data must be cleaned and processed before putting into the warehouse. This can be done programmatically although most data warehouses use a staging instead. A staging area simplifies building summaries and general warehouse management.

Data Warehouse Architecture with Staging Area and Data Marts
The basic difference between this architecture and the architecture discussed earlier is the inclusion of data mart. It is necessary to customize the data warehouse's architecture for different groups within an organization. This can be done by adding data marts, which are systems designed for a particular line of business. Let us illustrate this with an example where purchasing, sales, and inventories are separated. In this example, a financial analyst might want to analyze historical data for purchases and sales.

Data Mart
Data marts are complete logical subsets of the complete data warehouse. Data marts should be consistent in their data representation in order to assure Data Warehouse robustness. A data mart is a set of tables that focus on a single task. This may be for a department, such as production or maintenance department, or a single task such as handling customer products.

Metadata
In general metadata are defined as "data about data" or "data describing the meaning of data." In data warehousing, there are various types of metadata.
For example information about the operational sources, the structure and semantics of the data ware house data, the tasks performed during the construction, maintenance and access of a data ware house, etc. A data warehouse without adequate metadata is like "a filing cabinet stuffed with papers, but without any folders or labels." The quality of metadata and the resulting quality of information gained using a data warehouse solution are tightly linked.
In a data warehouse metadata are categorized into Business and Technical metadata. Business metadata describes what is in the ware house, its meaning in business terms. The business metadata lies above technical metadata, adding some more details to the extracted material. This type of metadata is important as it facilitates business users and increases the accessibility. In contrast, technical metadata describes the data elements as they exist in the ware house. These types of metadata are used for data modeling initially, and once the warehouse is erected this metadata are frequently used by warehouse administrator and software tools.
Implementing a concrete Data Warehouse architecture is a complex task comprising of two major phases. In the configuration phase, a conceptual view of the ware house is first specified according to user requirements which are often termed as data warehouse design. Then the involved data sources and the way data will be extracted and loaded into the warehouse is determined.
Finally, decisions about persistent storage of the warehouse using database technology and the various ways data will be accessed during analysis are made.