****

**Ambo University Woliso Campus**

**School of Technology and Informatics**

**Department of Information Technology**

Course title: - Advanced Database

**Module Name: -** Data and Information Management **Program: - Regular $ Weekend**

**Module code: - ITec-M2071 Target Group: - IT 2nd Reg $ 3rd Wknd**

**Course Name: - Advanced Database Semester: - II**

**Course code: - ITec-2072 Content: - Course Handout**

**Academic year: - 2020/2012 ECTS credits: - 7**

**Status of the course: - Core Instructor Name: - Husen Adem**

**Instructor Email: -** [**igguumuude@gmail.com**](mailto:igguumuude@gmail.com) **Phone Number: - +251925100878**

**Course Objectives**

***At the end of this course you will be able to****:*

* *Understand the database query processing and optimization*
* *Know the basics of transaction management*
* *Understand database security*
* *Use different recovery methods when there is a database failure Design a distributed database system in homogenous and heterogeneous environments*
* *Understand how to give privilege for the user of the database*



**Table of Contents**

[CHAPTER ONE 4](#_Toc41510247)

[QUERY PROCESSING AND OPTIMIZATION 4](#_Toc41510248)

[Relational Algebra 4](#_Toc41510249)

[TRANSLATING SQL QUERIES INTO RELATIONAL ALGEBRA 4](#_Toc41510250)

[QUERY PROCESSING 11](#_Toc41510251)

[Heuristic Query Tree Optimization 14](#_Toc41510252)

[Summary of Heuristics for Algebraic Optimization: 17](#_Toc41510253)

[Cost-based query optimization: 18](#_Toc41510254)

[Semantic Query Optimization: 19](#_Toc41510255)

[CHAPTER TWO 20](#_Toc41510256)

[DATABASE SECURITY AND AUTHORIZATION 20](#_Toc41510257)

[Introduction to Database Security Issues 20](#_Toc41510258)

[Authentication 20](#_Toc41510259)

[Authorization/Privilege 20](#_Toc41510260)

[Database Security and the DBA 21](#_Toc41510261)

[Comparing DAC and MAC 25](#_Toc41510262)

[Introduction to Statistical Database Security 25](#_Toc41510263)

[Types of Cryptosystems 26](#_Toc41510264)

[CHAPTER THREE 28](#_Toc41510265)

[TRANSACTION PROCESSING CONCEPTS 28](#_Toc41510266)

[Introduction to Transaction Processing 28](#_Toc41510267)

[ACID properties of the transactions 29](#_Toc41510268)

[SIMPLE MODEL OF A DATABASE 29](#_Toc41510269)

[Transaction Atomicity and Durability 30](#_Toc41510270)

[Serializability 33](#_Toc41510271)

[Transactions as SQL Statements 36](#_Toc41510272)

[Summary 38](#_Toc41510273)

[CHAPTER FOUR 40](#_Toc41510274)

[CONCURRENCY CONTROL TECHNIQUES 40](#_Toc41510275)

[Introduction to Concurrency control techniques 40](#_Toc41510276)

[The Two-Phase Locking Protocol 41](#_Toc41510277)

[Deadlock Detection and Recovery 43](#_Toc41510278)

[Recovery from Deadlock 43](#_Toc41510279)

[Timestamp-Based Protocols 44](#_Toc41510280)

[The Timestamp-Ordering Protocol 45](#_Toc41510281)

[Purpose of Concurrency Control 46](#_Toc41510282)

[CHAPTER FIVE 48](#_Toc41510283)

[DATABASE RECOVERY TECHNIQUES 48](#_Toc41510284)

[Recovery Outline and Categorization of Recovery Algorithms 48](#_Toc41510285)

[Write-Ahead Logging, Steal/No-Steal, and Force/No-Force 49](#_Toc41510286)

[Transaction Actions That Do Not Affect the Database 50](#_Toc41510287)

[Recovery Techniques Based on Immediate Update 51](#_Toc41510288)

[Shadow Paging 51](#_Toc41510289)

[The ARIES Recovery Algorithm 52](#_Toc41510290)

[Recovery in Multidata base Systems 53](#_Toc41510291)

[Database Backup and Recovery from Catastrophic Failures 53](#_Toc41510292)

[CHAPTER SIX 55](#_Toc41510293)

[DISTRIBUTED DATABASE SYSTEMS 55](#_Toc41510294)

[Distributed Database Concepts 55](#_Toc41510295)

[Parallel Versus Distributed Technology 55](#_Toc41510296)

[Additional Functions of Distributed Databases 57](#_Toc41510297)

[Data Fragmentation 58](#_Toc41510298)

[Data Replication and Allocation 60](#_Toc41510299)

[Types of Distributed Database Systems 61](#_Toc41510300)

[Federated Database Management Systems Issues 61](#_Toc41510301)

[Semantic Heterogeneity 62](#_Toc41510302)

[Query Processing in Distributed Databases 63](#_Toc41510303)

[Data Transfer Costs of Distributed Query Processing 63](#_Toc41510304)

[An Overview of Client-Server Architecture and Its Relationship to Distributed Databases 64](#_Toc41510305)

[CHAPTER EIGHT 66](#_Toc41510306)

[WEB- BASED DATABASES 66](#_Toc41510307)

[Databases on the World Wide Web 66](#_Toc41510308)

[Providing Access to Databases on the World Wide Web 66](#_Toc41510309)

[The Web Integration Option of INFORMIX 67](#_Toc41510310)

[The ORACLE Web Server 68](#_Toc41510311)

[Open Problems with Web Databases 69](#_Toc41510312)

[CHAPTER NINE 71](#_Toc41510313)

[DATA WAREHOUSING 71](#_Toc41510314)

[What is Data Warehouse? 71](#_Toc41510315)

[Understanding a Data Warehouse 72](#_Toc41510316)

[Why a Data Warehouse is separated from Operational Databases? 72](#_Toc41510317)

[Data Warehouse Applications 72](#_Toc41510318)

[Types of Data Warehouse 72](#_Toc41510319)

[OLTP vs. OLAP 73](#_Toc41510320)

[Data Warehouse Architecture 74](#_Toc41510321)

[What is an Aggregation? 75](#_Toc41510322)

[What is Data Mining? 75](#_Toc41510323)

# CHAPTER ONE

# QUERY PROCESSING AND OPTIMIZATION

## Relational Algebra

* The basic set of operations for the ***relational model*** is known as the ***relational algebra***. These operations enable a user ***to specify basic retrieval requests***.
* The result of retrieval is ***a new relation***, which may have been formed from one or more relations.
* The ***algebra operations*** thus ***produce new relations***, which can be further manipulated using operations of the same algebra.
* A ***sequence of relational algebra operations*** forms a ***relational algebra expression***, whose result will also be a relation that represents the result of a **database query** (or ***retrieval request****).*

## TRANSLATING SQL QUERIES INTO RELATIONAL ALGEBRA

**Query block**

* The basic unit that can be translated into the algebraic ***operators*** and ***optimized***.
* A query block contains a single ***SELECT-FROM-WHERE***expression, as well as ***GROUP BY*** and ***HAVING*** clause if these are part of the block.
* ***Nested queries*** within a query are identified as separate **query blocks**.
* ***Aggregate operators*** in SQL must be included in the ***extended algebra***

***SELECT*** LNAME, FNAME ***FROM***EMPLOYEE***WHERE*** SALARY > (***SELECT*** MAX (SALARY) ***FROM*** EMPLOYEE ***WHERE*** DNO = 5);

***SELECT*** MAX (SALARY) ***FROM*** EMPLOYEE ***WHERE*** DNO = 5

***SELECT*** LNAME, FNAME ***FROM*** EMPLOYEE

***WHERE*** SALARY > C

**π** LNAME, FNAME (**σ** SALARY>C(EMPLOYEE))

**ℱ** MAX SALARY (**σ** DNO=5 (EMPLOYEE))

**Relational Algebra consists of several groups of operations**

1. **Unary Relational Operations**

* SELECT (symbol: σ (sigma))
* PROJECT (symbol: π (pi))
* RENAME (symbol: ρ (rho))

1. **Relational Algebra Operations from Set Theory**

* UNION (∪), INTERSECTION (∩), DIFFERENCE (or MINUS, –)
* CARTESIAN PRODUCT (X)

1. **Binary Relational Operations**

* JOIN (several variations of JOIN exist)
* DIVISION

1. **Additional Relational Operations**

* OUTER JOINS, OUTER UNION
* AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

1. **Unary Relational Operations**

***A) SELECT Operation***

* **SELECT** operation is used to select a ***subset of the tuples*** *from a relation* that satisfy a ***selection condition***.
* It is a *filter* that keeps only those tuples that satisfy a qualifying condition those *satisfying the condition are selected while others are discarded.*
* Example: To select the EMPLOYEE tuples whose ***department number*** is ***four*** or those whose ***salary*** is ***greater than $30,000*** the following notation is used:

**Employee** is table name

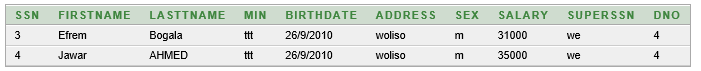
**σ DNO=4(EMPLOYEE)**

**σ SALARY>30,000(EMPLOYEE)**

* Ingeneral, the selection operation is denoted by ***σ<selection condition>(R)*** where the symbol **σ** (Sigma) is used to denote the **select operator** and the selection condition is a **Boolean expression** specified on the attributes of relation(**R**).

**What is the SQL statement of the above unary relational operations?**

**SELECT \* FROM** EMPLOYEE **WHERE** DNO=”4” **AND** SALARY>30000;



**SELECT Operation Properties**

The **SELECT** operations ***σ<selection condition>(R)*** Produces a relation **S** that has the same schema as **R.**

* The **SELECT** operations **σ** is ***commutative*** i.e. ***σ<condition1> (σ<condition2>(R)) = σ<condition2> (σ<condition1>(R)).***
* A **cascade** SELECT operation may be applied in any order i.e. ***σ<condition1> (σ<condition2> (σ <condition3>(R)) = σ<condition2> (σ<condition3> (σ <condition1>(R)))***
* A cascade **SELECT** operation may be replaced by a single selection with conjunctions of all the conditions. i.e. **σ<condition1> (σ<condition2> (σ <condition3>(R)) = σ<condition1>** AND **σ<condition2>** AND **σ <condition3>(R)))**

What are the results of the following select and projection operation? And what is its SQL Statement?

* Results of **SELECT** and **PROJECT** operations

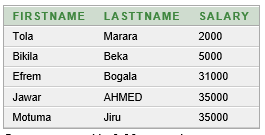
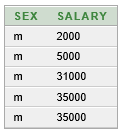
1. **σ (DNO=4 AND SALARY>25000) OR (DNO=5 AND SALARY>30000) (EMPLOYEE)**
2. **LNAME, FNAME, SALARY(EMPLOYEE) SEX, SALARY(EMPLOYEE)**

**The SQL statement of the above relational algebra is the following.**

* 1. **SELECT \* FROM** EMPLOYEE **WHERE** DNO='4' **AND** SALARY>2500 **OR** DNO='5' **AND SALARY>30000;**
  2. **SELECT** FIRSTNAME, LASTNAME, SALARY **FROM** EMPLOYEE;
  3. **SELECT** SEX, SALARY **FROM** EMPLOYEE;

The result of the above SQL statement is:



**B) PROJECT Operation**

* This operation ***selects certain columns from the table and discards the other columns***.
* The **PROJECT** creates a ***vertical partitioning*** one with the needed columns (attributes containing results of the operation and other containing the discarded Columns.
* Example: To list each employee’s first and last name and salary, the following is used:

**LAME, FNAME, SALARY (EMPLOYEE)**

* The general form of the project operation is**: π<attribute list>(R**)
* **π** (pi) is the symbol used to represent the project operation**<attribute list>** is the desired list of attributes from relation R.
* The project operation ***removes any duplicate tuples***. This is because the result of the project operation must be a set of tuples. Mathematical sets do not allow **duplicate elements.**

**PROJECT Operation Properties**

* The number of tuples in the results of projection ***π<attribute list>(R)*** is always less or equal to the number of tuples in R. If the lists of attributes include a key of R, then the number of tuples is equal to the number of tuples in R. ***π<list> (π<list2>(R)) = π<list>(R)*** as long as ***<list2>*** contains the attributes in ***<list2>.***

1. **Relational Algebra Operations from Set Theory**
2. **UNION OPERATION**

* The result of this operation, denoted by ***RUS*** is a relation that includes all tuples that are either in R or in S or in both R and S. ***Duplicate tuples is eliminated***.
* Example: To retrieve the social security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5, we can use the union operation as follows:

**DEP5\_EMPS σ** DNO=5**(EMPLOYEE)**

**RESULT1 π** SSN **(DEP5\_EMPS)**

**RESULT2 (SSN) π SUPERSSN (DEP5\_EMPS)**

**RESULT RESULT1 U RESULT2**

* The union operation produces the tuples that are in either **RESULT1** or **RESULT2** or **both**.
* The two operands must be “***type compatible***”.

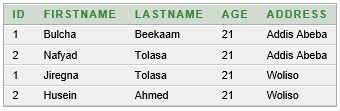
**TYPE COMPATABILITY**

* Type Compatibility of operands is required for the binary set operation ***UNION ∪, (also for INTERSECTION ∩, and SET DIFFERENCE*** -
* ***R1(A1, A2, ..., An)*** and ***R2(B1, B2, ..., Bn)*** are type compatible if:
* They have the same number of attributes, and
* The domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., n).
* The resulting relation for R1∪R2 (also for R1∩R2, or R1–R2) has the same attribute names as the first operand relation R1 (by convention)

**Instructor Student**

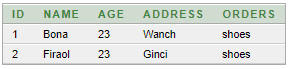
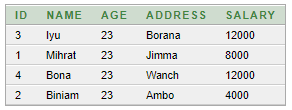


***Instructor U Student***



***SELECT \* FROM student UNION ALL SELECT \* FROM instructor;***

1. **INTERSECTION OPERATION**

* The result of this operation denoted by **R n S** is a relation that includes all tuples that are in both R and S.
* The two operands must be "**type compatible** “. Example: The result of the intersection operation (figure below) includes only those who are both students and instructors. 

  Customers  Customer

1. **Set Difference (or MINUS) Operation**

* The result of this operation, denoted by ***R - S***, is a ***relation that includes all tuples that are in R but not in S***. The two operands must be *"****type compatible****”.*
* Note: both union and intersection are commutative operations; i.e. ***RUS=SUR and R n S = S n R***
* Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are associative operations; that is ***RU(SUT)=(RUS)UT and (R n S) n =R n (S n T)***
* The minus operation is **not commutative**, that is in general.  ***R—S? S--R***

1. **CARTESIAN (or cross product) Operation**

* This operation is used to combine tuples from two relations in a combinational fashion.
* In general, the result of **R(A1, A2,….An) x S(B1, B2,….,Bm**) is a relation **Q** with degree **n+m** attributes **Q(A1, A2,….An, B1, B2,….,Bm)**, in that order.
* The resulting relation **Q** has one tuple for each combination of tuples—one from R and one from S. Hence, if R has nR tuples (denoted as |R|= nR), and S has nS tuples, then |RXS| will have nR\*nS tuples.
* The two operands do not have to be “***type compatible***”.

Example **FEMALE\_EMPS** **σ** SEX = ‘F’(EMPLOYEE)

**EMPNAMES**  **π FNAME, LNAME, SSN(FEMALE\_EMPS)**

**EMP\_DEPENDENTS EMPNAMES X DEPENDENT**

1. **Binary Relational Operations**
2. **JOIN Operation**

* The sequence of Cartesian product followed by select is used quite commonly to identify and select related tuples from two relations, a special operation, called ***JOIN***.
* It is denoted by a ****
* This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.
* The general form of a join operation on two relations ***R (A1, A2, …, An)*** and ***S (B1, B2,… Bn)*** is

**R<join condition> S. Where R and S** can be any relations that result from general relational algebra expressions.

* **EXAMPLE: -** Suppose that we want to retrieve the name of the manager of each Department. To get the managers name we need to combine each ***DEPARTMENT*** tuple with the EMPLOYEE tuple whose ***SSN*** value matches the MGRSSN value in the department tuple. We do this by using the join ****operation. **DEPT\_MGR DEPARTMENT  EMPLOYEE MGRSSN=SSN**

1. **EQUIJOIN Operation**

* The most common use of join involves join conditions with ***equality comparisons only***.
* Such a join, where the only comparison operator used is =, is called an EQUIJOIN.
* In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.
* The JOIN seen in the previous example was EQUIJOIN.

1. **NATURAL JOIN Operation**

* Because one of each pair of attributes with identical values is superfluous, a new operation called natural join—denoted by \*—was created to get rid of the second (superfluous) attribute in an **EQUIJOIN** condition.
* The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations.
* If this is not the case, a renaming operation is applied first.

EXAMPLE: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT\_LOCATIONS, it is sufficient to write: **DEPT\_LOCS DEPARTMENT \* DEPT\_LOCATIONS**

* The set of operations including **select, project, union(u), set difference (-), and Cartesian product(x)** is called a complete set because any other relational algebra expression can be expressed by a combination of these five operations. Example ***R n S =(RUS)-((R-S) U(S-R))***

1. **Additional Relational Operations**
2. **Aggregate Functions and Grouping**

* A type of request that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collections of values from the database.
* Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
* These functions are used in simple statistical queries that summarize information from the database tuples.
* Common functions applied to collections of numeric values include **SUM, AVERAGE, MAXIMUM, and MINIMUM**.
* The COUNT function is used for counting tuples or values.

**Use of the Functional operator F**

* **F MAX Salary(Employee)** retrieves the maximum salary value from the Employee relation
* **F MIN Salary(Employee)** retrieves the minimum Salary value from the Employee relation
* **F SUM Salary(Employee)** retrieves the sum of the Salary from the Employee relation
* **DNO F COUNT SSN, AVERAGE Salary(Employee)** groups employees by DNO (department number) and computes the count of employees and average salary per department.[ Note: count just counts the number of rows, without removing duplicates]

**What is Query Processing?**

* The Steps required transforming **high level SQL query** into a correct and **“efficient**” strategy for execution and retrieval. The activities involved in **parsing, validating, optimizing, and executing a query**.

**What is the aim of query processing?**

* 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 the relational algebra),** and to execute the strategy to retrieve the required data.

**What is Query Optimization?**

* The activity of choosing a single ***“efficient” execution strategy*** *(****from hundreds****)* as determined by database catalog statistics for processing a query. An important aspect of query processing is ***query optimization.***

**What is the aim of 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 resource usage***.
* Generally, we try to reduce the *total execution time of the query*, which is the sum of the execution times of all individual operations that make up the query.

**Examples for query Optimization:** Identify all managers who work in a London branch

SQL: - SELECT \* FROM Staff s, Branch b WHERE s.branchNo = b.branchNo AND s.position = ‘Manager’ AND b.city = ‘london’;

**Results in these equivalent relational algebra statements**

**(1) s(position=‘Manager’)^(city=‘London’)^(Staff.branchNo=Branch.branchNo) (Staff X Branch)**

**(2) s(position=‘Manager’)^(city=‘London’) (Staff Staff.branchNo = Branch.branchNo Branch)**

**(3) [s(position=‘Manager’) (Staff)] Staff.branchNo = Branch.branchNo [s(city=‘London’) (Branch)]**

**Assume:**

* + 1000 tuples in *Staff*.
  + 50 Managers
  + 50 tuples in *Branch*.
  + 5 London branches
  + No indexes or sort keys
  + All temporary results are written back to disk (memory is small)
  + Tuples are accessed one at a time (not in blocks)

**Query 1 (Bad)**



* ***Requires (1000+50) disk accesses to read from Staff and Branch relations***
* ***Creates temporary relation of Cartesian Product (1000\*50) tuples***
* ***Requires (1000\*50) disk access to read in temporary relation and test predicate***

***Total Work = (1000+50) + 2\*(1000\*50) = 101,050 I/O operations***

**Query 2 (Better)**



* + ***Again requires (1000+50) disk accesses to read from Staff and Branch***
  + ***Joins Staff and Branch on branchNo with 1000 tuples (1 employee : 1 branch )***
  + ***Requires (1000) disk access to read in joined relation and check predicate***

**Total Work = (1000+50) + 2\*(1000) = 3050 I/O operations**

**3300% Improvement over Query 1**

**Query 3 (Best)**



* + ***Read Staff relation to determine ‘Managers’ (1000 reads)***
    - ***Create 50 tuple relation(50 writes)***
  + ***Read Branch relation to determine ‘London’ branches (50 reads)***
    - ***Create 5 tuple relation (5 writes)***
  + ***Join reduced relations and check predicate (50 + 5 reads)***

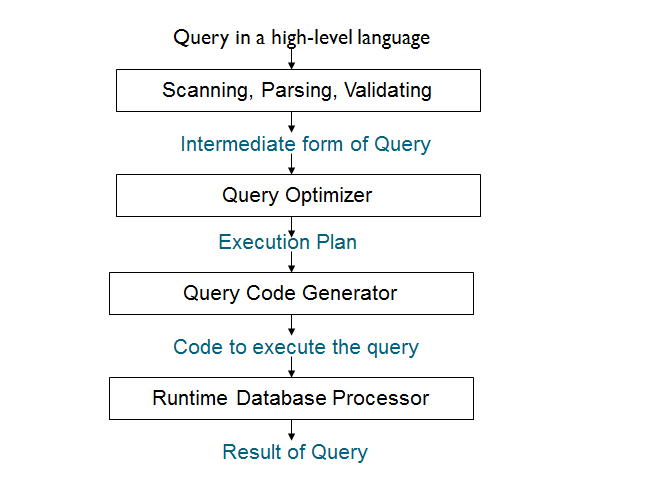
**Total Work = 1000 + 2\*(50) + 5 + (50 + 5) = 1160 I/O operations**

**8700% Improvement over Query 1**

**Two main Techniques for Query Optimization**

1. **Heuristic Rules: -*Rules for ordering the operations in query optimization*.**
2. **Systematical estimation: -**It estimates cost of different execution strategies and chooses the execution plan with ***lowest execution cost***.

**Steps in Processing High-Level Query**

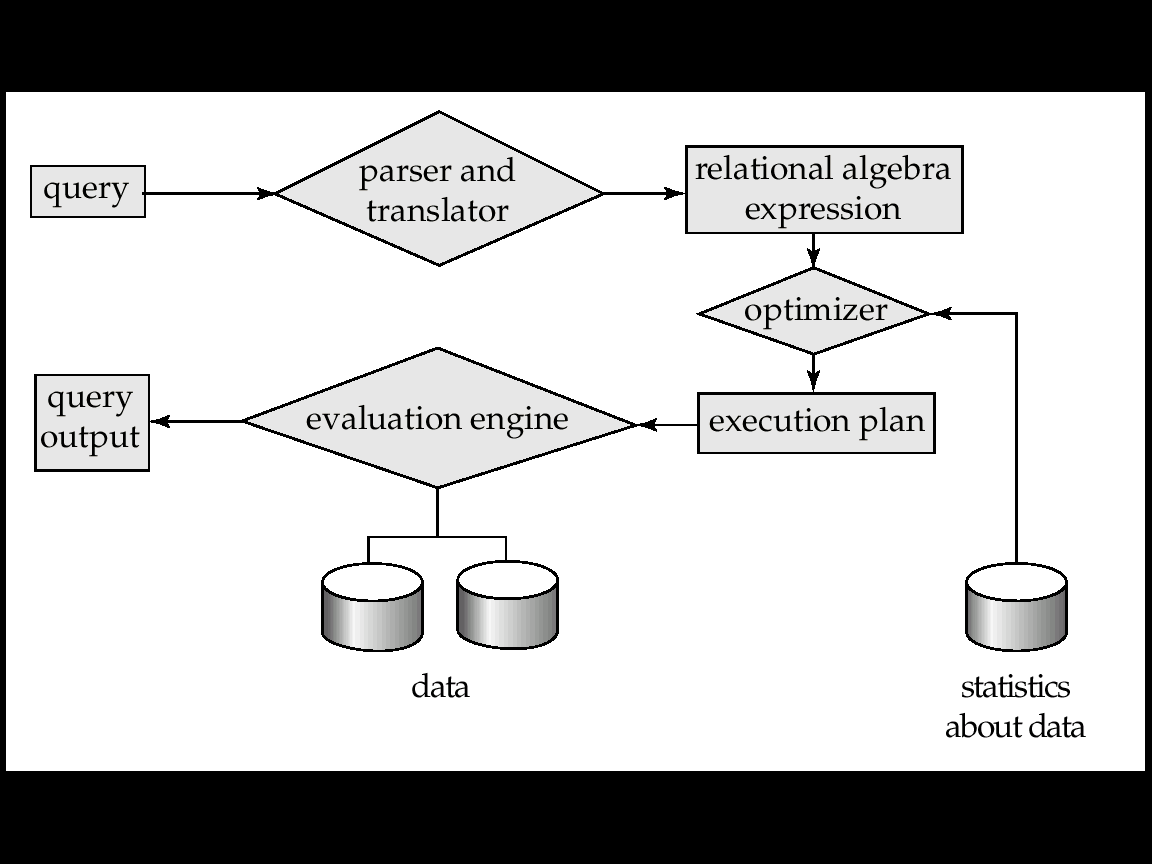


**Scanning, Parsing, Validating**

* ***Scanner:*** The scanner identifies the language tokens such as ***SQL Keywords****,* ***attribute names*,** and ***relation names*** in the text of the query.
* ***Parser:*** The parser checks the ***query syntax*** to determine whether it is formulated according to the ***syntax rules*** of the query language.
* ***Validation:*** The query must be validated by checking that all attributes and relation names are valid and semantically meaningful names in the schema of the particular database being queried.

## QUERY PROCESSING

* ***Query Optimization:*** The process of choosing a suitable execution strategy for processing a query. This module has the task of ***producing an execution plan.***
* ***Query Code Generator:*** It **generates** the **code to execute the plan.**
* ***Runtime Database Processor****:* It has the task of running the query code whether in compiled or interpreted mode. If a runtime error results an error message is generated by the **runtime database processor**.

**Query Processing Steps**

Processing can be divided into ***Decomposition, Optimization, Execution, and Code generation*** main categories.

**1. Query Decomposition**

* It is the process of transforming a high-level query into a relational algebra query, and to check that the query is syntactically and semantically correct.
* It Consists of parsing and validation

Typical stages in query decomposition are:

1. **Analysis:**

* Lexical and syntactical analysis of the query (correctness) based on attributes, data type...
* Query tree will be built for the query containing
  + - **leaf node** for base relations,
    - one or many **non-leaf nodes** for relations produced by relational algebra operations and
    - **Root node** for the result of the query.
* Sequence of operation is from the leaves to the root. (**SELECT \* FROM Catalog c, Author a Where a.authorid = c.authorid AND c.price>200 AND a.country= ‘ USA’ )**

**2. Normalization:**

* Convert the query into a normalized form.
* The predicate WHERE will be converted to Conjunctive (∨) or Disjunctive (∧) Normal form.

**3. Semantic Analysis**

* To reject normalized queries that is not correctly formulated or contradictory.
* Incorrect if components do not contribute to generate result.
* Contradictory if the predicate cannot be satisfied by any tuple.
* Say for example, (Catalog = “BS” Λ Catalog= “CS”) since a given book can only be classified in either of the category at a time

4. **Simplification**

* to detect redundant qualifications,
  + - eliminate common sub-expressions, and
    - Transform the query to a semantically equivalent but more easily and effectively computed form.
* For example, if a user doesn’t have the necessary access to all of the objects of the query, it should be rejected.

2. **Query Optimization**

* Everyone wants the performance of their database to be optimal.
* In particular, there is often a requirement for a specific query or object that is query based, to run faster.
* Problem of query optimization is to find the sequence of steps that produces the answer to user request in the most efficient manner, given the database structure.
* The performance of a query is affected by
  + - * the tables or queries that underlies the query and
      * The complexity of the query.
* Given a request for data manipulation or retrieval, an optimizer will choose an optimal plan for evaluating the request from among the manifold alternative strategies. i.e. there are many ways (access paths) for accessing desired file/record. Hence, ***DBMS is responsible to pick the best execution strategy based on various considerations (Least amount of I/O and CPU resources.***

**Transformational rules for relational Algebra**

1. Cascade of SELECTON: conjunctive SELCTION operations can cascade into individual selection operations and vice versa.



1. Commutativity of SELECTION operations



1. Cascade of **PROJECTION**: in the sequence of PROJECTION Operations, only last in the sequence is required.



1. Commutativity of SELECTION with PROJECTION and vice versa
2. If the predicate C1 involves only the attributes in the projection list(L1), then the selection and projection operations commute



1. Commutativity of THETA JOIN/ Cartesian product RXS is equivalent to SXR. Also holds for Equi-join and Natural join



1. Commutativity of SELECTION with THETA JOIN
2. If the predicate c1 involves only attributes of one of the relations (R) being joined, then the selection and join operations commute. 
3. If the predicate is in the forms of c1^ c2 and c1 involves only attributes of R and c2 involves only attributes of S, then the selection and theta join operations commute.



1. Commutativity of PROJECTION and THETA JOIN

If the projection list of the form L1 L2, where L1 involves only attributes of R and L2 involves only attributes of S being joined, and the predicate c involve only attributes in the projection list, then the SELECTION and JOIN operations commute.



1. Commutativity of the set operations: UNION and INTERSECTION but not STE DIFFERENCE



1. Associativity of THETA JOIN, CARTESIAN PRODUCT, UNION and INTERSECTION



1. Commuting SELECTION with SET OPERATIONS



1. Commuting SELECTION with UNION



Heuristic approach will be implemented by using the above transformation rules in the following sequence or steps. Sequence for applying Transformation Rules

1. Use Rule 1—Cascade Selection
2. Use

Rule 2. Commutatively of SELECTION

Rule 4. Commuting SELECTION with PROJECTION

Rule 6 Commuting SELECTION with JOIN and CARTESIAN

Rule 10 Commuting SELECTION with SET OPERATIONS

1. Use: Rule 9 Associativity of binary operations (JOIN, CARTESIAN, UNION and INTERSECTION). Rearrange nodes by making the most restrictive operations to be performed first (moving it as far down the tree as possible)
2. Perform Cartesian Operations with the subsequent selection operation
3. Use Rule 3 Cascade of PROJECTION

Rule 4 Commuting PROJECTION with SELECTION

Rule 7 Commuting PROJECTION with JOIN and CARTESIAN

Rule 11 Commuting PROJECTION with UNION

## Heuristic Query Tree Optimization

* It has some rules which utilize equivalence expressions to transform the initial tree into final, optimized query tree.
* **Process for heuristics optimization**
  1. The parser of a high-level query generates an initial internal representation;
  2. Apply heuristics rules to optimize the internal representation.
  3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.

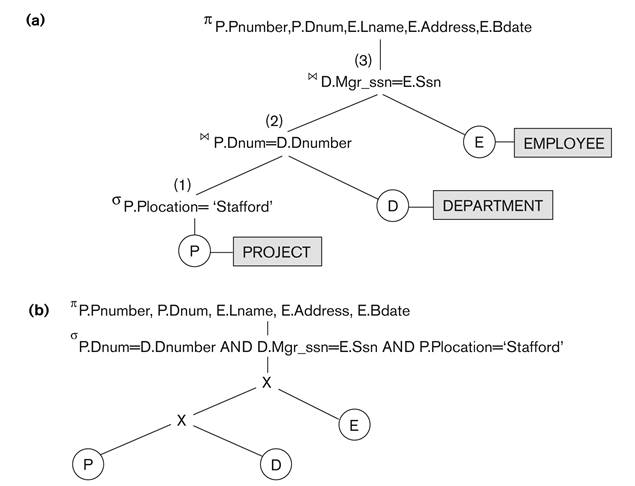
1. **The main heuristic is to apply first the operations that reduce the size of intermediate results.**

E.g. Apply ***SELECT*** and ***PROJECT*** operations before applying the JOIN or other binary operations.

* The main idea behind is to reduce intermediate results. This includes performing
  + ***SELECT operation to reduce the number of tuples &***
  + ***PROJECT operation to reduce number of attributes.***

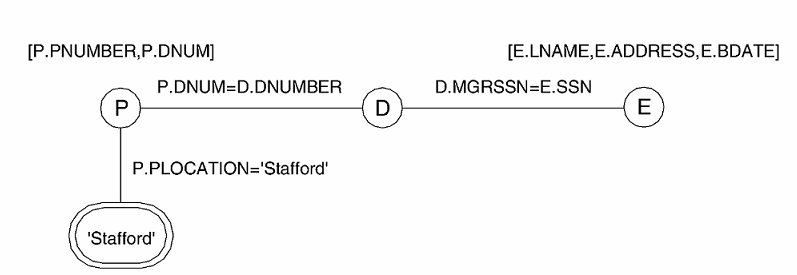
**Query tree**:

* A tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as **leaf nodes** of the **tree**, and represents the relational algebra operations as internal nodes.
* **Example:** For every project located in ‘Stafford’, retrieve the project number, the controlling department number and the department manager’s last name, address and birthdate.
* ***SQL query: Q2: SELECT P. NUMBER, P.DNUM,E.LNAME,E.ADDRESS, E.BDATE FROM PROJECT AS P,DEPARTMENT AS D, EMPLOYEE AS E WHERE P.DNUM=D.DNUMBER AND D.MGRSSN=E.SSN AND P.PLOCATION=‘STAFFORD’;***
* ***Relation algebra****:* ***P PNUMBER, DNUM, LNAME, ADDRESS, BDATE (((s PLOCATION=‘STAFFORD’ (PROJECT)) DNUM=DNUMBER (DEPARTMENT)) MGRSSN=SSN (EMPLOYEE))***
* The same query could correspond too many different relational algebra expressions and hence many different query trees.



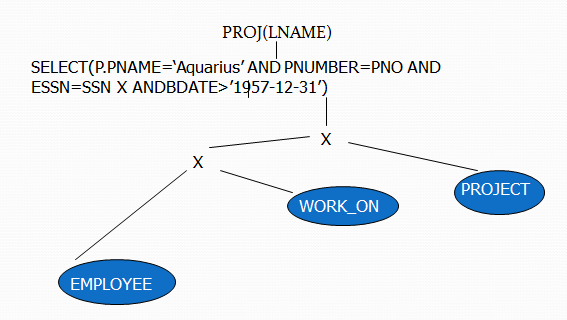
**Query graph**:

* A graph data structure that corresponds to a relational calculus expression. It does *not* indicate an order on which operations to perform first. Nodes represent Relations. Ovals represent constant nodes. Edges represent Join & Selection conditions. Attributes to be retrieved from relations represented in square brackets.
* ***Drawback: - Does not indicate an order on which operations are performed first.***
* There is only a **single** graph corresponding to each query.

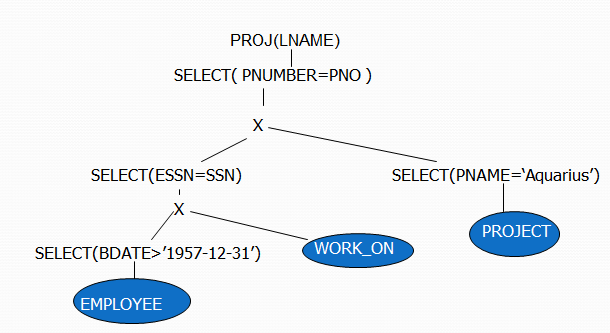


**Example 2 of Heuristic Optimization of Query Trees:**

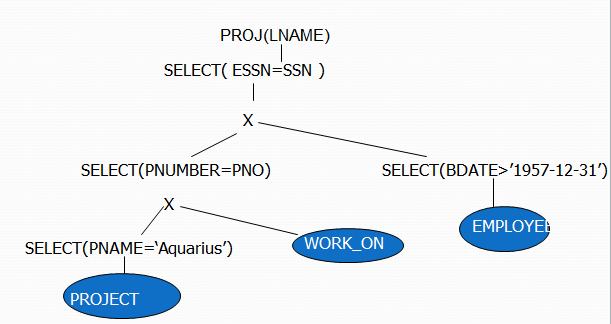
* The same query could correspond too many different relational algebra expressions and hence many different query trees.
* The task of heuristic optimization of query trees is to find a **final query tree** that is efficient to execute.
* **For Example :** Q:-***SELECT LNAME FROM EMPLOYEE, WORKS\_ON, PROJECT WHERE PNAME= ‘AQUARIUS’ AND PNUMBER=PNO AND ESSN=SSN AND BDATE > ‘1957-12-31’***



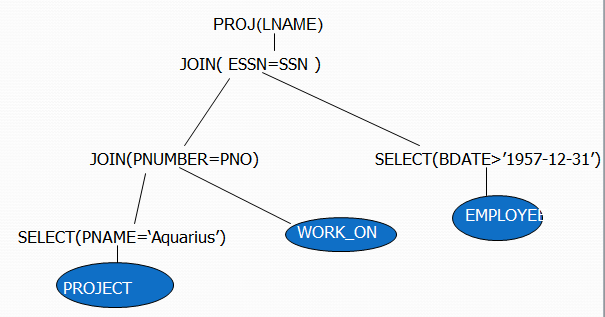
**Fig 1: Initial (canonical) query tree for SQL query Q**



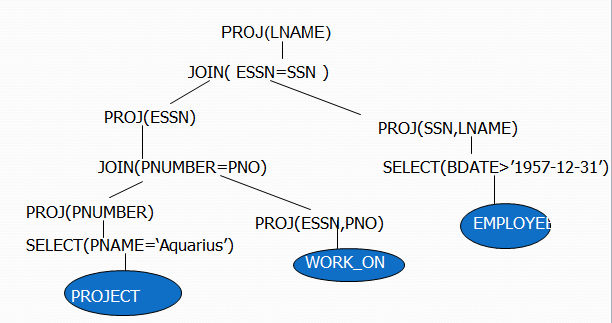
**Fig2: move select down the tree using cascade & commutatively rule of select operation**



**Fig3: rearrange of leaf nodes, using commutatively & associativity of binary operations.**



**Fig 4: converting select & Cartesian product into join**



**Fig 5: break-move of project using cascade & commuting rules of project operations.**

**Query Execution Plans**

* An execution plan for a relational algebra query consists of a combination of the relational algebra query tree and information about the access methods to be used for each relation as well as the methods to be used in computing the relational operators stored in the tree.
* **Materialized evaluation:** the result of an operation is stored as a temporary relation.
* **Pipelined evaluation:** as the result of an operator is produced, it is forwarded to the next operator in sequence.

## Summary of Heuristics for Algebraic Optimization:

1. The main heuristic is to apply first the operations that reduce the size of intermediate results.
2. Perform select operations as early as possible to reduce the number of tuples and perform project operations as early as possible to reduce the number of attributes. (This is done by moving select and project operations as far down the tree as possible.)
3. The select and join operations that are most restrictive should be executed before other similar operations. (This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.)

## Cost-based query optimization:

* + The optimizer examines alternative access paths and operator algorithms and chooses the execution plan with lowest estimate cost.
  + The query cost is calculated based on the estimated usage of resources such as **I/O, CPU and memory needed**.
  + Application developers could specify hints to the ORACLE query optimizer.
  + The idea is that an application developer might know more information about the data.
  + Issues of cost-based query optimization are.
    - Cost function
    - Number of execution strategies to be considered
  + Cost Components for Query Execution
    - Access cost to secondary storage
    - Storage cost
    - Computation cost
    - Memory usage cost
    - Communication cost
  + Note: Different database systems may focus on different cost components.
  + **Catalog Information Used in Cost Functions**
* Information about the size of a file
* number of records (tuples) (r),
* record size (R),
* number of blocks (b)
* blocking factor (bfr)
* Information about indexes and indexing attributes of a file
  + - Number of levels (x) of each multilevel index
    - Number of first-level index blocks (bI1)
    - Number of distinct values (d) of an attribute
    - Selectivity (sl) of an attribute
    - Selection cardinality (s) of an attribute. (s = sl \* r)

*1.* ***Access Cost of Secondary Storage***

* Data is going to be accessed from secondary storage, as a query will need some part of the data stored in the database. The disk access cost can again be analyzed in terms of:
  + - * Searching
      * Reading, and
      * Writing, data blocks used to store some portion of a relation.
* **Remark:** The disk access cost will vary depending on
* The file organization used and the access method implemented for the file organization.
* Whether the data is stored contiguously or in scattered manner, will affect the disk access cost.

***2. Storage Cost***

* While processing a query, as any query would be composed of many database operations, there could be one or more intermediate results before reaching the final output.
* These intermediate results should be stored in primary memory for further processing.
* The bigger the intermediate relation, the larger the memory requirement, which will have impact on the limited available space.

*3.* ***Computation Cost***

* Query is composed of many operations. The operations could be database operations like reading and writing to a disk, or mathematical and other operations like:
  + Searching
  + Sorting
  + Merging
  + Computation on field values

***4. Communication Cost***

* In most database systems the database resides in one station and various queries originate from different terminals.
* This will have impact on the performance of the system adding cost for query processing.
* Thus, the cost of transporting data between the database site and the terminal from where the query originate should be analyzed.

3. **Query Execution Plans**

* An execution plan for a relational algebra query consists of a combination of the relational algebra query tree and information about the access methods to be used for each relation as well as the methods to be used in computing the relational operators stored in the tree.

## Semantic Query Optimization:

* Uses constraints specified on the database schema in order to modify one query into another query that is more efficient to execute. Consider the following SQL query, ***SELECT E. LNAME, M. LNAME FROM EMPLOYEE E M WHERE E. SUPERSSN=M.SSN AND E. SALARY>M.SALARY***

Explanation:

* Suppose that we had a constraint on the database schema that stated that no employee can earn more than his or her direct supervisor.
* If the semantic query optimizer checks for the existence of this constraint, it need not execute the query at all because it knows that the result of the query will be empty. Techniques known as **theorem** proving can be used for this purpose.

**Exercise**

1. **Select the alternative answer from the given letter option** 
   1. Among the list below one is not cost based optimization
2. Cost function
3. Number of execution strategies to be considered
4. Computational cost
5. None of above
   1. The basic set of operations for the relational model is known as the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. Relational algebra.
7. Relational models
8. Relational query
9. Relational theory
   1. Nested queries within a query are identified as separate \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
10. query blocks
11. query summary
12. query invention
13. query control
    1. **\_\_\_\_\_\_\_\_\_\_\_\_\_** operation is used to select a subset of the tuplesfrom a relation that satisfy a selection condition.
       1. Delete
       2. Select
       3. Update
       4. Project
    2. Which one is Binary Relational Operations?
       1. Selection
       2. Deletion
       3. Join
       4. Projection
    3. Which one is Relational Algebra Operations from Set Theory?
       1. Select
       2. Project
       3. Rename
       4. Cartesian product
    4. Which one is unary relational algebra?
       1. Cartesian product
       2. Rename
       3. Division
       4. Projection
    5. Which one is not catalog information used in cost function used to exhibit relational algebra?
       1. Record size
       2. Number of blocks
       3. Cost function
       4. Blocking factor
    6. Which one is not part of information about indexes and indexing attributes of a file?
       1. Number of levels
       2. Selectivity
       3. Selection cardinality
       4. Number of blocks
    7. The disk access cost can again be analyzed in terms of:
       1. Searching
       2. Reading
       3. Writing
       4. All can be answer
14. **Write short answer for the following question** 
    1. To increase your understanding try to generalize relational algebra in terms of database!
    2. Illustrate query processing and explain each step undoubtable?
    3. How you define heuristic query optimizations?

# CHAPTER TWO

# DATABASE SECURITY AND AUTHORIZATION

## Introduction to Database Security Issues

* In today's society, some information is extremely important that needs to be protected.
  + For example, ***disclosure or modification*** of **military information** could cause danger to national security.
* A ***good database security management system*** has to ***handle the possible database threats***.
* ***Threat*** may be ***any situation or event,*** whether ***intentional (planned) or accidental***, that may adversely affect a system and consequently the organization.
* **Threats to databases**: It may result in degradation of some/all security goals like;
  + ***Loss of Integrity***
* *Only authorized users should be allowed to modify data.*
* *For example, students may be allowed to see their grades, but not allowed to modify them.*
  + ***Loss of Availability***-if DB is not available for those users/ to which they have a legal right to uses the data.
    - Authorized users should not be denied access.
    - For example, an instructor who wishes to change a grade should be allowed to do so.
  + ***Loss of Confidentiality***
* *Information should not be disclosed to unauthorized users.*
* *For example, a student should not be allowed to examine other students' grades.*

## Authentication

* ***All users*** of the database will have ***different access levels and permission for different data*** objects.
* **Authentication** is the process of checking whether the user is the one with the privilege for the access level. Thus, the system will check whether the user with a specific **username and password** is trying to use the resource.

## Authorization/Privilege

* ***Authorization*** refers to the process that determines the mode in which a particular (previously authenticated) client is allowed to access a specific resource controlled by a server.
* Any database access request will have the following three major components.

1. ***Requested Operation****: what kind of operation is requested by a specific query?*
2. ***Requested Object:*** *on which resource or data of the database is the operation sought to be applied?*
3. ***Requesting User****: who is the user requesting the operation on the specified object?*

**Forms of user authorization**

There are different forms of user authorization on the resource of the database. These include:

1. ***Read Authorization:*** the user with this privilege is allowed only to read the content of the data object.
2. ***Insert Authorization:*** the user with this privilege is allowed only to insert new records or items to the data object.
3. ***Update Authorization:*** users with this privilege are allowed to modify content of attributes but are not authorized to delete the records.
4. ***Delete Authorization:*** users with this privilege are only allowed to delete a record and not anything else.

*Note: Different users, depending on the power of the user, can have one or the combination of the above forms of authorization on different data objects.*

## Database Security and the DBA

* The ***database administrator (DBA)*** is the ***central authority for managing a database system***.
* The ***DBA’s*** responsibilities include
  + - *Account creation*
    - *granting privileges to users who need to use the system*
    - *Privilege revocation*
    - *classifying users and data in accordance with the policy of the organization*

**Access Protection, User Accounts, and Databases Audits**

* Whenever a person or group of persons need to access a database system, the individual or group must first apply for a user account.
* The ***DBA*** will then create a new ***account id*** and ***password*** for the user if he/she believes there is a legitimate need to access the database.
* The user must log in to the ***DBMS*** by ***entering account id and password*** whenever database access is needed.
* The database system must also keep ***track of all operations*** on the database that are applied by a certain user throughout ***each login session***.
* If any tampering with the database is assumed, a ***database audit*** is performed
  + A ***database audit*** consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period.
* A ***database log*** that is used mainly for security purposes is sometimes called an ***audit trail.***
* To protect databases against the possible threats two kinds of countermeasures can be implemented:

1. ***Access control, and***
2. ***Encryption***

**Access Control (AC)**

**1. Discretionary Access Control (DAC)**

* The typical method of ***enforcing discretionary access control*** in a database system is ***based on the granting and revoking privileges.***
* The ***granting and revoking of privileges*** for discretionary privileges ***known as the access matrix model*** where
  + *The rows of a matrix M represents subjects (users, accounts, programs)*
  + *The columns represent objects (relations, records, columns, views, operations).*
  + *Each position* ***M(i,j)*** *in the matrix represents the types of privileges (read, write, update) that subject* ***i*** *holds on object* ***j****.*
* To control the ***granting and revoking of relation privileges***, each relation R in a database is assigned an ***owner account***, which is typically the account that was used when the relation was created in the first place.
  + The ***owner of a relation*** is given all privileges on that relation.
  + The ***owner account holder*** can pass privileges on any of the owned relation to other users by granting privileges to their accounts.

**Privileges Using Views**

* The mechanism of ***views*** is an important ***discretionary authorization mechanism*** in its own right.
  + *For example, If the* ***owner A*** *of a* ***relation R*** *wants another* ***account B*** *to be able to retrieve only* ***some fields of R****, then A* ***can create a view V*** *of R that includes only those attributes and then grant* ***SELECT on V*** *to B.*

**Revoking Privileges**

* In some cases, it is desirable to grant a privilege to a ***user temporarily***.
* For example, the ***owner of a relation*** may want to grant the ***SELECT*** privilege to a user for a specific task and then revoke that privilege once the task is completed. Hence, ***a mechanism for revoking privileges is needed.***
* In SQL, a **REVOKE** command is included for the purpose of ***canceling privileges***.

**Propagation of Privileges using the GRANT** OPTION

* Whenever the ***owner A of a relation R grants*** a privilege on R to ***another account B***, privilege can be given to B with or without the ***GRANT OPTION.***
* If the ***GRANT OPTION*** is given, this means that ***B can also grant that privilege on R to other accounts.*** 
  + Suppose that ***B*** is given the ***GRANT OPTION*** by A and that B then grants the privilege on R to a third account ***C***, also with ***GRANT OPTION***.
  + In this way, privileges on R can ***propagate*** to other accounts ***without the knowledge of the owner of R.***
  + If the owner account ***A now revokes*** the privilege ***granted to B***, ***all the privileges that B propagated based*** on that privilege should automatically ***be revoked by the system.***

**Example 1**

* Suppose that the ***DBA creates four accounts***: **A1, A2, A3, A4** and wants only A1 to be able to create relations. Then the DBA must issue the following ***GRANT*** command in SQL: -  **GRANT CREATE TABLE TO A1;**

**Example 2**

* Suppose that ***A1 creates*** the two base relations ***EMPLOYEE*** and ***DEPARTMENT.*** A1 is then ***owner of these two relations*** and hence A1 has ***all the relation privileges*** on each of them.
* Suppose that ***A1 wants to grant A2*** the privilege to ***insert and delete rows*** in both of these relations, but A1 does not want A2 to be able to propagate these privileges to additional accounts:
  + - ***GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;***

**Example 3**

* Suppose that ***A1 wants to allow A3*** to ***retrieve information*** from either of the table ***(Department or Employee***) and also to be able to propagate the ***SELECT privilege*** to other accounts. A1 can issue the command:
  + - ***GRANT SELECT ON*** *EMPLOYEE, DEPARTMENT* ***TO*** *A3* ***WITH GRANT OPTION;***
* **A3** can grant the **SELECT** privilege on the **EMPLOYEE** relation to A4 by issuing:
  + - ***GRANT SELECT******ON*** *EMPLOYEE* ***TO*** *A4;*
* Notice that A4 can’t propagate the SELECT privilege because GRANT OPTION was not given to A4

**Example 4**

* Suppose that A1 decides to ***revoke the SELECT privilege*** on the EMPLOYEE relation from ***A3; A1*** can issue:
  + - **REVOKE SELECT ON** EMPLOYEE **FROM** A3**;**
* The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from A4, too, because A3 granted that privilege to A4 and A3 does not have the privilege any more.

**Example 5**

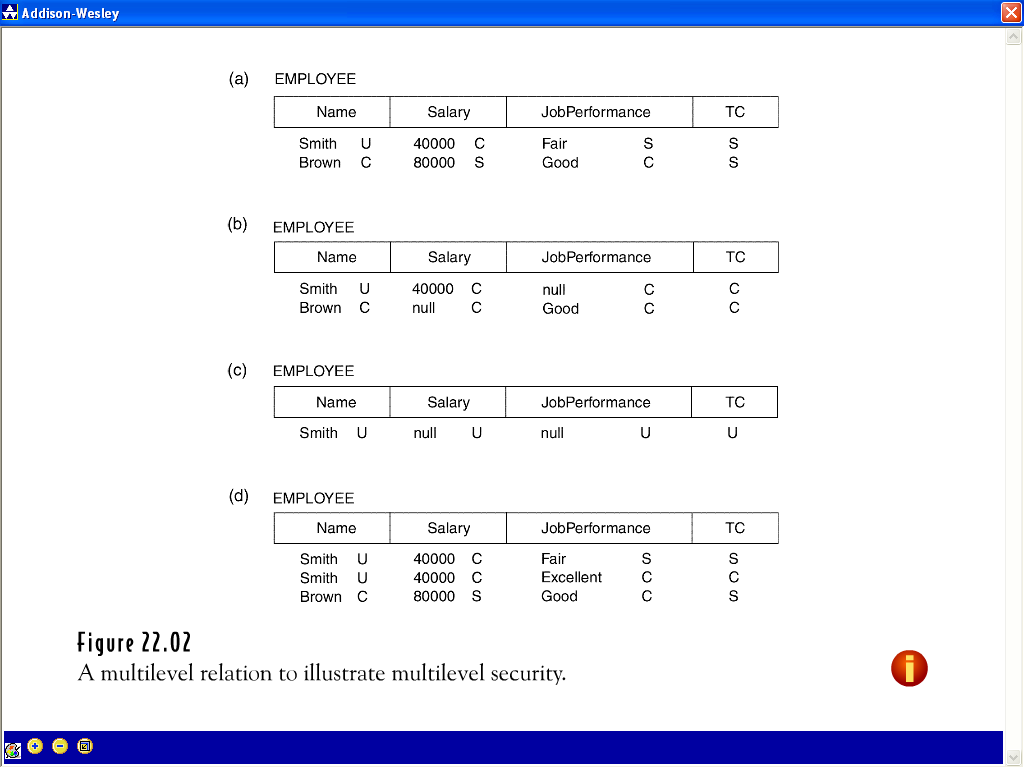
* Suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege. The limitation is to retrieve only the ***NAME, BDATE, and ADDRESS*** attributes and only for the tuples with DNO=5.
* A1 then create the view:
  + - ***CREATE VIEW A3 EMPLOYEE AS SELECT NAME, BDATE, ADDRESS FROM EMPLOYEE WHERE DNO = 5;***
* After the view is created, A1 can grant SELECT on the view ***A3 EMPLOYEE to A3*** as follows:
  + - ***GRANT SELECT ON A3 EMPLOYEE TO A3 WITH GRANT OPTION;***

**Example 6**

* + - Finally, suppose that A1 wants to allow A4 to update only the SALARY attribute of EMPLOYEE; A1 can issue:
    - ***GRANT UPDATE ON EMPLOYEE (SALARY) TO A4;***

**2 Mandatory Access Control (MAC)**

* ***DAC*** *techniques* are an ***all-or-nothing method***: A user ***either has or does not have*** a certain privilege.
* In many applications, ***additional security policy*** is needed that classifies data and users based on security classes.
* Typical ***security classes*** are ***top secret (TS), secret (S), confidential (C), and unclassified (U),*** where TS is the highest level and U the lowest: ***TS ≥ S ≥ C ≥ U***
* The commonly used model for multilevel security, known as the ***Bell-LaPadula model***, classifies each ***subject (user, account, program) and object (relation, tuple, column, view, operation)*** into one of the ***security classifications, TS, S, C, or U***:
  + - **Clearance** (classification) of a subject S as **class(S**) and to the **classification** of an object O as **class (O)**.
* Two restrictions are enforced on data access based on the subject/object classifications:
  + *A subject S is not allowed read access to an object O unless* ***class(S) ≥ class (O).***
  + *A subject S is not allowed to write an object O unless* ***class(S) ≤ class (O).***
* To ***incorporate multilevel security notions*** into the ***relational database model***, it is common to consider attribute values and rows as data objects. Hence, each attribute A is associated with a ***classification attribute C*** in the schema.
* In addition, in some models, a **tuple classification** attribute ***TC*** is added to the relation attributes to provide a classification for each tuple as a whole.
* Hence, a ***multilevel relation*** schema R with n attributes would be represented as
  + ***R(A1,C1,A2,C2, …, An,Cn,TC)*** *where each Ci represents the classification attribute associated with attribute Ai.*
* The value of the **TC** attribute in each tuple t – which is the highest of all attribute classification values within t – provides a general classification for the tuple itself.
* Whereas, each ***Ci*** provides a finer security classification for each attribute value within the tuple.
* A ***multilevel relation*** will appear to contain different data to subjects (users) with different clearance levels.
  + In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as ***filtering***.
  + In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the ***apparent key.***
* This leads to the concept of **poly instantiation** where several tuples can have the same apparent key value but have different attribute values for users at different classification levels.
* ***Example***. Consider query ***SELECT \* FROM employee***



* 1. The original employee table,
  2. After filtering employee table for classification C users,
  3. After filtering employee table for classification U users
  4. Poly instantiation of the smith row for C users who want to modify some value
* A user with a security clearance S would see the same relation shown above (a) since all row classification are less than or equal to S as shown in (a).
* However, a user with security clearance C would not allow to see values for salary of Brown and job performance of Smith, since they have higher classification as shown in (b)
* For a user with security clearance U, filtering introduces null values for attributes values whose security classification is higher than the user’s security clearance as shown in (c)
* A user with security clearance C may request for update on the values of job performance of smith to ‘Excellent’ and the view will allow him to do so. However, the user shouldn't be allowed to overwrite the existing value at the higher classification level.
  + - Solution: to create Ploy Station for smith row at the lower classification level C as shown in (d)

## Comparing DAC and MAC

* ***DAC policies*** are ***characterized by a high degree of flexibility,*** which makes them suitable for a large variety of application domains.
  + The main ***drawback of DAC models*** is their ***weakness to malicious attacks****,* such as Trojan horses embedded in application programs.
* By contrast, ***mandatory policies*** ensure a high degree of protection in a way; they prevent any illegal flow of information.
  + Mandatory policies have the drawback of being ***too rigid and they are only applicable in limited environments.***
* In many practical situations, ***discretionary policies are preferred*** because they offer a better trade-off between security and applicability.

**3. Role-Based Access Control**

* Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles.
* Roles can be created using the **CREATE ROLE** and **DESTROY ROLE** commands.
  + - The **GRANT** and **REVOKE** commands discussed under DAC can then be used to assign and revoke privileges from roles.
* **RBAC** appears to be a feasible alternative to discretionary and mandatory access controls;
* It ensures that only authorized users are given access to certain data or resources.
* Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.
* Role hierarchy in **RBAC** is a natural way of organizing roles to reflect the organization’s lines of authority and responsibility[**:\My DB File\Role.ppt**](file:///\\Role.ppt)

## Introduction to Statistical Database Security

* ***Statistical databases*** are used mainly to produce statistics on various populations.
* The database may contain ***confidential data*** on individuals, which should be protected from user access.
* Users are permitted to retrieve ***statistical information*** on the populations, such as ***averages, sums, counts, maximums, minimums,*** *and* ***standard deviations****.*
* A **population** is a ***set of rows of a relation (table) that satisfy some selection condition***.
* Statistical queries involve *applying* ***statistical functions*** to a ***population*** *of rows.*
* For example, we may want to retrieve the *number* of individuals in a ***population*** or the *average income* in the population.
  + - However, statistical users are not allowed to retrieve individual data, such as the income of a specific person.
* ***Statistical database security techniques*** must ***disallow the retrieval of individual data***.

This can be achieved by elimination of queries that retrieve attribute values and by allowing only queries that involve statistical aggregate functions such as, ***SUM, MIN, MAX,*** and Such queries are sometimes called ***statistical queries.***

* It is ***DBMS’s responsibility to ensure confidentiality of information about individuals***, while still providing useful statistical summaries of data about those individuals to users.
* Provision of **privacy protection** of users in a statistical database is paramount.
* In some cases, it is possible to **infer** the values of individual rows from sequence statistical queries.
  + This is particularly true when the conditions result in a population consisting of a small number of rows.

**Encryption**

* ***Authorization*** may **not be sufficient to protect data in database systems**, especially when there is a situation where data should be ***moved from one location to the other using network facilities.***
* ***Encryption*** is used to ***protect information stored at a particular site or transmitted between sites from being accessed by unauthorized users.***
* ***Encryption*** is the ***encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key.***
* It is not possible for encrypted data to be read unless the reader knows how to ***decipher/decrypt*** the encrypted data.
* If a ***database system holds particularly sensitive data,*** it may be believed necessary to encode it as an insurance against possible ***external threats or attempts to access it.***
* The ***DBMS can access data after decoding it***, although there is ***degradation in performance because of the time taken to decode it***.
* ***Encryption*** also protects data ***transmitted over communication lines***.
* To transmit data securely over insecure networks requires the use of a ***Cryptosystem,*** which includes:

1. An *encryption key to encrypt the data* ***(plaintext)***
2. An *encryption algorithm that, with the encryption key,* transforms the plaintext into cipher text
3. A *decryption key to decrypt the cipher text*
4. A *decryption algorithm that, with the decryption key,* transforms the cipher text back into plaintext

* ***Data encryption standard*** is an approach which does both a substitution of characters and a rearrangement of their order based on an encryption key.

## Types of Cryptosystems

* Cryptosystems can be categorized into two:

1. ***Symmetric encryption – uses the same key for both encryption and decryption*** and relies on safe communication lines for exchanging the key.
2. ***Asymmetric encryption – uses different keys for encryption and decryption.***

* Generally,
  + ***Symmetric algorithms*** are much faster to execute on a computer than those that are ***asymmetric.***
  + ***Asymmetric algorithms*** are ***more secure than symmetric algorithms.***

**Public Key Encryption** algorithm: ***Asymmetric encryption***

* This algorithm operates with ***modular arithmetic*** – ***mod n,*** where ***n*** is the ***product of two large prime numbers.***
* Two keys, ***d and e***, are used for ***decryption and encryption.***
  + **n** is chosen as a large integer that is a product of **two large distinct prime numbers**, **p** and **q**.
  + The encryption key **e** is a randomly chosen number between 1 and **n** that is **relatively prime** to ***(p-1) x (q-1).***
* The **plaintext** **m** is encrypted as ***C****=* ***me mod n****.*
* However, the decryption key **d** is carefully chosen so that ***C d mod n = m****.*
* The decryption key **d** can be computed from the condition that **d x e -1** is divisible by **(p-1)x(q-1).**
* Thus, the legitimate receiver who knows **d** simply computes **Cd mod n = m** and recovers **m.**

**Simple Example: *Asymmetric encryption***

1. Select primes p=11, q=3.
2. n = pq = 11\*3 = 33
3. Find phi winch Is Gien by, phi = (p-1)(q-1) = 10\*2 = 20
4. Choose e=3 ( 1<e<phi)
5. Check for gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
6. Compute d (1<d<phi) such that d \*e -1 is divisible by phi   
   Simple testing (d = 2, 3 ...) gives d = 7
7. Check: ed-1 = 3\*7 - 1 = 20, which is divisible by phi (20).

Given

Public key = (n, e) = (33, 3)

Private key = (n, d) = (33, 7)

* Now say we want to encrypt the message m = 7
  + c = me mod n = 73 mod 33 = 343 mod 33 = 13
  + Hence the ciphertext c = 13
* To check decryption, we compute
  + m = cd mod n = 137 mod 33 =62,748,517 mod 33 = 7

Exercise

Select the best answer from the given option

1.

# CHAPTER THREE

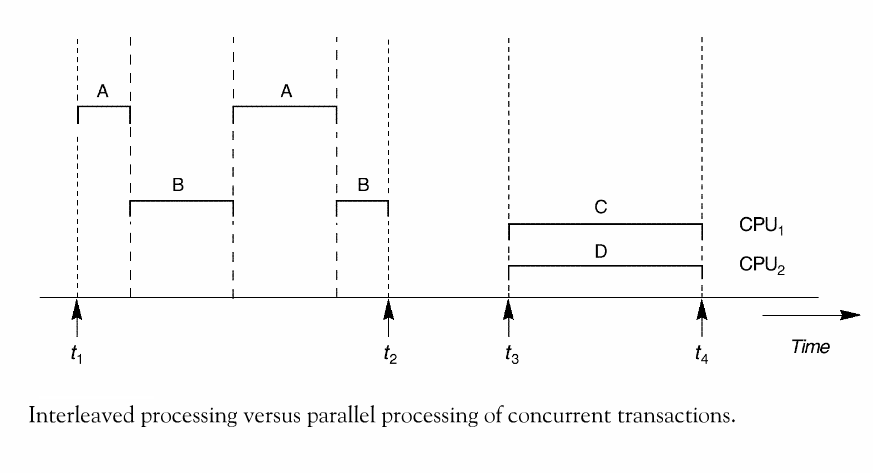
# TRANSACTION PROCESSING CONCEPTS

## Introduction to Transaction Processing

* **Single-User System:**
* At most one user at a time can use the database management system. Eg. ***Personal computer system***.
* **Multi-user System:**
* Many users can access the DBMS concurrently. Eg. ***Airline reservation, Bank*** and the like system are operated by many users who submit transaction concurrently to the system.
* This is achieved by ***multi programming***, which allows the computer to ***execute multiple programs /processes at the same time***.

**Concurrency**

* ***Interleaved processing:***
* Concurrent execution of processes is interleaved in a single CPU using for example, round robin algorithm
* **Advantages:**
* keeps the CPU busy when the process requires I/O by switching to execute another process rather than ***remaining idle*** during I/O time and hence this will increase system throughput (average no. of transactions completed within a given time)
* Prevents long process from delaying other processes (minimize unpredictable delay in the response time).
* ***Parallel processing:***
* If Processes are concurrently executed in multiple CPUs.



* **A Transaction**
  + Logical unit of database processing that includes one or more access operations (***read -retrieval, write - insert or update, delete***). Examples include ***ATM transactions, credit card approvals, flight reservations, hotel check-in, phone calls, supermarket scanning, academic registration and billing***.
  + Collections of operations that form a single logical unit of work are called ***transactions***.
  + A **transaction** is a unit of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high-level data-manipulation language (typically SQL), or programming language (for example, C++, or Java), with embedded database accesses in JDBC or ODBC.
* **Transaction boundaries**
  + Any single transaction in an application program is bounded with ***Begin* and *End statements***. An **application program** may contain several transactions separated by the ***Begin* and *End transaction*** boundaries.
  + This collection of steps must appear to the user as a ***single, indivisible unit***. Since a transaction is ***indivisible***, it either executes in its entirety or not at all. Thus, if a transaction begins to execute but fails for whatever reason, any changes to the database that the transaction may have made must be ***undone***. This requirement holds regardless of whether the transaction itself failed. For example,
    - * *if it divided by zero,*
      * *the operating system crashed, or*
      * *The computer itself stopped operating.*

## ACID properties of the transactions

1. ***Atomicity.*** Either all operations of the transaction are reﬂected properly in the database, or none are. This “***all-or-none***” property is referred to as ***atomicity***.
2. ***Consistency.*** Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

A transaction must preserve ***database consistency***—if a transaction is run atomically in isolation starting from a ***consistent database***, the database must again be consistent at the end of the transaction.

1. ***Isolation.*** Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions ***Ti and Tj***, it appears to Ti that either Tj ﬁnished execution before Ti started or Tj started execution after Ti ﬁnished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

The database system must take special actions to ensure that transactions ***operate properly without interference from concurrently executing database statements***. This property is referred to as ***isolation.***

1. ***Durability.*** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

## SIMPLE MODEL OF A DATABASE

* **A database** is a collection of named data items.
* Because SQL is a powerful and complex language, we focus on when data are moved from disk to main memory and from main memory to disk.
* **Granularity** of data a field, a record, or a whole disk block that measure the size of the data item
* Basic operations that a transaction can perform are **read** and **write. Transactions access data using two operations.**
  + ***read item(X):*** Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
  + ***write item(X):*** Writes the value of program variable X into the database item named X.
* Basic unit of data transfer from the disk to the computer main memory is ***one block***.
  + **read item(X**) command includes the following steps:
    - Find the address of the disk block that contains item X.
    - Copy that ***disk block*** into a ***buffer in main memory*** (if that disk block is not already in some main memory buffer).
    - Copy ***item X*** from the buffer to the program variable named X.
  + **write item(X**) command includes the following steps:
    - Find the address of the disk block that contains item X.
    - Copy that ***disk block*** into a ***buffer in main memory*** (if that disk block is not already in some main memory buffer).
    - Copy item X from the program variable named X into its correct location in the buffer.
    - Store the updated block from the buffer back to disk (either immediately or later).
* The DBMS maintains a number of ***buffers in the main memory*** that holds database disk blocks which contains the database items being processed.
  + *When this buffer is occupied and*
  + *if there is a need for additional database block to be copied to the main memory.*
* Some buffer management policy is used to choose for replacement but if the chosen buffer has been modified, it must be written back to disk before it is used.

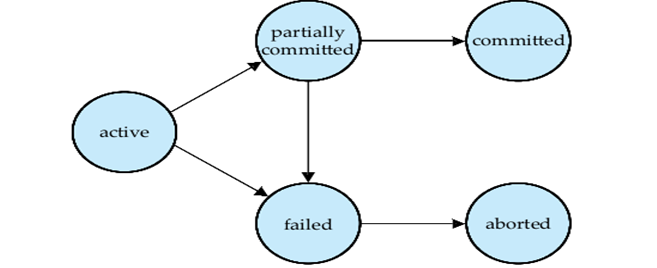
## Transaction Atomicity and Durability

* A transaction may ***not always complete*** its execution successfully. Such a transaction is termed ***aborted.***
* If we are to ensure the atomicity property, an ***aborted transaction*** must have ***no effect*** on the state of the database. Thus, any changes that the ***aborted transaction*** made to the database must be ***undone.***
* Once the changes caused by an aborted transaction have been undone, we say that the transaction has been ***rolled back.*** It is part of the responsibility of the ***recovery scheme to manage transaction aborts.*** This is done typically by maintaining a ***log***.
* Each database modiﬁcation made by a transaction is ﬁrst ***recorded in the log***. We record the identiﬁer of the transaction performing the modiﬁcation, the identiﬁer of the data item being modiﬁed, and both the ***old value (prior to modiﬁcation)*** and the ***new value (after modiﬁcation)*** of the data item. Only then is the database itself modiﬁed.
* *Maintaining a* ***log*** *provides the possibility of redoing a modiﬁcation to ensure atomicity and durability as well as the possibility of undoing a modiﬁcation to ensure atomicity in case of a failure during transaction execution.*
* A transaction that ***completes its execution successfully*** is said to be ***committed***. A committed transaction that has performed updates transforms the database into a ***new consistent state***, which must persist even if there is a system failure.
* Once a transaction has committed, we cannot undo its effects by ***aborting it.***
* *The only way to undo the effects of a committed transaction is to* ***execute a compensating transaction (pay costs).*** *For instance, if a transaction added* ***$20*** *to an account, the compensating transaction would subtract* ***$20*** *from the account.*

**Transaction States**

A transaction must be in one of the following states:

* **Active***, the initial state; the transaction stays in this state while it is executing.*
* **Partially committed***, after the ﬁnal statement has been executed.*
* **Failed,** *after the discovery that normal execution can no longer proceed.*
* **Aborted,** *after the transaction has been* ***rolled back*** *and the database has been* ***restored*** *to its state prior to the start of the transaction.*
* **Committed,** *after successful completion.*
* A transaction has **committed** only if it has entered the **committed state**.
* A transaction has **aborted** only if it has entered the **aborted state**.
* A transaction is said to have **terminated** if it has either **committed or aborted.**



**Figure 1: - State diagram of a transaction**

* *A transaction starts in the* ***active state****. When it ﬁnishes its ﬁnal statement, it enters the* ***partially committed state.*** *At this point, the transaction has completed its execution, but it is still possible that it may have to be* ***aborted****, since the actual output may still be temporarily residing in main memory, and thus a hardware failure may preclude its successful completion.*
* The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be re-created when the system restarts after the failure. When the last of this information is written out, the transaction enters the **committed state**.

A ***transaction*** enters the ***failed state*** after the system determines that the transaction ***can no longer proceed with its normal execution*** (for example, because of ***hardware or logical errors).*** Such a transaction must be ***rolled back.*** Then, it enters the aborted state. At this point, the system has two options:

1. *It can* ***restart the transaction,*** *but only if the transaction was aborted as a result of some hardware or software error that was not created through the internal logic of the transaction. A restarted transaction is considered to be a* ***new transaction.***
2. *It can* ***kill the transaction****. It usually does so because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad, or because the desired data were not found in the database.*

**Transaction Isolation**

* ***Transaction-processing systems*** usually allow **multiple transactions** to ***run concurrently.*** Allowing multiple transactions to update data concurrently causes several complications with ***consistency of the data***. There are two good reasons for allowing concurrency:

1. ***Improved throughput and resource utilization***. To increases the ***throughput of the system***—that is, the number of transactions executed in a given amount of time. Correspondingly, the ***processor (CPU)*** and ***disk utilization*** also increase; in other words, the processor and disk spend less time idle, or not performing any useful work.
2. ***Reduced waiting time***

* If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete, which can ***lead to unpredictable delays in running a transaction***.
* If the transactions are operating on different parts of the database, it is better to let them run concurrently, **sharing the CPU cycles and disk accesses** among them.
* ***Concurrent execution reduces the unpredictable delays in running transactions.*** Moreover, it also ***reduces the average response time***: the average time for a transaction to be completed after it has been submitted.
* *The database system must control the interaction among the concurrent transactions to prevent them from destroying the* ***consistency of the database****. It does so through a variety of mechanisms called* ***concurrency-control schemes.***

Example: - Let **T1 and T2** be two transactions that transfer funds from **one account to another**.

1. *Transaction T1 transfers* ***50 birr*** *from account A to account B. It is deﬁned as:*

***T1:read(A);***

***A:= A − 50;***

***write(A);***

***read(B);***

***B:= B + 50;***

***write(B)***

1. *Transaction T2 transfers* ***10 percent*** *of the balance from account A to account B. It is deﬁned as:*

***T2:read(A);***

***Temp: = A \*0.1;***

***A: = A − temp;***

***write(A);***

***read(B);***

***B:= B + temp;***

***write(B)***

* Suppose the current values of accounts A and B are ***1000 birr and 2000 birr***, respectively. Suppose two transactions are executed ***one at a time in the order T1 followed by T2***.
* This execution sequence appears in **Figure 2**. In the ﬁgure, the sequence of instruction steps is in chronological order from **top to bottom.** The ﬁnal values of accounts A and B, after the execution in **Figure 2** takes place, are ***855 birr*** and ***2145 birr***, respectively. Thus, the total amount of money in accounts A and B—that is, the ***sum A + B***—is preserved after the execution of both transactions.***T1 T2***

8 read(A)

9 temp: = A ∗ 0.1

10 A: = A − temp

11 write(A)

12 read(B)

13 B: = B + temp

14 write(B)

15 commit

1 read(A)

2 A: = A − 50

3 write(A)

4 read(B)

5 B: = B + 50

6 write(B)

7 commit

**Figure 2 Schedule 1a serial schedule in which T1 is followed by T2**.

* Schedule 1—a serial schedule in which T1 is followed by T2. Similarly, if the transactions are executed one at a time in the order T2 followed by T1, then the corresponding execution sequence is that of **Figure 3.** Again, as expected, the sum A + B is preserved, and the ﬁnal values of accounts A and B are ***850 birr*** and ***2150 birr***, respectively.

1 read(A)

2 temp: = A ∗ 0.1

3 A: = A − temp

4 write(A)

5 read(B)

6 B: = B + temp

7 write(B)

8 commit

9 read(A)

10 A: = A − 50

11 write(A)

12 read(B)

13 B: = B + 50

14 write(B)

15 commit

***T1 T2***

***read(A)***

***A:= A − 50***

***write(A)***

***read(B)***

***B:= B + 50***

***write(B)***

**Figure 3 Schedule 2 a serial schedule in which T2 is followed by T1.**

* The execution sequences just described are called ***schedules***. They represent the chronological order in which instructions are executed in the system. These schedules are ***serial***.
* Each ***serial schedule*** consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule.
* When the ***database system*** executes several transactions concurrently, the corresponding schedule no longer needs to be ***serial***. If two transactions are running concurrently, the operating system may execute one transaction for a little while, then perform a ***context switch,*** execute the second transaction for some time, and then switch back to the ﬁrst transaction for some time, and so on. With multiple transactions, the CPU time is shared among all the transactions.
* *Not all concurrent executions result in a* ***correct state****. Consider the schedule of Figure 5. After the execution of this schedule, we arrive at a state where the ﬁnal values of accounts A and B are* ***950 birr and 2100 birr****, respectively. This ﬁnal state is an* ***inconsistent state****, since we have gained* ***50 birr*** *in the process of the concurrent execution. Indeed, the sum A + B is not preserved by the execution of the two transactions.*
* *It is the job of the database system to ensure that any schedule that is executed will leave the database in a* ***consistent state****. The concurrency-control component of the database system carries out this task.*

**4 read(A)**

**5 temp: = A ∗ 0.1**

**6 A: = A − temp**

**7 write(A)**

12 read(B)

13 B: = B + temp

14 write(B)

15 commit

**1 read(A)**

**2 A: = A −50**

**3 Write(A)**

8 read(B)

9 B: = B + 50

10 write(B)

11 commit

**T1 T2**

**Figure 4 Schedule 3 a concurrent schedule equivalent to schedule 1**.

* *We can ensure consistency of the database under concurrent execution by making sure that any schedule that is executed has the same effect as a schedule that could have occurred without any concurrent execution. That is, the schedule should, in some sense, be equivalent to a serial schedule. Such schedules are called* ***serializable schedules.***

***3 read(A)***

***4 temp: = A ∗ 0.1***

***5 A: = A − temp***

***6 write(A)***

***7 read(B***)

13 B: = B + temp

14 write(B)

15 commit

**T1 T2**

***1 read(A)***

***2 A: = A −50***

**8** Write(A)

9 read(B)

10 B: = B + 50

11 write(B)

12 commit

**Figure 5 Schedule 4—a concurrent schedule resulting in an inconsistent state.**

## Serializability

* Serial schedules are ***serializable*,** but if steps of multiple transactions are interleaved, it is harder to determine whether a ***schedule is serializable***. Since ***transactions are programs***, it is difﬁcult to determine exactly
* *What operations a transaction performs and*
* *How operations of various transactions interact.*
* For this reason, we shall consider only on two types of operations that a transaction can perform on a data item: ***read and write*.**
* We assume that, between a ***read (Q) instruction*** *and* ***a write (Q) instruction*** on a data item **Q**, a transaction may perform an arbitrary sequence of operations on the copy of **Q** that is residing in the local buffer of the transaction.
* Let us consider a ***schedule S*** in which there are two ***consecutive instructions***, ***I and J, of transactions*** **Ti** and **Tj**, respectively **(i = j).**
* If ***I and J*** refer to ***different data items***, then we can **swap (exchange) I and J** without affecting the results of any instruction in the schedule.
* If ***I and J*** refer to the ***same data item Q***, then the order of the two steps may matter.

1. ***I = read (Q), J = read (Q).*** The order of I and J does not matter.
2. ***I = read (Q), J = write (Q).*** If ***I come before J***, then **Ti** does not read the value of **Q** that is written by **Tj in instruction J**. If ***J comes before I***, then Ti reads the value of Q that is written by **Tj**. Thus, the order of **I and J matters.**

**1 read(A)**

**2 write(A)**

**5 read(B)**

**6 write(B)**

**3 read(A)**

**4 write(A)**

**7 read(B)**

**8 write(B)**

**T1 T2**

**Figure 6 Schedule 3 showing only the read and write instructions.**

**T1 T2**

**3 read(A)**

**5 write(A)**

**7 read(B)**

**8 write(B)**

**1 read(A)**

**2 write(A))**

**4 read(B)**

**6 write(B)**

**Figure 7 Schedule 5 schedule 3 after swapping of a pair of instructions.**

1. ***I = write (Q), J = read (Q).*** The order of I and J matters for reasons similar to those of the previous case.
2. ***I = write (Q), J = write (Q).*** Since both instructions are write operations, the order of these instructions does not affect either ***Ti or Tj.***

However, the value obtained by the next ***read (Q)*** instruction of ***S*** is affected, since the result of only the latter of the two write instructions is preserved in the database.

If there is no other ***write (Q)*** instruction after ***I and J in S,*** then the order of ***I and J*** directly affects the ﬁnal value of Q in the database state that results from schedule ***S.***

We say that ***I and J conﬂict*** if they are operations by different transactions on the same data item, and at ***least one of these instructions is a write operation.***

To illustrate the concept of conﬂicting instructions, we consider schedule 3. The ***write (A) instruction*** of ***T1 conﬂicts*** with the ***read (A) instruction*** of T2. However, the ***write (A) instruction*** of T2 does not conﬂict with the **read (B) instruction** of T1, because the two instructions access different data items.

**5 read(A)**

**6 write(A)**

**7 read(B)**

**8 write(B)**

**1 read(A)**

**2 write(A)**

**3 read(B)**

**4 write(B)**

**T1 T2**

**Figure 8 Schedule 6 a serial schedule that is equivalent to schedule 3.**

**1 read(Q)**

3 write(Q)

**T3 T4**

**2 write(Q)**

**Figure 9 Schedules 7.**

* Let ***I and J*** be consecutive instructions of a ***schedule S***. If ***I and J*** are instructions of different transactions and ***I and J*** do not conﬂict, then we can swap the order of **I and J** to produce a new ***schedule S.***
* ***S is equivalent to S***, since all instructions appear in the same order in both schedules except for **I and J**, whose order does not matter.
* Since the **write (A)** **instruction** of T2 in schedule 3 does not conﬂict with the ***read (B) instruction of T1,*** we can swap these instructions to generate an **equivalent schedule**, ***schedule 5,*** ***in Figure 7***. Regardless of the initial system state, ***schedules 3 and 5*** both produce the same ﬁnal system state.
* We continue to swap non conﬂicting instructions:
* *Swap the* ***read (B) instruction of T1*** *with the* ***read (A) instruction of T2****.*
* *Swap the* ***write (B) instruction of T1*** *with the* ***write (A) instruction of T2****.*
* *Swap the* ***write (B) instruction of T1*** *with the* ***read (A) instruction of T2.***
* The ﬁnal result of these swaps, schedule 6 of Figure 8, is a ***serial schedule.***

**Transaction Isolation and Atomicity**

* Effect of transaction failures during concurrent execution.

**3 read(A)**

**4 commit**

**T1 T5**

**1 read(A)**

2 write(A)

5 read(B)

**Example Schedule 9, a non-recoverable schedule.**

* If a transaction ***Ti fails,*** for whatever reason, we need to ***undo the effect of this transaction to ensure the atomicity property of the transaction***.
  + In a system that allows ***concurrent execution***, the atomicity property requires that any ***transaction Tj*** that is ***dependent on Ti*** (that is, Tj has read data written by Ti) is also ***aborted.***
  + To achieve this, we need to place restrictions on the type of schedules permitted in the system. In the following two subsections, we address the issue of what schedules are acceptable from the viewpoint of recovery from ***transaction failure.***

**Recoverable Schedules**

* A ***recoverable schedule*** is one where, for each pair of transactions ***Ti*** *and* ***Tj*** such that ***Tj*** reads a data item previously written by ***Ti***, the commit operation of ***Ti appears before the commit operation of Tj.***

**Cascade less Schedules**

* Even if a schedule is ***recoverable***, to recover correctly from the failure of a transaction ***Ti***, we may have to ***roll back several transactions***. Such situations occur if transactions have read data written by ***Ti.***
* Example ***transaction T8*** writes a value of A that is ***read by transaction T9***. Transaction ***T9*** ***writes a value of A*** that is read by ***transaction T10***.
  + Suppose that, at this point, ***T8 fails.*** T8 must be ***rolled back***. Since ***T9*** is dependent on T8, ***T9*** must be ***rolled back***. Since **T10** is dependent on T9, **T10** must be ***rolled back***.
* This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called ***cascading rollback.* T9 T10**

**6 read(A)**

**1 read(A)**

2 read(B)

3 write(A)

7 abort

**4 read(A)**

**5 write(A)**

**T8 T10**

***Example Schedules 10.***

* ***Cascading rollback*** is undesirable, since it leads to the undoing of a signiﬁcant amount of work.
* It is desirable to restrict the schedules to those where ***cascading rollbacks cannot occur***. Such schedules are called ***cascade less schedules***.
  + Formally, a ***cascade less schedule*** is one where, for each pair of ***transactions Ti and Tj*** such that ***Tj*** reads a data item previously written by ***Ti***, the commit operation of Ti appears before the read operation of Tj. It is easy to verify that every ***cascade less schedule is also recoverable.***

**Transaction Isolation Levels**

* ***Serializability*** is allows programmers to ignore issues related to concurrency when they code transactions.
* The ***SQL standard*** also allows a transaction to specify that it may be executed in such a way that it becomes ***non-serializable*** with respect to other transactions. For instance, a transaction may operate at the isolation level of ***read uncommitted***, which permits the transaction to read a data item even if it was written by a transaction that has ***not been committed.***

The isolation levels speciﬁed by the SQL standard are as follows:

* ***Serializable*** usually ensures ***serializable execution***.
* ***Repeatable read*** allows only committed data to be read and further requires that, between two reads of a data item by a transaction, no other transaction is allowed to update it
* ***Read committed*** allows only committed data to be read, but does not require repeatable reads. For instance, between two reads of a data item by the transaction, another transaction may have updated the data item and committed.
* ***Read uncommitted*** allows uncommitted data to be read. It is the lowest isolation level allowed by SQL.

All the isolation levels above additionally disallow ***dirty writes***, that is, they disallow writes to a data item that has already been written by another transaction that has not yet ***committed or aborted***.

**Locking**

* Instead of ***locking the entire database***, a transaction could, lock only those data items that it accesses. Under such a policy, the transaction must hold ***locks long enough to ensure Serializability****,* but for a period short enough not to harm performance excessively.
* There are two kinds of locks***: shared and exclusive***.
  1. **Shared locks** are used for data that the transaction reads and
  2. ***Exclusive locks*** are used for those it writes.
* Many transactions can hold ***shared locks*** on the same data item at the same time, but a transaction is allowed an ***exclusive lock*** on a data item only if no other transaction holds any lock (regardless of whether shared or exclusive) on the data item. **This use of two modes of locks along with two-phase locking allows concurrent reading of data while still ensuring Serializability.**

## Transactions as SQL Statements

* In SQL, **insert statements** ***create new data*** and ***delete statements*** **delete data**.
* These two statements are, ***write operations***, since they change the database, but their interactions with the actions of other transactions are different.
  + Example, consider the following SQL query on our university database that ﬁnds all instructors who earn more than ***90,000 birrs***.

**Select ID, name from instructor where salary > 90000;**

* + Using our sample instructor relation, we ﬁnd that only **Ein-stein and Brandt** satisfy the condition. Now assume that around the same time we are running our query, another user inserts a new instructor named “**James**” whose salary is **$100,000**.

**Insert into instructor values (’11111’, ’James’, ’Marketing’, 100000);**

* The result of our query will be different depending on whether this ***insert comes before or after our query is run***. In a concurrent execution of these transactions, it is intuitively clear that they conﬂict, but this is a conﬂict not captured by ***our simple model***. This situation is referred to as the ***phantom phenomenon,*** because a conﬂict may exist on “phantom” data.
* But in an SQL statement, the speciﬁc ***data items (tuples)*** referenced may be determined by a where clause predicate. So, the ***same transaction, if run more than once, might reference different data items each time it is run if the values in the database change between runs.***
* One way of dealing with the above problem is to recognize that it is not sufﬁcient for concurrency control to consider only the tuples that are accessed by a transaction; the information used to ﬁnd the tuples that are accessed by the transaction must also be considered for the purpose of concurrency control.
* The information used to ﬁnd tuples could be updated by an **insertion or deletion**, or in the case of an index, even by an update to a search-key attribute.
* For example, if ***locking*** is used for concurrency control, the data structures that track the tuples in a relation, as well as index structures, must be appropriately locked. However, such locking can lead to **poor concurrency** in some situations; index-locking protocols which maximize concurrency, while ensuring Serializability in spite of **inserts, deletes, and predicates in queries.**
  + Let us consider again the query:

***Select ID, name from instructor where salary> 90000;***

* + And the following SQL update:

**Update instructor set salary = salary \*0.9 where name = ’Wu’;**

* If our query ***reads the entire instructor relation***, then it reads the tuple with **Wu’s data** and conﬂicts with the update. However, if an index were available that allowed our query direct access to those tuples with **salary > 90000**, then our query would not have accessed **Wu’s data** at all because **Wu’s salary** is initially **90,000 birr** in our example instructor relation, and reduces to **81,000 birr** after the update.
* However, using the above approach, it would appear that the existence of a conﬂict depends on a **low-level query processing** decision by the system that is unrelated to a user-level view of the meaning of the two SQL statements! An alternative approach to concurrency control treats an **insert, delete or update** as conﬂicting with a predicate on a relation, if it could affect the set of tuples selected by a predicate.
* In our example query above, the predicate is “***salary > 90000***”, and an update of **Wu’s salary** from **90,000 birr** to a value greater than **90,000 birr**, or an update of **Einstein’s salary** from a value greater than **90,000 birr** to a value less than or equal to **90,000 birr**, would conﬂict with this predicate. Locking based on this idea is called **predicate locking**; however predicate locking is expensive, and not used in practice.

## Summary

* A ***transaction*** is a unit of program execution that accesses and updates various data items.
* Transactions are required to have the **ACID properties**: ***atomicity, consistency, isolation***, and **durability.**
* **Atomicity** ensures that either all the effects of a transaction are reﬂected in the database, or none are; a failure cannot leave the database in a state where a transaction is partially executed.
* **Consistency** ensures that, if the database is initially consistent, the execution of the transaction (by itself) leaves the database in a consistent state.
* **Isolation** ensures that concurrently executing transactions are isolated from one another, so that each has the impression that no other transaction is executing concurrently with it.
* **Durability** ensures that, once a transaction has been committed, that transaction’s updates do not get lost, even if there is a system failure.
* Concurrent execution of transactions improves throughput of transactions and system utilization, and also reduces waiting time of transactions.
* The various types of storage in a computer are volatile storage, nonvolatile storage, and stable storage. Data in volatile storage, such as in RAM, are lost when the computer crashes. Data in nonvolatile storage, such as disk, are not lost when the computer crashes, but may occasionally be lost because of failures such as disk crashes. Data in stable storage are never lost.
* Stable storage that must be accessible online is approximated with mirrored disks, or other forms of RAID, which provide redundant data storage. Ofﬂine or archival, stable storage may consist of multiple tape copies of data stored in physically secure locations.
* When several transactions execute concurrently on the database, the consistency of data may no longer be preserved. It is therefore necessary for the system to control the interaction among the concurrent transactions.
* Since a transaction is a unit that preserves consistency, a serial execution of transactions guarantees that consistency is preserved.
* A schedule captures the key actions of transactions that affect concurrent execution, such as read and write operations, while abstracting away internal details of the execution of the transaction.
* We require that any schedule produced by concurrent processing of a set of transactions will have an effect equivalent to a schedule produced when these transactions are run serially in some order.
* A system that guarantees this property is said to ensure serializability.
* There are several different notions of equivalence leading to the concepts of conﬂict serializability and view serializability.
* Serializability of schedules generated by concurrently executing transactions can be ensured through one of a variety of mechanisms called concurrency-control policies.
* We can test a given schedule for conﬂict serializability by constructing a precedence graph for the schedule, and by searching for absence of cycles in the graph. However, there are more efﬁcient concurrency-control policies for ensuring serializability.
* Schedules must be recoverable, to make sure that if transaction a sees the effects of transaction b, and b then aborts, then an also gets aborted.
* Schedules should preferably be cascade less, so that the abort of a transaction does not result in cascading aborts of other transactions. **Cascadelesness** s is ensured by allowing transactions to only read committed data.
* The concurrency-control–management component of the database is responsible for handling the concurrency-control policies.

# CHAPTER FOUR

# CONCURRENCY CONTROL TECHNIQUES

## Introduction to Concurrency control techniques

* When ***several transactions execute concurrently*** in the database, however, the isolation property may ***no longer be preserved***. To ensure that it is, the system must control the interaction among the concurrent transactions; this control is achieved by the mechanisms called ***concurrency-control schemes.***
* There are a variety of **concurrency-control schemes**. The most frequently used schemes are ***two-phase locking*** and ***snapshot isolation***.

**Lock-Based Protocols**

* One way to ensure isolation is ***to require that data items be accessed in a mutually exclusive manner;*** that is, **while one transaction is accessing a data item, no other transaction can modify that data item.**
* The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item.

**Locks**

* There are various modes in which a data item may be locked. Two modes of locks mostly focused are:

1. ***Shared.*** *If a transaction* ***Ti*** *has obtained a* **shared-mode lock** *(denoted by S) on item Q, then Ti can read, but cannot write, Q.*
2. ***Exclusive****. If a transaction* ***Ti*** *has obtained an* **exclusive-mode lock (denoted by X)** *on item Q, then Ti can both read and write Q.*

* Every transaction request a **lock** in an appropriate mode on ***data item Q***, depending on the types of operations that it will perform on Q. The transaction makes the request to the ***concurrency-control manager***.
* The transaction can proceed with the operation only after the ***concurrency-control manager*** ***grants*** the lock to the transaction. The use of these two lock modes ***allows multiple transactions to read a data item but limits write access to just one transaction at a time***.
* ***Examples: -***Let A and B represent arbitrary lock modes. Suppose that a transaction Ti requests a lock of mode A on item Q on which transaction Tj (Ti =Tj) currently holds a lock of mode B. If transaction Ti can be granted a lock on Q immediately, in spite of the presence of the mode B lock, then we say mode A is compatible with mode B. Such a function can be represented conveniently by a matrix.

**Granting of Locks**

* When a transaction requests a lock on a data item in a particular mode, and ***no other transaction*** has a lock on the same data item in a ***conﬂicting mode***, the lock can be granted. However, care must be taken to avoid the following scenario.
* Suppose a ***transaction T2*** has a ***shared-mode lock on a data item***, and ***transaction T1*** requests an ***exclusive-mode lock on the data item***. Clearly, T1 has to wait for T2 to release the ***shared-mode lock.***
* Meanwhile, a ***transaction T3*** may request a ***shared-mode lock on the same data item***. The lock request is compatible with the ***lock granted to T2***, so T3 may be granted the ***shared-mode lock***.
* At this point ***T2*** may release the **lock**, but still **T1** has to ***wait for T3 to ﬁnish***. But again, there may be a ***new transaction T4*** that requests a shared-mode lock on the same data item, and is granted the ***lock before T3 releases*** it.
* In fact, it is possible that there is a sequence of transactions that each requests a ***shared-mode lock on the data item***, and each transaction releases the lock a short while after it is ***granted***, but T1 never gets the ***exclusive-mode lock on the data item.*** The transaction T1 may never make progress, and is said to be ***starved***.
* *We can* ***avoid starvation of transactions*** *by granting locks in the following manner: When a transaction* ***Ti*** *requests a* ***lock on a data item Q*** *in a particular mode M, the* ***concurrency-control manager grants*** *the lock provided that:*

1. *There is no other transaction holding a lock on Q in a mode that conﬂicts with M.*
2. *There is no other transaction that is waiting for a lock on Q and that made its lock request*

*before Ti. Thus, a lock request will never get blocked by a lock request that is made later.*

## The Two-Phase Locking Protocol

* One protocol that ***ensures serializability*** is the ***two-phase locking protocol***. **2-phase locking protocol** is one in which there are 2 phases that a transaction goes through. The first is the **growing phase** in which it is acquiring locks, the second is one in which it is releasing locks. This protocol requires that each transaction issue ***lock*** *and* ***unlock requests in two phases***:

1. ***Growing phase.*** *A transaction may obtain locks, but may not release any lock.*
2. ***Shrinking phase.*** *A transaction may release locks, but may not obtain any new locks.*

* Initially, a transaction is in the ***growing phase***. Once the transaction releases a lock, it enters the ***shrinking phase***, and it can issue no more lock requests.
* For example, ***transactions T3 and T4*** are two phases. On the other hand, ***transactions T1 and T2 are not two phase***. Note that the unlock instructions do not need to appear at the end of the transaction.
* For example, in the case of ***transaction T3***, we could move the ***unlock (B) instruction*** to just after the ***lock-X (A) instruction***, and still retain the two-phase locking property.
* We can show that the two-phase locking protocol ensures conﬂict serializability. Consider any transaction. The point in the schedule where the transaction has obtained its ﬁnal lock (the end of its growing phase) is called the **lock point** of the transaction.
* Now, transactions can be ordered according to their lock points— this ordering is, in fact, a serializability ordering for the transactions.
* **Two-phase locking** does not ensure ***freedom from deadlock***. Observe that transactions T3 and T4 are two phase, but, in schedule 2 they are deadlocked.
* ***Cascading rollback*** may occur under ***two-phase locking***.
* ***Cascading rollbacks*** can be avoided by a ***modiﬁcation of two-phase locking*** called the **strict two-phase locking protocol.**
  + *This protocol requires not only that locking be two phase, but also that all* ***exclusive-mode locks*** *taken by a transaction be held until that transaction commits.*
  + *This requirement ensures that any data written by an* ***uncommitted transaction are locked in exclusive mode until the transaction commits****, preventing any other transaction from reading the data.*
* Another variant of two-phase locking is the **rigorous two-phase locking protocol,** which requires that all locks be held until the transaction commits.
* **Deadlock Handling**
* A system is in a ***deadlock state*** if every transaction in the set is waiting for another transaction in the set.
* There exists a set of waiting transactions ***{T0, T1,..., Tn}*** such that **T0** is waiting for a data item that ***T1 holds***, and **T1** is waiting for a data item that ***T2 holds***, and ... ,and ***Tn−1*** is waiting for a data item that ***Tn holds***, and ***Tn*** is waiting for a data item that ***T0*** holds. None of the transactions can make progress in such a situation.
* To ensure this problem the ***system rolling back*** some of the transactions create such problem.
* There are two principal methods for dealing with the deadlock problem.
  1. *Use a* ***deadlock prevention protocol*** *to ensure that the system will* ***never enter a deadlock state****.*
  2. *Allow the* ***system to enter a deadlock state****, and then try to recover by using a* ***deadlock detection and deadlock recovery scheme****.*
* As we shall see, both methods may result in ***transaction* *rollback***. Prevention is commonly used if the probability that the system would enter a deadlock state is relatively high; otherwise, ***detection and recovery are more efﬁcient.***
* **Deadlock Prevention**
* There are ***two approaches*** to deadlock prevention.

1. *Ensures that* ***no cyclic waits can occur*** *by ordering the requests for locks, or requiring all locks to be acquired together.*
2. ***Performs transaction rollback*** *instead of waiting for a lock, whenever the* ***wait could potentially*** *result in a* ***deadlock****.*

* The ***simplest scheme under the ﬁrst approach*** requires that ***each transaction locks all its data items before it begins execution.*** Moreover, either all are locked in one step or none are locked. There are ***two main disadvantages*** to this protocol:

1. *It is often* ***hard to predict, before the transaction begins****, what data items need to be locked;*
2. *Data-item* ***utilization may be very low****, since many of the data items may be locked but unused for a long time.*

* Another approach for ***preventing deadlocks*** is to ***impose an ordering of all data items***, and to require that a transaction lock data items only in a sequence consistent with the ordering.
* The ***second approach for preventing deadlocks*** is to ***use preemption and transaction rollbacks.***
  + In ***preemption***, when a transaction ***Tj*** requests a lock that transaction ***Ti holds***, the lock granted to ***Ti*** may be ***preempted by rolling back of Ti, and granting of the lock to Tj.***
  + *To* ***control the preemption****, we assign a* ***unique timestamp****, based on a counter or on the system clock, to each transaction when it begins.*
  + *The system* ***uses these timestamps*** *only to decide whether a transaction should* ***wait or roll back.***
  + ***Locking*** *is still used for* ***concurrency control****.*
  + *If a* ***transaction*** *is* ***rolled back****, it retains its* ***old timestamp*** *when restarted.*
* Two different ***deadlock-prevention schemes using timestamps*** have been proposed:

1. The **wait–die** scheme is a ***non-preemptive technique***.
   * *When* ***transaction Ti requests a data item currently held by Tj****, Ti is allowed to wait only if it has a* ***timestamp smaller than that of Tj*** *(that is,* ***Ti is older than Tj****). Otherwise,* ***Ti is rolled back (dies).***
   * *For example, suppose that transactions* ***T14, T15, and T16*** *have t****imestamps 5, 10, and 15****, respectively.* 
     1. ***If T14 requests a data item held by T15, then T14 will wait.***
     2. ***If T24 requests a data item held by T15, then T16 will be rolled back.***
2. The **wound–wait** scheme is a ***preemptive technique***.
   * It is a ***counterpart*** to the **wait–die scheme**.
   * When transaction ***Ti requests a data item currently held by Tj,*** ***Ti*** is allowed to wait only if it has a ***timestamp larger than that of Tj*** (that is, ***Ti is younger than Tj***). Otherwise, **Tj is rolled back** (***Tj is wounded by Ti***).
   * Returning to our example, with transactions ***T14, T15, and T16***,
     1. If ***T14*** requests a data item held by ***T15***, then the data item will be preempted from ***T15, and T15 will be rolled back***.
     2. If ***T16 requests a data item held by T15***, then ***T16 will wait***.

* The major problem with both of these schemes is that **unnecessary rollbacks may occur**.
* Another simple approach to deadlock prevention is based on ***lock timeouts***.
  + In this approach, a transaction that has requested a ***lock waits*** for at most a speciﬁed amount of time.
  + If the lock has not been granted within that time, the transaction is said to ***time out***, and it ***rolls itself back and restarts***.
  + If there was in fact a ***deadlock***, one or more transactions involved in the deadlock will ***time out and roll back, allowing the others to proceed***. *This scheme falls somewhere between* ***deadlock prevention****, where a deadlock will never occur, and* ***deadlock detection and recovery****.*
  + The ***timeout scheme*** is particularly ***easy to implement***, and works well if transactions are ***short*** and if ***long waits are likely to be due to deadlocks***. However, in general it is hard to decide how long a transaction must wait before timing out. Too long a wait results in unnecessary delays once a deadlock has occurred. Too short a wait results in transaction rollback even when there is no deadlock, ***leading to wasted resources.***
  + ***Starvation*** is also a possibility with this scheme. Hence, the timeout-based scheme has limited applicability.

## Deadlock Detection and Recovery

* If a system does not employ some ***protocol that ensures deadlock freedom***, then a detection and recovery scheme must be used.
* An algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred.
* If one has, then the system must attempt ***to recover from the deadlock***. To do so, the system must:
* *Maintain information about the current allocation of data items to transactions, as well as any outstanding data item requests.*
* ***Provide an algorithm*** *that uses this information to determine whether the system has* ***entered a deadlock state.***
* ***Recover from the deadlock*** *when the* ***detection algorithm*** *determines that a* ***deadlock exists****.*

## Recovery from Deadlock

* When a ***detection algorithm determines that a deadlock exists***, the system must ***recover*** from the deadlock.
* The most common solution is ***to roll back one or more transactions*** to break the ***deadlock***. Three actions need to be taken:

1. **Selection of a victim: -** Given a set of ***deadlocked transactions***, we must determine which transaction to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. ***Many factors may determine the cost of a rollback***, including:
2. *How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task.*
3. *How many data items the transaction has used?*
4. *How many more data items the transaction needs for it to complete.*
5. *How many transactions will be involved in the rollback?*
6. **Rollback: -** Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be ***rolled back***. The simplest solution is a ***total rollback***: Abort the transaction and then restart it.
7. **Starvation**. In a system where the selection of victims is ***based primarily on cost factors***, it may happen that the same transaction is always ***picked as a victim***. As a result, this transaction never completes its designated task, thus there is ***starvation***. We must ensure that a transaction can be picked as a victim only a (small) ﬁnite number of times. The most common solution is to include the number of rollbacks in the cost factor.

## Timestamp-Based Protocols

* The ***locking protocols*** that we have described thus far determine the order between every pair of conﬂicting transactions at execution time by the ﬁrst lock that both members of the pair request that involves incompatible modes.
* Another method for ***determining the serializability order*** is to select an ordering among transactions in advance. The most common method for doing so is to use a ***timestamp-ordering scheme.***

**Timestamps**

* With each transaction ***Ti*** in the system, we associate a ***unique ﬁxed timestamp***, denoted by ***TS(Ti).*** This ***timestamp is assigned by the database system*** before the ***transaction Ti starts execution***.
* If a transaction Ti has been assigned ***timestamp TS(Ti***), and a ***new transaction Tj*** enters the system, then ***TS(Ti) < TS(Tj).***
* There are two simple methods for implementing this scheme:

1. *Use the value of the* ***system clock*** *as the timestamp; that is, a transaction’s timestamp is equal to the value of the clock when the transaction enters the system.*
2. *Use a* ***logical counter*** *that is incremented after a new timestamp has been assigned; that is, a transaction’s timestamp is equal to the value of the counter when the transaction enters the system.*

* The ***timestamps of the transactions determine the serializability order***. Thus, if ***TS(Ti) < TS(Tj),*** then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction Ti appears before transaction Tj.
* To implement this scheme, we associate with each data item Q two timestamp values:

1. ***W-timestamp(Q)*** *denotes the largest timestamp of any transaction that executed write(Q) successfully.*
2. ***R-timestamp(Q)*** *denotes the largest timestamp of any transaction that executed read(Q) successfully.*

* These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

## The Timestamp-Ordering Protocol

* The **timestamp-ordering protocol** ensures that any conﬂicting read and write operations are executed in timestamp order. This protocol operates as follows:

*1****. Suppose that transaction Ti issues read(Q).***

1. *If TS(Ti) < W-timestamp(Q), then Ti needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and Ti is rolled back.*
2. *If TS(Ti) ≥ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(Ti).*

*2.* ***Suppose that transaction Ti issues write(Q).***

1. *If TS(Ti) < R-timestamp(Q), then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls Ti back.*
2. *If TS(Ti) < W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls Ti back.*
3. *Otherwise, the system executes the write operation and sets W-timestamp(Q) to TS(Ti).*

* If a transaction Ti is rolled back by the concurrency-control scheme as result of issuance of either a read or writes operation, the system assigns it a new timestamp and restarts it.
* To illustrate this protocol, we consider transactions T25 and T26. Transaction T25 displays the contents of accounts A and B:

**T25: read(B);**

**read(A);**

**display (A + B).**

* Transaction T26 transfers $50 from account B to account A, and then displays the contents of both:

**T26: read(B);**

**B := B − 50;**

**write(B);**

**read(A);**

**A := A + 50;**

**write(A);**

**display (A + B).**

**Validation-Based Protocols**

* The **validation protocol** requires that each transaction Ti executes in two or three different phases in its lifetime, depending on whether it is a **read-only or an update transaction**. The phases are, in order:

1. ***Read phase.*** *The system executes transaction Ti. It reads the values of the various data items and stores them in variables local to Ti. It performs all write operations on temporary local variables, without updates of the actual database.*
2. ***Validation phase****. The validation test is applied to transaction Ti. This determines whether Ti is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction.*
3. ***Write phase.*** *If the validation test succeeds for transaction Ti, the temporary local variables that hold the results of any write operations performed by Ti are copied to the database.* ***Read-only transactions omit this phase.***

* To perform the ***validation test***, we need to know when the various phases of transactions took place. We shall, therefore, associate three different timestamps with each transaction Ti:

1. ***Start (Ti), the time when Ti started its execution.***
2. ***Validation (Ti), the time when Ti ﬁnished its read phase and started its validation phase.***
3. ***Finish (Ti), the time when Ti ﬁnished its write phase.***

* The **validation test** for transaction Ti requires that, for all transactions Tk with TS(Tk) < TS(Ti), one of the following two conditions must hold:

1. ***Finish (Tk) < Start (Ti).*** *Since Tk completes its execution before Ti started, the serializability order is indeed maintained.*
2. *The set of data items written by Tk does not intersect with the set of data items read by Ti, and Tk completes its write phase before Ti starts its validation phase (Start(Ti) < Finish(Tk) < Validation(Ti)). This condition ensures that the writes of Tk and Ti do not overlap. Since the writes of Tk do not affect the read of Ti, and since Ti cannot affect the read of Tk, the serializability order is indeed maintained.*
3. **Read(B)**
4. **B: =B-50 read(A)**
5. **A: =A+50**

**8. <Validate>**

**9. Write(B)**

**10. Write(A)**

**T26**

1. **Read(B)**

**5. Read(A)**

**6. <Validate>**

**7. Display(A+B)**

**T25**

**Figure Schedule 6, a schedule produced by using validation.**

* As an illustration, consider again transactions T25 and T26. Suppose that TS(T25) < TS(T26). Then, the validation phase succeeds in the schedule 6. Note that the writes to the actual variables are performed only after the validation phase of T26. Thus, T25 reads the old values of B and A, and this ***schedule is serializable.***
* The ***validation scheme*** automatically guards against ***cascading rollbacks***, since the actual writes take place only after the transaction issuing the write has committed. However, there is a possibility of starvation of long transactions, due to a sequence of conﬂicting short transactions that cause repeated restarts of the long transaction.
* ***To avoid starvation, conﬂicting transactions must be temporarily blocked, to enable the long transaction to ﬁnish.***
* This ***validation scheme*** is called the **optimistic concurrency-control** scheme since transactions execute optimistically, assuming they will be able to ﬁnish execution and validate at the end. In contrast, locking and timestamp ordering are pessimistic in that they force a wait or a rollback whenever a conﬂict is detected, even though there is a chance that the schedule may be conﬂict serializable.

## Purpose of Concurrency Control

* + ***To enforce Isolation (through mutual exclusion) among conflicting transactions.***
  + ***To preserve database consistency through consistency preserving execution of transactions.***
  + ***To resolve read-write and write-write conflicts.***

**Concurrency Control Techniques**

* **Locking**
* **Timestamp**
* **Optimistic**
* **Multi-version**
* **Lock Granularity**
* Locking and timestamp are conservative/ traditional approach

**Starvation**

* + **Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further while other transaction continues normally**
  + **This may occur, if the waiting method for item locking:** 
    - **Gave priority for some transaction over others**
    - **Problem in Victim selection algorithm- it is possible that the same transaction may consistently be selected as victim and rolled-back .example In Wound-Wait**
  + **Solution**
    - **FIFO**
    - **Allow for transaction that wait for a longer time**
    - **Give higher priority for transaction that have been aborted for many time**

# CHAPTER FIVE

# DATABASE RECOVERY TECHNIQUES

## Recovery Outline and Categorization of Recovery Algorithms

* ***Recovery from transaction failures*** usually means that the ***database is restored to the most recent consistent state just before the time of failure****.* 
  + To do this, the ***system must keep information about the changes that were applied to data items by the various transactions***. This information is typically kept in the ***system log***.
* A typical strategy for recovery may be summarized informally as follows:

1. If there is ***extensive damage to a wide portion of the database*** due to ***catastrophic failure***, such as a ***disk crash***, the recovery method restores a past copy of the database that was backed up to archival storage and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed up log, up to the time of failure.
2. When the ***database on disk is not physically damaged***, and a ***non-catastrophic failure*** has occurred, the recovery strategy is to identify any changes that may cause an inconsistency in the database.

* For ***non-catastrophic failure***, the recovery protocol ***does not need a complete archival copy of the database***. Rather, the entries kept in the online system log on disk are analyzed to determine the appropriate actions for recovery.
* Conceptually, we can distinguish ***two main techniques for recovery from non-catastrophic*** transaction failures:

1. ***Deferred update*** *and*
2. ***Immediate update****.*

***Deferred Update***

* *The* ***deferred update techniques******do not physically update the database on disk until after a transaction reaches its commit point****; then the updates are recorded in the database.*
  + *Before* ***reaching commit,******all transaction updates are recorded in the local transaction workspace or in the main memory buffers that the DBMS maintains****.*
  + *Before commit, the* ***updates are recorded persistently in the log****, and then after commit, the* ***updates are written to the database on disk****.*
* *If a transaction fails before reaching its commit point, it will not have changed the database in any way, so* ***UNDO is not needed****.*
* *It may be* ***necessary to REDO the effect of the operations of a committed transaction from the log****, because their effect may not yet have been recorded in the database on disk. Hence, deferred update is also known as the* ***NO-UNDO/REDO algorithm****.*

***Immediate Update***

* *In the* ***immediate update techniques****, the* ***database may be updated by some operations of a transaction before the transaction reaches its commit point****.* 
  + *However, these operations must also be recorded in the log on disk by* ***force-writing*** *before they are applied to the database on disk, making recovery still possible.*
* *If a transaction fails after recording some changes in the database on disk but before reaching its commit point,* ***the effect of its operations on the database must be undone****; that is, the transaction* ***must be rolled back.***
* *In the general case of* ***immediate update,******both undo and redo may be required during recovery.*** *This technique, known as the* ***UNDO/REDO algorithm****, requires both operations during recovery, and is used most often in practice.*
* *A variation of the algorithm where all updates are required to be recorded in the database on disk before a transaction commits requires undo only, so it is known as the* ***UNDO/NO-REDO algorithm.***

***Caching (Buffering) of Disk Blocks***

* Typically, ***multiple disk pages that include the data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk.***
  + The ***caching of disk pages*** is traditionally an ***operating system function***, but because of its importance to the efficiency of recovery procedures, it is ***handled by the DBMS by calling low-level operating systems routines.***
* *When the* ***DBMS requests action on some item****,* 
  1. ***First it checks the cache directory to determine whether the disk page containing the item is in the DBMS cache.***
  2. ***Second if it is not, the item must be located on disk, and the appropriate disk pages are copied into the cache.***
* *It may be necessary to replace (or flush) some of the cache buffers to make space available for the new item.*
* *Some page replacement strategy similar to these used in operating systems, such as* ***least recently used (LRU)*** *or* ***first-in-first out (FIFO****), or a new strategy that is DBMS-specific can be used to select the buffers for replacement, such as* ***DBMIN or Least-Likely-to-Use.***
* Two main strategies can be employed when flushing a ***modified buffer back to disk.***

1. ***In-place updating****, writes the buffer to the* ***same original disk location,*** *thus overwriting the old value of any changed data items on disk.*
2. ***Shadowing,*** *writes an updated buffer at a* ***different disk location****, so multiple versions of data items can be maintained.*

* In general, the **old value of the data item before updating** is called the **before image (BFIM),** and the **new value after updating** is called the **after image (AFIM).**
* If shadowing is used, both the ***BFIM and the AFIM can be kept on disk***; hence, it is not strictly necessary to maintain a log for recovering.

## Write-Ahead Logging, Steal/No-Steal, and Force/No-Force

* When ***in-place updating*** is used, it is necessary to use ***a log for recovery*.**
* In this case, the ***recovery mechanism*** must ensure that the ***BFIM of the data item is recorded in the appropriate log entry and that the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the database on disk***. This process is generally known as ***write-ahead logging*.**
* Thereare ***two types of log entry information*** included for a write command:

1. ***The information needed for UNDO:*** *The* ***UNDO-type log entries*** *include the old value (BFIM) of the item since this is needed to undo the effect of the operation from the log (by setting the item value in the database back to its BFIM).*
2. ***The information needed for REDO****. A* ***REDO-type log entry*** *includes the new value (AFIM) of the item written by the operation since this is needed to redo the effect of the operation from the log (by setting the item value in the database on disk to its AFIM).*

* ***Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify the rules that govern when a page from the database can be written to disk from the cache:***

1. *If a cache buffer page updated by a transaction* ***cannot be written to disk******before the transaction commits,*** *the recovery method is called* ***a no-steal approach.*** 
   * ***On the other hand, if the recovery protocol allows writing an updated buffer before the transaction commits, it is called steal approach.***
2. *If all pages updated by a transaction are immediately written to disk before the transaction commits, it is called a* ***force approach****. Otherwise, it is called* ***no-force****.* 
   * ***The force rule means that REDO will never be needed during recovery, since any committed transaction will have all its updates on disk before it is committed.***

* The ***deferred update (NO-UNDO)*** recovery scheme follows a ***no-steal approach***. However, typical database systems employ a ***steal/no-force*** *strategy*.
  + *The* ***advantage of steal*** *is that it* ***avoids the need for a very large buffer space to store all updated pages in memory.***
  + *The* ***advantage of no-force*** *is that* ***an updated page of a committed transaction may still be in the buffer when another transaction needs to update it, thus eliminating the I/O cost*** *to write that page multiple times to disk, and possibly to have to read it again from disk.* 
    - ***This may provide a substantial saving in the number of disk I/O operations when a specific page is updated heavily by multiple transactions.***

## Transaction Actions That Do Not Affect the Database

* In general, a transaction ***will have actions that do not affect the database***, such as ***generating and printing messages or reports*** from information retrieved from the database.
* If a ***transaction fails before completion***, we may not want the user to get these reports, since the transaction has failed to complete.
  + If such erroneous reports are produced, ***part of the recovery process would have to inform the user that these reports are wrong, since the user may take an action based on these reports that affects the database. Hence, such reports should be generated only after the transaction reaches its commit point.***
* A common method of dealing with such actions is to issue the commands that generate the reports but keep them as ***batch jobs,*** which are ***executed only after the transaction reaches its commit point.*** If the ***transaction fails***, the ***batch jobs are canceled***.

***NO-UNDO/REDO Recovery Based on Deferred Update***

* The idea behind deferred update is to ***defer or postpone any actual updates to the database on disk until the transaction completes its execution successfully and reaches its commit point***.
* During ***transaction execution***, the ***updates are recorded only in the log and in the cache buffers***.
* We can state a typical ***deferred update protocol*** as follows:

1. ***A transaction cannot change the database on disk until it reaches its commit point.***
2. ***A transaction does not reach its commit point until all its REDO-type log entries are recorded in the log and the log buffer is force-written to disk.***

* Notice that step 2 of this protocol is a restatement of the ***write-ahead logging (WAL)*** protocol.
  + - *Because the database is never updated on disk until after the transaction commits, there is never a need to UNDO any operations.*
    - *REDO is needed in case the system fails after a transaction commits but before all its changes are recorded in the database on disk.* 
      * *In this case, the transaction operations are redone from the log entries during recovery.*
* For ***multiuser systems*** with concurrency control, the concurrency control and recovery processes are interrelated.
  + *Consider a system in which concurrency control uses* ***strict two-phase locking,*** *so the locks on items remain in effect until the transaction reaches its commit point. After that, the locks can be released. This ensures* ***strict and serializable schedules.***
* If a ***transaction is aborted for any reason (say, by the deadlock detection method), it is simply resubmitted, since it has not changed the database on disk***.
  + ***A drawback of the method described here is that it limits the concurrent execution of transactions because all write-locked items remain locked until the transaction reaches its commit point.***
  + ***Additionally, it may require excessive buffer space to hold all updated items until the transactions commit.***
* The method’s main benefit is that transaction operations never need to be undone, for two reasons:

1. ***A transaction does not record any changes in the database on disk until after it reaches its commit point****—that is, until it completes its execution successfully. Hence, a transaction is never rolled back because of failure during transaction execution.*
2. ***A transaction will never read the value of an item that is written by an uncommitted transaction, because items remain locked until a transaction reaches its commit point.*** *Hence,* ***no cascading rollback will occur.***

## Recovery Techniques Based on Immediate Update

* In these techniques, when a transaction issues an ***update command***, the database on disk can be updated immediately, without any need to wait for the transaction to reach its commit point.
  + ***Notice that it is not a requirement that every update be applied immediately to disk; it is just possible that some updates are applied to disk before the transaction commits.***
* Theoretically, we can distinguish ***two main categories of immediate update algorithms***.

1. If the ***recovery technique ensures*** that ***all updates of a transaction are recorded in the database on disk before the transaction commits,*** there is never a need to REDO any operations of committed transactions. This is called the ***UNDO/NO-REDO recovery algorithm.*** 
   * In this method, all updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed. Hence, this method must utilize the force strategy for deciding when updated main memory buffers are written back to disk.
2. If ***the transaction is allowed to commit before all its changes are written to the database***, we have the most general case, known as the ***UNDO/REDO recovery algorithm***. In this case, the ***steal/no-force strategy*** is applied.

## Shadow Paging

* This ***recovery scheme*** does ***not require the use of a log in a single-user environment.***
* In a ***multiuser environment***, a ***log may be needed*** for the concurrency control method.
  + ***Shadow paging*** considers the database ***to be made up of a number of fixed size disk pages (or disk blocks)—say, n—for recovery purposes***.
* A directory with n entries is constructed, where the **ith** entry points to the **ith** database page on disk.
* The ***directory is kept in main memory*** if it is not too large, and all references—read or writes—to database pages on disk go through it.
* When a transaction begins executing, the current directory—whose entries point to the most recent or current database pages on disk—is copied into a shadow directory.
* The shadow directory is then saved on disk while the current directory is used by the transaction.
* During transaction execution, the shadow directory is never modified. When a write item operation is performed, a new copy of the modified database page is created, but the old copy of that page is not overwritten.
* The database thus is returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a ***NO-UNDO/ NO-REDO technique for recovery.***

## The ARIES Recovery Algorithm

* It is used in many relational database-related products of IBM.
* ARIES uses a ***steal/no-force approach for writing***, and it is based on three concepts:

1. ***Write-ahead logging****,* ***repeating history during redo, and logging changes during undo****.*
2. *The second concept, repeating history, means that ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred.* 
   1. ***Transactions that were uncommitted at the time of the crash (active transactions) are undone.***
3. *The third concept,* ***logging during undo****, will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.*

* The ARIES recovery procedure consists of three main steps: ***analysis, REDO, and UNDO.***
  1. **Analysis:** The analysis step identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of the crash.
  2. **Redo**
     + The appropriate point in the log where the REDO operation should start is also determined.
     + The REDO phase actually reapplies updates from the log to the database.
     + Generally, the REDO operation is applied only to committed transactions.
     + In ARIES, every log record has an associated ***log sequence number (LSN)*** that is monotonically increasing and indicates the address of the log record on disk.
     + Each LSN corresponds to a specific change (action) of some transaction.
     + Also, each data page will store the LSN of the latest log record corresponding to a change for that page.
* A log record is written for any of the following actions: **updating a page (write),** c**ommitting a transaction (commit), aborting a transaction (abort), undoing an update (undo),** and **ending a transaction (end)**.

## Recovery in Multidata base Systems

* These databases may even be stored on different types of DBMSs; for example, some DBMSs may be ***relational, whereas others are object oriented, hierarchical, or network DBMSs.***
  + *In such a case, each DBMS involved in the* ***multidata base transaction*** *may have its* ***own recovery technique and transaction manager*** *separate from those of the other DBMSs.*
  + *This situation is somewhat similar to the case of a* ***distributed database management system****, where parts of the database reside at different sites that are connected by a communication network.*
* To ***maintain the atomicity of a multidata base transaction,*** it is necessary to have a two-level recovery mechanism.
* A global recovery manager, or coordinator, is needed to maintain information needed for recovery, in addition to the local recovery managers and the information they maintain (log, tables).
* The coordinator usually follows a protocol called the ***two-phase commit protocol;*** whose two phases can be stated as follows:
* ***Phase 1.*** *When all participating databases signal the coordinator that the part of the multidata base transaction involving each has concluded, the coordinator sends a message prepare for commit to each participant to* ***get ready for committing the transaction.***
  + *Each participating database receiving that message will* ***force-write all log records*** *and needed information for local recovery to disk and then send a ready to commit or OK signal to the coordinator.*
  + *If the force-writing to disk fails or the local transaction cannot commit for some reason, the participating database sends cannot commit or not OK signal to the coordinator.*
  + *If the coordinator does not receive a reply from the database within a certain time out interval, it assumes a not OK response.*
* ***Phase 2****. If all participating databases reply OK, and the coordinator’s vote is also OK, the transaction is successful, and the coordinator sends a commit signal for the transaction to the participating databases.*
* The net effect of the ***two-phase commit protocol*** is that either all participating databases commit the effect of the transaction or none of them do.
  + *In case any of the participants—or the coordinator—fails, it is always possible to recover to a state where either the transaction is committed or it is rolled back.*
* A failure during or before Phase 1 usually requires the ***transaction to be rolled back,*** whereas a failure during Phase 2 means that a successful transaction can ***recover and commit.***

## Database Backup and Recovery from Catastrophic Failures

* So far, all the techniques we have discussed apply to ***non-catastrophic failures***.
* A key assumption has been that the ***system log is maintained on the disk and is not lost as a result of the failure***.
* Similarly, the ***shadow directory must be stored on disk to allow recovery when shadow paging is used.*** 
  + *The recovery techniques we have discussed use the entries in the system log or the shadow directory to recover from failure by bringing the database back to a* ***consistent state.***
* The recovery manager of a DBMS must also be equipped to handle more catastrophic failures such as ***disk crashes.***
  + ***The main technique used to handle such crashes is a database backup, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices.***
* In case of a catastrophic system failure, the latest backup copy can be reloaded from the tape to the disk, and the system can be restarted.
* Data from critical applications such as ***banking, insurance, stock market, and other databases*** is periodically backed up in its entirety and moved to physically separate safe locations.
  + ***To avoid losing all the effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full database backup by periodically copying it to magnetic tape.***
* The system log is usually substantially smaller than the database itself and hence can be backed up more frequently. ***Therefore, users do not lose all transactions they have performed since the last database backup.***
* All committed transactions recorded in the portion of the system log that has been backed up to tape can have their effect on the database redone.
  + ***A new log is started after each database backup.***
  + ***Hence, to recover from disk failure, the database is first recreated on disk from its latest backup copy on tape.***
* Following that, the effects of all the committed transactions whose operations have been recorded in the backed-up copies of the system log are reconstructed.

# CHAPTER SIX

# DISTRIBUTED DATABASE SYSTEMS

## Distributed Database Concepts

* ***Distributed databases*** bring the advantages of distributed computing to the database management domain.
* A distributed computing system consists of a number of processing elements, not necessarily ***homogeneous***, that are ***interconnected by a computer network***, and that cooperate in performing certain assigned tasks.
* As a general goal, ***distributed computing systems partition a big, unmanageable problem into smaller pieces and solve it efficiently in a coordinated manner.***
* The ***economic viability of this approach*** stems from two reasons:

1. ***More computer power is harnessed(connected) to solve a complex task, and***
2. ***Each autonomous (independent) processing element can be managed independently and develop its own applications.***

***Generally***

* ***Distributed database (DDB)***as a collection of multiple logically interrelated databases distributed over a computer network.
* ***Distributed database management system (DDBMS)*** as a *software system that manages a distributed database while making the distribution transparent to the user*.
* A collection of files stored at different nodes of a network and the maintaining of inter relationships among them via ***hyperlinks*** has become a common organization on the Internet, with files of Web pages.
* Examples DDBMS ***Operational database, Analytical database, Hypermedia database***

## Parallel Versus Distributed Technology

* Turning our attention to system architectures, there are two main types of ***multiprocessor system*** architectures that are commonplace:

1. ***Shared memory (tightly coupled) architecture****: Multiple processors share secondary (disk) storage and also share primary memory.*
2. ***Shared disk (loosely coupled) architecture:*** *Multiple processors share secondary (disk) storage but each has their own primary memory.*

* These ***architectures*** enable processors ***to communicate without the overhead of exchanging messages over a network***.
* ***Database management systems*** developed using the above types of architectures are termed ***parallel database management systems*** rather than ***DDBMS***, since they ***utilize parallel processor technology.***
* Another type of ***multiprocessor architecture*** is called ***shared nothing architecture***. In this architecture, every processor has its ***own primary and secondary (disk) memory***, no common memory exists, and the processors communicate over a high-speed interconnection network (bus or switch).

**Advantages of Distributed Databases**

* ***Distributed database management*** has been proposed for various reasons ranging from organizational decentralization and economical processing to greater autonomy.
* We highlight some of these advantages here.

1. ***Management of distributed data with different levels of transparency****:*

* Ideally, a ***DBMS*** should be distribution transparent in the sense of hiding the details of where each file (table, relation) is physically stored within the system.
* The ***EMPLOYEE, PROJECT, and WORKS\_ON*** tables may be fragmented horizontally (that is, ***into sets of rows and stored with possible replication***).
* The following types of transparencies are possible:
* ***Distribution or network transparency:*** This refers **t*o freedom for the user from the operational details of the network***. It may be divided into **location transparency and naming transparency**.

1. ***Location transparency*** *refers to the fact that the command used to perform a task is independent of the location of data and the location of the system where the command was issued.*
2. ***Naming transparency*** *implies that once a name is specified, the named objects can be accessed unambiguously without additional specification.*

* ***Replication transparency:*** ***copies of data may be stored at multiple sites for better availability, performance, and reliability***. Replication transparency makes the user unaware (uninformed) of the existence of copies.
* ***Fragmentation transparency:*** Two types of fragmentation are possible.

1. ***Horizontal fragmentation*** distributes a relation into sets of tuples (rows).
2. ***Vertical fragmentation*** distributes a relation into sub relations where each sub relation is defined by a subset of the columns of the original relation.

**A global query by the user must be transformed into several fragment queries. Fragmentation transparency makes the user unaware of the existence of fragments.**

1. ***Increased reliability and availability:***

* These are two of the most common potential advantages cited for distributed databases.
  + 1. ***Reliability***is broadly defined as the probability that a system is running (not down) at a certain time point.
    2. ***Availability*** is the probability that the system is continuously available during a time interval.
  + When the ***data and DBMS software*** are distributed over several sites, one site may fail while other sites continue to operate.
  + Only the data and software that exist at the failed site cannot be accessed. This improves both reliability and availability. Further improvement is achieved by judiciously replicating data and software at more than one site.
  + *In a* ***centralized system,*** *failure at a single site makes the whole system unavailable to all users.*
  + *In a* ***distributed database****, some of the data may be unreachable, but users may still be able to access other parts of the database.*

1. ***Improved performance:***

* A ***distributed DBMS fragments*** the database by keeping the data closer to where it is needed most.
* *Data localization reduces the contention for CPU and I/O services and simultaneously reduces access delays involved in wide area networks.*
* When a large database is distributed over multiple sites, smaller databases exist at each site.
* ***As a result, local queries and transactions accessing data at a single site have better performance because of the smaller local databases.***
* In addition, each site has a smaller number of transactions executing than if all transactions are submitted to a single centralized database.
* Moreover, inter query and intra query parallelism can be achieved by executing multiple queries at different sites, or by breaking up a query into a number of sub queries that execute in parallel. ***This contributes to improved performance.***

1. ***Easier expansion (scalability):***

* In ***a distributed environment,*** expansion of the system in terms of adding more data, increasing database sizes, or adding more processors is much easier.
* The transparencies we discussed in (1) above lead to a compromise between ease of use and the overhead cost of providing transparency.
* Total transparency provides the global user with a view of the entire DDBS as if it is a single centralized system.
* Transparency is provided as a complement to autonomy, which gives the users tighter control over their own local databases.
* Transparency features may be implemented as a part of the user language, which may translate the required services into appropriate operations. In addition, transparency impacts the features that must be provided by the operating system and the DBMS.

**Disadvantages of Distributed Databases**

* ***Complexity-*** The data replication, failure recovery, network management …makes the system more complex than the central DBMSs.
* ***Cost***- since DDBMS needs more people and more hardware, maintaining and running the system can be more expensive than the centralized system.
* ***Problem of connecting Dissimilar Machine-*** Additional layers of operating system software are needed to translate and coordinate the flow of data between machines.
* ***Data integrity and security problem****-* Because data maintained by distributed systems can be accessed at any locations in the network, controlling the integrity of a database can be difficult.

## Additional Functions of Distributed Databases

* Distribution leads ***to increased complexity in the system design and implementation***.
* To achieve the potential advantages listed previously, the ***DDBMS software must be able to provide the following functions in addition to those of a centralized DBMS:***
* ***Keeping track of data:*** *The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.*
* ***Distributed query processing:*** *The ability to access remote sites and transmit queries and data among the various sites via a communication network.*
* ***Distributed transaction management:*** *The ability to devise/plan execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain integrity of the overall database.*
* ***Replicated data management:*** *The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of a replicated data item.*
* ***Distributed database recovery:*** *The ability to recover from individual site crashes and from new types of failures such as the failure of a communication links.*
* ***Security:*** *Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.*
* ***Distributed directory (catalog) management:*** *A directory contains information (metadata) about data in the database. The directory may be global for the entire DDB, or local for each site. The placement and distribution of the directory are design and policy issues.*
* These functions themselves ***increase the complexity of a DDBMS over a centralized DBMS***.
* Before we can realize the full potential advantages of distribution, we must find satisfactory solutions to these design issues and problems.
* Including all this additional functionality is hard to accomplish, and finding optimal solutions is a step beyond that.
* At the physical hardware level, the following main factors distinguish a DDBMS from a centralized system:
* ***There are multiple computers, called sites or nodes.***
* ***These sites must be connected by some type of communication network to transmit data and commands among sites.***
* The sites may all be located in physical proximity—say, within the same building or group of adjacent buildings—and connected via a local area network, or they may be geographically distributed over large distances and connected via a long-haul or wide area network.
* Local area networks typically use cables, whereas long-haul networks use telephone lines or satellites. It is also possible to use a combination of the two types of networks.

**Data Fragmentation, Replication, and Allocation Techniques for Distributed Database Design**

* *Techniques that are used to* ***break up the database into logical units****, called* ***fragments****, which may be assigned for storage at the various sites.*
* ***Data replication****, which* ***permits certain data to be stored in more than one site****, and the process of allocating fragments—or replicas of fragments—for storage at the various sites.*
* *These techniques are used during the process of* ***distributed database design****.*
* *The information concerning data fragmentation, allocation, and replication is stored in a global directory that is accessed by the DDBS applications as needed.*

## Data Fragmentation

* There are two approaches to store the relation in the distributed database:

**Replication and Fragmentation.**

* ***Data Fragmentation*:** is a technique used to break up the database into logically related units called fragments. A database can be fragmented as:
  + 1. ***Horizontal Fragmentation***
    2. ***Vertical Fragmentation***
    3. ***Mixed (Hybrid) Fragmentation***
* The main reasons for fragmenting a relation are

1. ***Efficiency (good organization)****- data that is not needed by the local applications is not stored*
2. ***Parallelism****- a transaction can be divided into several sub queries that operate on fragments which will increase the degree of concurrency.*

* In a DDB, decisions must be made regarding ***which site should be used to store which portions of the database***.
* Before we decide on how to distribute the data, we must ***determine the logical units of the database that are to be distributed.***

**Horizontal Fragmentation**

* A ***horizontal fragment of a relation is a subset of the tuples in that relation***.
* The tuples that belong to the horizontal fragment are specified by a condition on one or more attributes of the relation. Often, only a single attribute is involved.
* For example, we may define three horizontal fragments on the ***EMPLOYEE relation:*** (DNO= 5), (DNO= 4), and (DNO= 1)—each fragment contains the EMPLOYEE tuples working for a particular department.
* Similarly, we may define three horizontal fragments for the ***PROJECT relation***, with the conditions (DNUM= 5), (DNUM= 4), and (DNUM= 1)—each fragment contains the PROJECT tuples controlled by a particular department.
* ***Horizontal fragmentation divides a relation "horizontally" by grouping rows to create subsets of tuples, where each subset has a certain logical meaning. These fragments can then be assigned to different sites in the distributed system.***
* ***Derived horizontal fragmentation*** applies the partitioning of a primary relation (DEPARTMENT in our example) to other secondary relations (EMPLOYEE and PROJECT in our example), which are related to the primary via a foreign key. This way, related data between the primary and the secondary relations gets fragmented in the same way.

**Vertical Fragmentation**

* Each site may not need all the attributes of a relation, which would indicate the need for a different type of fragmentation.
* ***Vertical fragmentation divides a relation "vertically" by columns.***
* ***A vertical fragment of a relation keeps only certain attributes of the relation.***
* For example, we may want to fragment the EMPLOYEE relation into two vertical fragments.
  + The first fragment includes personal information—***NAME, BDATE, ADDRESS, and SEX****—*and
  + The second includes work-related information—***SSN, SALARY, SUPERSSN, DNO.***
* *This* ***vertical fragmentation is not quite proper*** *because, if the two fragments are stored separately, we cannot put the original employee tuples back together, since there is* ***no common attribute*** *between the two fragments.*
* *It is necessary to include the* ***primary key or some candidate key attribute*** *in every vertical fragment so that the full relation can be reconstructed from the fragments. Hence, we must add the* ***SSN*** *attribute to the* ***personal information fragment****.*

## Data Replication and Allocation

* ***Replication*** is useful in ***improving the availability of data***.
* The most extreme case is replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database.
  + ***This can improve availability remarkably because the system can continue to operate as long as at least one site is up.***
  + ***It also improves performance of retrieval for global queries, because the result of such a query can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module.***
* The **disadvantage of full replication** is that it can ***slow down update operations* *drastically*,** since a single logical update must be performed on every copy of the database to keep the copies consistent. This is especially true if many copies of the database exist.
* **Full replication** makes the ***concurrency control and recovery techniques more expensive than they would be if there were no replication***,
* The other extreme from full replication involves having no replication—that is, each fragment is stored at exactly one site.
* In this case all fragments must be disjoint, except for the repetition of primary keys among vertical (or mixed) fragments. This is also called ***non redundant allocation.***
* ***Between these two extremes, we have a wide spectrum of partial replication of the data—that is, some fragments of the database may be replicated whereas others may not.***
* ***The number of copies of each fragment can range from one up to the total number of sites in the distributed system.***
* A special case of ***partial replication*** is occurring heavily in applications where mobile workers—such as sales forces, financial planners, and claims adjustors—carry partially replicated databases with them on laptops and personal digital assistants and synchronize them periodically with the server database.
* A description of the replication of fragments is sometimes called a ***replication schema.***
* ***Each fragment—or each copy of a fragment—must be assigned to a particular site in the distributed system. This process is called data distribution (or data allocation).***
* The choice of sites and the degree of replication depend on the ***performance and availability goals of the system and on the types and frequencies of transactions submitted at each site***.
* *For example, if high availability is required and transactions can be submitted at any site and if most transactions are retrieval only, a fully replicated database is a good choice.*
* *However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only.*
* *Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication. Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.*

## Types of Distributed Database Systems

* The term ***distributed database management system*** can describe various systems that differ from one another in many respects.
* The main thing that all such systems ***have in common is the fact that data and software are distributed over multiple sites connected by some form of communication network.***
* ***In this section we discuss a number of types of DDBMSs and the criteria and factors that make some of these systems different. The first factor we consider is the degree of homogeneity of the DDBMS software. If all servers (or individual local DBMSs) use identical software and all users (clients) use identical software, the DDBMS is called homogeneous; otherwise, it is called heterogeneous.***
* Another ***factor related to the degree of homogeneity*** is the degree of ***local autonomy***.
* If there is ***no provision for the local site to function as a stand-alone DBMS, then the system has no local autonomy.***
* On the other hand, if direct access by local transactions to a server is permitted, the system ***has some degree of local autonomy***. At one extreme of the autonomy spectrum, we have a DDBMS that "looks like" a centralized DBMS to the user.
* A single conceptual schema exists, and all access to the system is obtained through a site that is part of the DDBMS—which means that ***no local autonomy exists***.
* At the other extreme we encounter a type of DDBMS called a ***federated DDBMS (or a multi database system)***.
* In such a system, each server is ***an independent and autonomous centralized DBMS that has its own local users, local transactions, and DBA and hence has a very high degree of local autonomy***.
* The term ***federated database system (FDBS)*** is used when there is some global view or schema of the federation of databases that is shared by the applications.
* On the other hand, a ***multi database system*** does not have a ***global schema and interactively constructs one as needed by the application***. Both systems are hybrids between distributed and centralized systems and the distinction we made between them is not strictly followed.
* In a ***heterogeneous FDBS***, one server may be a ***relational DBMS***, another network DBMS, and a third an object or hierarchical DBMS; in such a case it is necessary to have a canonical system language and to include language translators to translate sub queries from the canonical language to the language of each server.

## Federated Database Management Systems Issues

* The type of heterogeneity present in FDBSs may arise from several sources.
* We discuss these sources first and then point out how the different types of autonomies contribute to a semantic heterogeneity that must be resolved in a heterogeneous FDBS.

1. ***Differences in data models****: Databases in an organization come from a variety of data models including the so-called* ***legacy models,******the relational data model, the object data model, and even files****.*

*The modeling capabilities of the models vary. Hence, to deal with them uniformly via a single global schema or to process them in a single language is challenging.*

*Even if two databases are both from the RDBMS environment, the same information may be represented as an attribute name, as a relation name, or as a value in different databases.*

*This calls for an intelligent query processing mechanism that can relate information based on metadata.*

1. ***Differences in constraints:*** *Constraint facilities for specification and implementation vary from system to system. There are comparable features that must be reconciled in the construction of a global schema. For example, the relationships from ER models are represented as referential integrity constraints in the relational model. Triggers may have to be used to implement certain constraints in the relational model. The global schema must also deal with potential conflicts among constraints.*
2. ***Differences in query languages:*** *Even with the same data model, the languages and their versions vary. For example, SQL has multiple versions like SQL-89, SQL-92 (SQL2), and SQL3, and each system has its own set of data types, comparison operators, string manipulation features, and so on.*

## Semantic Heterogeneity

* ***Semantic heterogeneity*** occurs when there are differences in the ***meaning, interpretation, and intended use of the same or related data***.
* ***Semantic heterogeneity*** among component database systems (DBSs) creates the biggest hurdle in designing global schemas of heterogeneous databases.
* The design autonomy of component DBSs refers to their freedom of choosing the following design parameters, which in turn affect the eventual complexity of the FDBS:

1. ***The universe of discourse from which the data is drawn:*** *For example, two customer accounts databases in the federation may be from United States and Japan with entirely different sets of attributes about customer accounts required by the accounting practices. Currency rate fluctuations would also present a problem. Hence, relations in these two databases which have identical names—CUSTOMER or ACCOUNT—may have some common and some entirely distinct information.*
2. ***Representation and naming:*** *The representation and naming of data elements and the structure of the data model may be pre specified for each local database.*
3. ***The understanding, meaning, and subjective interpretation of data.*** *This is a chief contributor to semantic heterogeneity.*
4. ***Transaction and policy constraints:*** *this deal with Serializability criteria, compensating transactions, and other transaction policies.*
5. ***Derivation of summaries:*** *Aggregation, summarization, and other data-processing features and operations supported by the system.*

* **Communication autonomy** of a component DBS refers to it stability to decide whether to communicate with another component DBSs.
* **Execution autonomy** refers to the ability of a component DBS to execute local operations without interference from external operations by other component DBSs and its ability to decide the order in which to execute them.
* The ***association autonomy*** of a component DBS implies that it has the ability to decide whether and how much to share its functionality (operations it supports) and resources (data it manages) with other component DBSs.
* The ***major challenge of designing FDBSs*** is to let component ***DBSs interoperate while still providing the above types of autonomies to them***.
* Typical five-level schema architecture to support global applications in the FDBS environment. In this architecture, the local schema is the conceptual schema (full database definition) of a component database, and the component schema is derived by translating the local schema into a canonical data model or common data model (CDM) for the FDBS.
* Schema translation from the local schema to the component schema is accompanied by generation of mappings to transform commands on a component schema into commands on the corresponding local schema.
* The export schema represents the subset of a component schema that is available to the FDBS.
* The federated schema is the global schema or view, which is the result of integrating all the shareable export schemas.
* The external schemas define the schema for a user group or an application, as in the three-level schema architecture.

## Query Processing in Distributed Databases

### Data Transfer Costs of Distributed Query Processing

* In a ***distributed system,*** several additional factors further complicate query processing.
* The first is the **cost of transferring data over the network**.
* This data includes intermediate files that are transferred to other sites for further processing, as well as the final result files that may have to be transferred to the site where the query result is needed.
* Although these costs may not be very high if the sites are connected via a high-performance local area network, they become quite significant in other types of networks.
* Hence, ***DDBMS query optimization algorithms*** consider the goal of reducing the amount of data transfer as an optimization criterion in choosing a distributed query execution strategy.
* We illustrate this with two simple example queries.
  + Suppose that the EMPLOYEE and DEPARTMENT relations of.
    - The size of the EMPLOYEE relation is 100 \* 10,000 = 106 bytes, and
    - The size of the DEPARTMENT relation is 35 \* 100 = 3500 bytes.
* Consider the query Q: "For each employee, retrieve the employee name and the name of the department for which the employee works." This can be stated as follows in the relational algebra:

***Q: pFNAME, LNAME, DNAME (EMPLOYEEDNO=DNUMBER DEPARTMENT)***

* The result of this query will include 10,000 records, assuming that every employee is related to a department.
* Suppose that each record in the query result is ***40 bytes long***.
* The query is submitted at a distinct site 3, which is called the result site because the query result is needed there.
* Neither the EMPLOYEE nor the DEPARTMENT relations reside at site 3.

There are three simple strategies for executing this distributed query:

1. ***Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3. In this case a total of 1,000,000+ 3500 = 1,003,500 bytes must be transferred.***
2. ***Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site the size of the query result is 40 \* 10,000 = 400,000 bytes, so 400,000 + 1,000,000 = 1,400,000 bytes must be transferred.***

If minimizing the amount of data transfer is our optimization criterion, we should choose strategy 3. Now consider another query Q: "For each department, retrieve the department name and the name of the department manager." This can be stated as follows in the relational algebra:

### An Overview of Client-Server Architecture and Its Relationship to Distributed Databases

* Distributed database applications are being developed in the context of the client-server architecture.
* Exactly how to divide the DBMS functionality between client and server has not yet been established.
* ***Different approaches have been proposed. One possibility is to include the functionality of a centralized DBMS at the server level.***
* A number of relational DBMS products have taken this approach, where an SQL server is provided to the clients.
* Each client must then formulate the appropriate SQL queries and provide the user interface and programming language interface functions.
* Since SQL is a relational standard, various SQL servers, possibly provided by different vendors, can accept SQL commands.
* The client may also refer to a data dictionary that includes information on the distribution of data among the various SQL servers, as well as modules for decomposing a global query into a number of local queries that can be executed at the various sites.
* Interaction between client and server might proceed as follows during the processing of an SQL query:

1. ***The client parses a user query and decomposes it into a number of independent site queries. Each site query is sent to the appropriate server site.***
2. ***Each server processes the local query and sends the resulting relation to the client site.***
3. ***The client site combines the results of the sub queries to produce the result of the originally submitted query.***

* In this approach, the SQL server has also been called a ***transaction server (or a database processor (DP) or a back-end machine),*** whereas the client has been called an ***application processor (AP)(or a front-end machine).***
* The interaction between client and server can be specified by the user at the client level or via a specialized DBMS client module that is part of the DBMS package.
  + For example, the user may know what data is stored in each server, break down a query request into site sub queries manually, and submit individual sub queries to the various sites.
  + The resulting tables may be combined explicitly by a further user query at the client level.
  + The alternative is to have the client module undertake these actions automatically.
* In a typical DDBMS, it is customary to divide the ***software modules into three levels:***

1. **The *server software* is responsible for local data management at a site, much like centralized DBMS software.**
2. **The *client software* is responsible for most of the distribution functions; it accesses data distribution information from the DDBMS catalog and processes all requests that require access to more than one site. It also handles all user interfaces.**
3. **The *communications software* (sometimes in conjunction with a distributed operating system) provides the communication primitives that are used by the client to transmit commands and data among the various sites as needed. This is not strictly part of the DDBMS, but it provides essential communication primitives and services.**

* The ***client*** is responsible for generating a distributed execution plan for a multisite query or transaction and for supervising distributed execution by sending commands to servers.
* These commands include ***local queries and transactions to be executed***, as well as ***commands to transmit data to other clients or servers***.
* Hence, client software should be included at any site where multisite queries are submitted. Another function controlled by the client (or coordinator) is that of ensuring ***consistency of replicated copies of a data item by employing distributed (or global) concurrency control techniques***.
* The ***client*** must also ***ensure the atomicity of global transactions by performing global recovery when certain sites fail.***

One possible ***function of the client*** is ***to hide the details of data distribution from the user; that is, it enables the user to write global queries and transactions as though the database were centralized, without having to specify the sites at which the data referenced in the query or transaction resides***. This property is called ***distribution transparency.***

# CHAPTER SEVEN

# Spatial /multimedia/mobile databases

## What is a Spatial Database?

* A spatial database is a database that is enhanced to store and access spatial data or data that defines a geometric space.
* Data on spatial databases are stored as coordinates, points, lines, polygons and topology.
* Spatial RDBMS allows to use SQL data types, such as *int* and *varchar*, as well as spatial data types, such as *Point*, *Line string* and *Polygon* for geometric calculations like distance or relationships between shapes.
* RDBMS uses the **B-Tree** series or **Hash Function** to process indexes; only one-dimensional data can be processed. Since spatial data types are two or three dimensional:
* **R-Tree** series or **Quad Trees** can be used in Spatial RDBMS to process such data;
* Or it is necessary to transform two- or three-dimensional data to one dimensional data, then **B-Tree** can be used.

### Spatial Database Applications

* GIS applications (maps):
  + Urban planning, route optimization, fire or pollution monitoring, utility networks, etc.
* Other applications:
  + VLSI design, CAD/CAM, model of human brain, etc.
* Traditional applications:
  + Multidimensional records

### Spatial Data Types

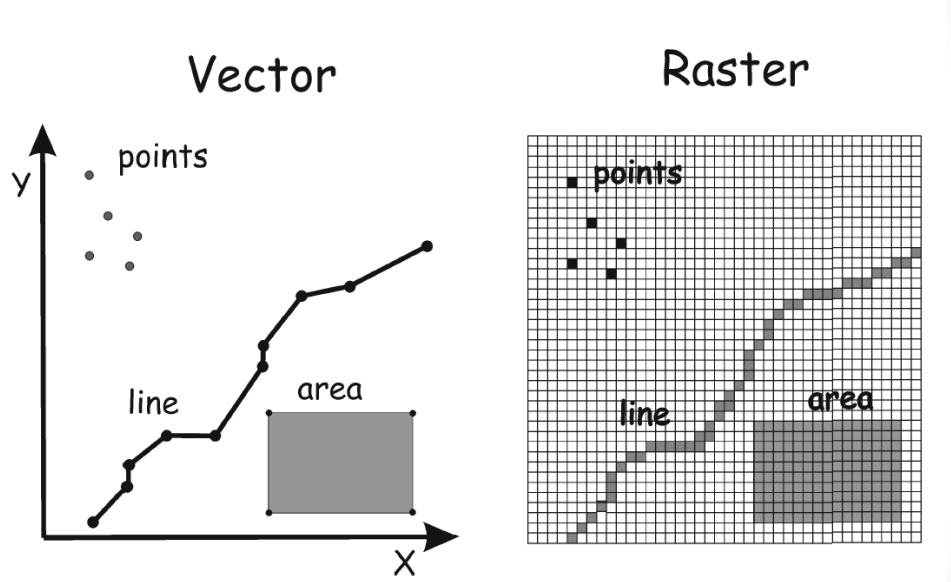
region

line

point

* **Point**: 2 real numbers
* **Line**: sequence of points
* **Region**: area included inside n-points

An Example of Spatial Representation



### Spatial Relationship

* Topological relationships:
  + adjacent, inside, disjoint, etc.
* Direction relationships:
  + Above, below, north of, etc.
* Metric relationships: “distance < 100”

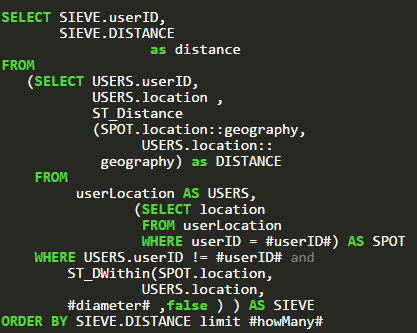
**EXAMPLE**

* A database:
  + Relation states (sname: string, area: region, spop: int)
  + Relation cities (cname: string, center: point; ext: region)
  + Relation rivers (rname: string, route:line)
* SELECT \* FROM rivers WHERE route intersects Ethiopia
* SELECT cname, sname FROM cities, states WHERE center inside area
* SELECT rname, length (intersection (route, California)) FROM rivers WHERE route intersects Oromia

### Spatial Queries

* Selection queries:   
   “Find all objects inside query q”, inside-> intersects, north
* Nearest Neighbor-queries:   
   “Find the closets object to a query point q”, k-closest objects
* Spatial join queries: Two spatial relations S1 and S2, find all pairs: {x in S1, y in S2, and x rel y= true}, rel= intersect, inside, etc

**Example of Spatial SQL Nearest Neighbor search**

****

**Access Methods**

* Point Access Methods (PAMs):
* Index methods for 2 or 3-dimensional points (k-d trees, Z-ordering, grid-file)
* Spatial Access Methods (SAMs):
  + Index methods for 2 or 3-dimensional regions and points (R-trees)

## Mobile Database

A mobile database is either a stationary [**database**](http://en.wikipedia.org/wiki/Database) that can be connected to by a [**mobile computing**](http://en.wikipedia.org/wiki/Mobile_computing) device such as **smart phones** or PDAs over a **mobile network**, or a database which is actually carried by the mobile device. This could be a list of contacts, price information, distance travelled, or any other information.

* + Many applications require the ability to download information from an information repository and operate on this information even when out of range or disconnected.
* Mobile databases are highly concentrated in the retail and logistics industries. They are increasingly being used in aviation and transportation industry.

**Home Directory**

* An example of this is a mobile workforce. In this scenario, a user would require access to update information from files in the home directories on a server or customer records from a database.
* A home directory is a file system directory on a multi-user operating system containing files for a given user of the system.
* The specifics of the home directory (such as its name and location) is defined by the operating system involved; for example, Windows systems between 2000 and 2003 keep home directories in a folder called Documents and Settings.

**Mobile Database Considerations**

* Mobile users must be able to work without a network connection due to poor or even non-existent connections.
* A cache could maintain to hold recently accessed data and transactions so that they are not lost due to connection failure.
* Users might not require access to truly live data, only recently modified data, and uploading of changing might be deferred until reconnected.
* Bandwidth must be conserved (a common requirement on wireless networks that charge per megabyte.
* Mobile computing devices tend to have slower CPUs and limited battery life.
* Users with multiple devices (i.e.: smartphone and tablet) may need to synchronize their devices to a centralized data store. This may require application-specific automation features.
* Users may change location geographically and on the network. Usually dealing with this, is left to the operating system, which is responsible for maintaining the wireless network connection.

**Mobile Database Capabilities**

* Can physically move around without affecting data availability
* Can reach to the place data is stored
* Can process special types of data efficiently
* Not subjected to connection restrictions
* Very high reachability
* Highly portable

**Mobile Database Limitations**

* Limited wireless bandwidth
* Wireless communication speed
* Limited energy source (battery power)
* Less secured
* Vulnerable to physical activities
* Hard to make theft proof.

**Mobile Database Applications**

* Insurance companies
* Emergencies services (Police, medical, etc.)
* Traffic control
* Taxi dispatch
* E-commerce

## Multimedia Database Management System (MMDBMS)

## A true MMDBMS should be able to:

* Operate (*create, update, delete, retrieve*) with at least all the audio-visual multimedia data types (*text, images, video, audio*) and perhaps some non-audio-visual types (*e.g. olfactory, taste, haptic*)
* Fullfil all the standard requirements of a DBMS *i.e. data integrity, ppersistence, transaction management, concurrency control, system recovery, queries, versioning, data integrity, data security, etc.*
* Manipulate huge data volumes (virtually no restriction concerning the number of multimedia structures and their size)
* Allow interaction with the user *e.g. object searches, generated media*
* Retrieve multimedia data based on their content (attributes, features and concepts)
* Efficiently manage data distribution over the nodes of a computer network (distributed architecture)
* By way of combination of basic multimedia objects, it should be possible to create new complex objects (i.e. “specialisation” and “inheritance”)
* Because many different data types are involved, special methods may be needed for optimal storage, access, indexing, and retrieval - in particular, for time-based media objects
* May include advanced features such as character recognition, voice recognition, and image analysis
* With hypermedia databases, “Hyper bases”, links between generated pages/reports are derived from associations between objects
* Many hypermedia systems are based on relational or object-relational database technologies e.g. *Active Server Pages, ColdFusion, Java Server Pages*

## Multimedia Data Types

* **DBMS provide different kinds of domains for multimedia data:**
  + **Large object domains** – these are long unstructured sequences of data e.g. Bbinary Large Objects (BLOBs) or Character Large Objects (CLOBs)
  + **File references** – instead of holding the data, a file reference contains a link to the data
  + MS Windows Object Linking and Embedding (OLE) as in Microsoft Access
  + Actual multimedia domains
    - object-relational databases support specialised multimedia data types
    - object-oriented databases support specialised multimedia classes

## Multimedia Database Models

* **Multimedia File Managers**
  + Programs which organise, index, catalogue, search and retrieve multimedia objects stored as conventional standalone files in the operating system environment
  + May include thumbnail panel describing each object and its file characteristics
  + May also include a “preview” facility, and / or facility to launch associated media-editing tools
  + Vary from simple “scrapbook” catalogue to complex programs offering off-line storage or mounting of files via network
  + May support data compression, such as JPEG
* **Relation DBMS with Extended / Add-on Capabilities**
  + extension of ANSI SQL data types to facilitate storage and manipulation of binary data e.g. BLOBs
  + use of filters / windows / external viewers / OLE to handle rich multimedia data
* **Object-Oriented Databases**
  + handle all kinds of multimedia data is a specialised classis
  + quite low market share in database technology
* **Object-Relational Databases**
  + hybrid combination of features of RDBMS and OODB
  + fully SQL-compliant, with added support for complex data types via object definition language (ODL)
* **Document Management Systems / Content Management Systems**
  + Specialised systems for managing documents/content of many different types
  + Automate workflow, pages/reports generated automatically
  + Very popular for Web sites where content changes rapidly *e.g. Kenny’s Bookshop (www.kennys.ie), Irish Times (ireland.com)*
  + Can buy off-the-shelf (*e.g.* *Terminal Four, Broad Vision*), acquire via open source, or build it yourself! (*e.g. Lotus Notes*)

**Multimedia database management system: system requirements**

* **High network and communications bandwidth**
  + network standards such as asynchronous transfer mode (ATM), fibre distributed data interface (FDDI), and Fast Ethernet
  + improved communications through ISDN, fibre optic technologies, wireless networks, and cable TV
* **High capacity storage, and fast retrieval mechanisms**
  + enhancements in compression algorithms
    - Uncompressed CD = 10 MB/min (30 songs approx. 1GB)
    - MP3 high-quality compression = 1 MB/min (300 songs approx. 1GB)
    - MPEG-AAC (advanced audio) = 500 KB/min (600 songs approx. 1 GB)
  + optical disks and magneto-optic technologies can store many terabytes (TB) of data which is accessible near-line
* **Highly-specified desktop PCs / workstations**
  + desktop machines must be capable of processing multimedia data, and must be fitted with requisite peripheral devices
* **User interfaces**
  + interfaces need to be redesigned to facilitate real-time multi-user interactive multimedia applications

**Multimedia data storage**

* High capacity storage, and fast retrieval mechanisms
* preferred approach storage is HSM (hierarchical storage management)

Main

Memory

(RAM)

On-line devices:

magnetic disks,

optical disks

Off-line devices:

magnetic tapes, optical storage

increasing probability

of access

increasing cost

improving

performance

increasing storage capacity

increasing permanence

increasing access

time

Near-line devices:

optical storage

## Applications of Multimedia Databases

* **Generic business & Office Information Systems**: *document imaging, document editing tools, mmultimedia email, mmultimedia conferencing, mmultimedia workflow management, teleworking*
* **Software development**: *multimedia object libraries; computer-aided software engineering (CASE); multimedia communication histories; multimedia system artefacts*
* **Education**: *multimedia encyclopaedia; multimedia courseware & training materials; education-on-demand (distance education, JIT learning)*
* **Banking**: *tele-banking*
* **Retail:** *home shopping; customer guidance*
* **Tourism & hospitality**: *trip visualisation & planning; sales & marketing*
* **Publishing**: *electronic publishing; document editing tools; multimedia archives*
* **Medicine**: *digitised x-rays; patient histories; image analysis; tele-consultation*
* **Criminal investigation**: *biometrics; fingerprint matching; “photo-fit” analysis*
* **entertainment:** *video-on-demand; interactive TV; Virtual Reality gaming*
* Museums & Libraries
* **science**: *spatial data analysis; cartographic databases; geographic information systems (GIS’s)*
* **engineering**: *computer-aided design / manufacture (CAD / CAM); collaborative design; concurrent engineering*
* **pharmaceuticals**: *dossier management for new drug applications (NDA’s)*

# CHAPTER EIGHT

# WEB- BASED DATABASES

## Databases on the World Wide Web

The World Wide Web (WWW)—popularly known as "the Web"—originally developed in Switzerland at CERN in early 1990 as a large-scale *hypermedia information service system* for biological scientists to share information. Today this technology allows universal access to this shared information to anyone having access to the Internet and the Web contains hundreds of millions of Web pages within the reach of millions of users.

In Web technology, basic client-server architecture underlies all activities. Information is stored on computers designated as Web servers in publicly accessible shared files encoded using *Hyper Text Markup Language (HTML).*

A number of tools enable users to create Web pages formatted with HTML tags, freely mixed with multimedia content from graphics to audio and even to video. A page has many interspersed hyperlinks literally a link that enables a user to "browse" or move from one page to another across the Internet. This ability has given a tremendous power to end users in searching and navigating related information often across different continents.

Information on the Web is organized according to a **Uniform Resource Locator** (URL) something similar to an address that provides the complete pathname of a file. The pathname consists of a string of machine and directory names separated by *slashes and ends in a filename*. For example, the table of contents of this book is currently at the following URL:

A URL always begins with a ***hypertext transport protocol*** *(http),* which is the protocol used by the Web browsers, a program that communicates with the Web server, and vice versa. Web browsers interpret and present HTML documents to users. Popular Web browsers include the Internet Explorer of Microsoft and the Netscape Navigator. A collection of HTML documents and other files accessible via the URL on a Web server is called **a Web site**. In the above URL, "www.awl.com" may be called the Web site of Addison Wesley Publishing.

## Providing Access to Databases on the World Wide Web

Today’s technology has been moving rapidly from static to dynamic Web pages, where content may be in a constant state of flux. The Web server uses a standard interface called the ***Common Gateway Interface* (CGI)** to act as the middleware the additional software layer between the ***user interface******front-end*** and the ***DBMS back-end*** that facilitates access to heterogeneous databases.

The CGI middleware executes external programs or scripts to obtain the dynamic information, and it returns the information to the server in HTML, which is given back to the browser. As the Web undergoes its latest transformations, it has become necessary to allow users access not only to file systems but to databases and DBMSs to support query processing, report generation, and so forth. The existing ***approaches may be divided into two categories:***

1. **Access using CGI scripts:** The database server can be made to interact with the Web server via CGI. The main disadvantage of this approach is that for each user request, the ***Web server must start a new CGI process***: each process makes a new connection with the DBMS and the Web server must wait until the results are delivered to it. No efficiency is achieved by any grouping of multiple users’ requests; moreover, the developer must keep the scripts in the CGI-bin subdirectories only, which opens it to a possible breach of security. The fact that CGI has no language associated with it but requires database developers to learn PERL or Tcl is also a drawback. *Manageability of scripts is another problem if the scripts are scattered everywhere*.
2. **Access using JDBC:** JDBC is a set of Java classes developed by *Sun Microsystems* to allow access to relational databases through the execution of SQL statements. It is a way of connecting with databases, without any additional processes for each client request. Note that JDBC is a name trademarked by Sun; it does not stand for Java Data Base connectivity as many believe. JDBC has the capabilities to connect to a database, send SQL statements to a database and to retrieve the results of a query using the Java classes Connection, Statement, and Result Set respectively. With Java’s claimed platform independence, an application may run on any Java-capable browser, which loads the Java code from the server and runs it on the client’s browser.

The Java code is DBMS transparent; the JDBC drivers for individual DBMSs on the server end carry the task of interacting with that DBMS. If the JDBC driver is on the client, the application runs on the client and its requests are communicated to the DBMS directly by the driver. For standard SQL requests, many RDBMSs can be accessed this way. The drawback of using JDBC is the prospect of executing Java through virtual machines with inherent efficiency. The JDBC bridge to ***Object Database Connectivity (ODBC)*** remains another way of getting to the RDBMSs.

Besides CGI, other Web server vendors are launching their own middleware products for providing multiple database connectivity. These include **Internet Server *API (ISAPI)*** from Microsoft and Netscape API (NSAPI) from Netscape

## The Web Integration Option of INFORMIX

Informix has addressed the limitations of CGI and the incompatibilities of CGI, NSAPI, and ISAPI by creating the ***Web Integration Option (WIO).***

WIO eliminates the need for scripts. Developers use tools to create intelligent HTML pages called **Application Pages** (or App Pages) directly within the database. They execute SQL statements dynamically, format the results inside HTML, and return the resulting Web page to the end users.

Driver, a lightweight CGI process that is invoked when a URL request is received by the Web server. A unique session identifier is generated for each request but the WIO application is persistent and does not terminate after each request. When the WIO application receives a request from the Web driver, it connects to the database and executes Web Explode, a function that executes queries within Web pages and formats results as a Web page that goes back to the browser via the Web driver.

**Informix HTML** tag extensions allow Web authors to create applications that can dynamically construct Web page templates from the Informix Dynamic Server and present them to the end users.

WIO also lets users create their own customized tags to perform specialized tasks. Thus, without resorting to any programming or script development, powerful applications can be designed.

Another feature of WIO helps transaction-oriented applications by providing an application programming interface (API) that offers a collection of basic services such as connection and session management that can be incorporated into Web application.

WIO supports applications developed in C, C++, and Java. This flexibility lets developer’s port existing applications to the Web or develops new applications in these languages. The WIO is integrated with Web server software and utilizes the native security mechanism of the Informix Dynamic Server. The open architecture of WIO allows the use of various Web browsers and servers.

## The ORACLE Web Server

The client requests files that are called "static" or "dynamic" files from the Web server. Static files have a fixed content whereas dynamic files may have content that includes results of queries to the database.

There is an HTTP demon (a process that runs continuously) called Web Listener running on the server that listens for the requests originating in the clients. A static file (document) is retrieved from the file system of the server and displayed on the Web browser at the client. Request for a dynamic page is passed by the listener to a ***Web request broker*** (WRB), which is a multi-threaded dispatcher that adheres to cartridges. Cartridges are software modules that perform specific functions on specific types of data; they can communicate among themselves.

Currently cartridges are provided for PL/SQL, Java, and Live HTML; customized cartridges may be provided as well. Web Server has been fully integrated with PL/SQL, making it efficient and scalable.

The cartridges give it additional flexibility, making it possible to work with other languages and software packages. An advanced secure sockets layer may be used for secure communication over the Internet. The Designer 2000 development has a Web generator that enables previous applications developed for LANs to be ported to the Internet and Intranet environments.

## Open Problems with Web Databases

The Web is an important factor in planning for enterprise-wide computing environments, both for providing external access to the enterprise’s systems and information for customers and suppliers and for marketing and advertising purposes. At the same time, due to security requirements, employees of some organizations are restricted to operate within intranets—sub networks that cannot be accessed freely from the outside world.

Among the prominent applications of the intranet and the WWW are databases to support electronic storefronts, parts and product catalogs, directories and schedules, newsstands, and bookstores. Electronic commerce the purchasing of products and services electronically on the Internet is likely to become a major application supported by such databases.

The future challenges of managing databases on the Web will be many, among them the following:

* Web technology needs to be integrated with the object technology. Currently, the web can be viewed as a distributed object system, with HTML pages functioning as objects identified by the URL.
* HTML functionality is too simple to support complex application requirements. As we saw, the Web Integration Option of Informix adds further tags to HTML. In general, additional facilities will be needed to

1. To make Web clients function as application front ends, integrating data from multiple heterogeneous databases;
2. To make Web clients present different views of the same data to different users; and
3. To make Web clients "intelligent" by providing additional data mining functionality

* Web page content can be made more dynamic by adding more "behavior" to it as an object (In this respect

1. client and server objects (HTML pages) can be made to interact;
2. Web pages can be treated as collections of programmable objects; and
3. Client-side code can access these objects and manipulate them dynamically.

* The support for a large number of clients coupled with reasonable response times for queries against very large (several tens of gigabytes in size) databases will be major challenges for Web databases. They will have to be addressed both by Web servers and by the underlying DBMSs. Efforts are underway to address the limitations of the current data structuring technology, particularly by the World Wide Web Consortium (W3C). The W3C is designing a Web Object Model. W3C is also proposing an Extensible Markup Language (XML) for structured document interchange on the Web.

XML defines a subset of SGML (the Standard Generalized Markup Language), allowing customization of markup languages with application-specific tags. XML is rapidly gaining ground due to its extensibility in defining new tags.

W3C’s Document Object Model (DOM) defines an object-oriented API for HTML or XML documents presented by a Web client. W3C is also defining metadata modeling standards for describing Internet resources.

The technology to model information using the standards discussed above and to find information on the Web is undergoing a major evolution. Overall, the Web servers have to gain robustness as a reliable technology to handle production-level databases for supporting 24x7 applications—24 hours a day, 7 days a week.

Security remains a critical problem for supporting applications involving financial and medical databases. Moreover, transferring from existing database application environments to those on the Web will need adequate support that will enable users to continue their current mode of operation and an expensive infrastructure for handling migration of data among systems without introducing inconsistencies. The traditional database functionality of querying and transaction processing must undergo appropriate modifications to support Web-based applications. One such area is mobile databases.

# CHAPTER NINE

# DATA WAREHOUSING

## What is Data Warehouse?

* A data warehouse is a ***subject oriented, integrated, time-variant, and non-volatile collection of data*.**
* *This* ***data*** *helps analysts* ***to take informed decisions in an organization****.*
* A ***Data Warehouse*** consists of ***data from multiple heterogeneous data sources*** and is ***used for analytical reporting and decision making***.
* Data Warehouse is a ***central place where data is stored from different data sources and applications***.
* An ***operational database*** undergoes frequent changes on a daily basis on account of the transactions that take place.
  + *Suppose a business executive wants to analyze previous feedback on any data such as a product, a supplier, or any consumer data, then the executive will have no data available to analyze because the previous data has been updated due to transactions.*
* A ***data warehouses*** provides us ***generalized and consolidated data in multidimensional view***.
  + Along with generalized and consolidated view of data, a data warehouses also provides us ***Online Analytical Processing (OLAP)*** tools.

These tools help us ***in interactive and effective analysis of data in a multidimensional space***. This analysis results in data ***generalization and data mining***.

* ***Data mining*** functions such as ***association, clustering, classification, prediction*** can be integrated with ***OLAP operations*** to ***enhance the interactive mining of knowledge at multiple level of abstraction.*** That's why data warehouse has now become an important platform for data analysis and online analytical processing.



* In the above image, you can see that the ***data is coming from multiple heterogeneous data sources to a Data Warehouse***. Common data sources for a data warehouse includes.
* *Operational databases*
* *SAP and non-SAP Applications*
* *Flat Files*
* ***Data in data warehouse*** *is accessed by* ***BI Business Intelligence*** *users for Analytical Reporting, Data Mining and Analysis. This is used for decision making by Business Users, Sales Manager, and Analysts to define future strategy.*

## Understanding a Data Warehouse

* A ***data warehouse*** is ***a database***, which is ***kept separate from the organization's operational database.***
* There is ***no frequent updating done*** in a ***data warehouse***.
* It ***possesses consolidated historical data***, which helps the organization ***to analyze its business***.
* A data warehouse ***helps executives to organize, understand, and use their data to take strategic decisions.***
* ***Data warehouse*** systems ***help in the integration of diversity of application systems***.

## Why a Data Warehouse is separated from Operational Databases?

A data warehouses is kept separate from operational databases due to the following reasons −

* An ***operational database*** is constructed for ***well-known tasks and workloads such as searching particular records, indexing, etc.*** In contract, ***data warehouse*** queries are ***often complex and they present a general form of data***.
* ***Operational databases*** support ***concurrent (parallel) processing of multiple transactions***. ***Concurrency control and recovery mechanisms*** are required for *operational databases* ***to ensure robustness and consistency of the database***.
* An Operational Database query allows reading and modifying operations ***insert, delete and Update*** while an ***OLAP query*** needs ***only read-only access of stored data Select statement***.
* An ***operational database*** maintains ***current data of an organization***. On the other hand, a ***data warehouse*** ***maintains historical data***.

### Data Warehouse Applications

* A ***data warehouse*** helps ***business executives to organize, analyze, and use their data for decision making.***
* A data warehouse serves as a ***sole part of a plan-execute-assess "closed-loop" feedback system for the enterprise management.***
* Data warehouses are widely used in the following fields
* *Financial services*
* *Banking services*
* *Consumer goods*
* *Retail sectors*
* *Controlled manufacturing*

### Types of Data Warehouse

***Information processing, analytical processing and data mining*** are the three types of data warehouse applications that are discussed below

1. ***Information Processing*** − A data warehouse allows to process the data stored in it. The data can be processed by ***means of querying, basic statistical analysis, reporting using crosstabs, tables, charts, or graphs.***
2. ***Analytical Processing*** − A data warehouse supports analytical processing of the information stored in it. The data can be analyzed by ***means of basic OLAP operations, including slice-and-dice, drill down, drill up, and pivoting.***
3. ***Data Mining*** − Data mining ***supports knowledge discovery by finding hidden patterns and associations, constructing analytical models, performing classification and prediction***. These mining results can be presented using the visualization tools.

**Characteristics of a Data Warehouse**

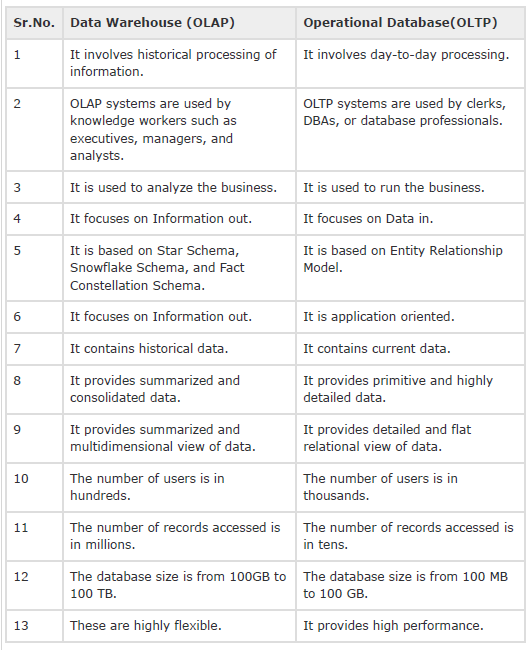
The following are the key characteristics of a Data Warehouse −

1. ***Subject Oriented*** − In a DW system, the ***data is categorized and stored by a business subject*** rather than by application like equity plans, shares, loans, etc.
2. ***Integrated*** − Data from multiple data sources are ***integrated in a Data Warehouse.***
3. ***Non-Volatile*** − Data in data warehouse is ***non-volatile***. It means when data is loaded in DW system, it is not altered.
4. ***Time Variant*** − A DW system contains historical data as compared to Transactional system which contains only current data. In a Data warehouse you can see data for 3 months, 6 months, 1 year, 5 years, etc.

**Note** − A data warehouse does not require transaction processing, recovery, and concurrency controls, because it is physically stored and separate from the operational database.

## OLTP vs. OLAP

* Firstly, OLTP stands for ***Online Transaction Processing***, while OLAP stands for ***Online Analytical Processing***
* In an ***OLTP system***, there ***are a large number of short online transactions such as INSERT, UPDATE, and DELETE.*** Whereas, in an ***OLTP system***, an ***effective measure is the processing time of short transactions and is very less***. It controls data integrity in multi-access environments. For an OLTP system, the number of transactions per second measures the effectiveness.
* An OLTP Data Warehouse System contains current and detailed data and is maintained in the schemas in the entity model 3 NF.
* *For Example − A Day-to-Day transaction system in a retail store, where the customer records are inserted, updated and deleted on a daily basis. It provides faster query processing. OLTP databases contain detailed and current data. The schema used to store OLTP database is the Entity model.*
* *In an* ***OLAP system****, there is lesser number of transactions as compared to a transactional system. The queries executed are complex in nature and involves* ***data aggregations****.*



## Data Warehouse Architecture

* ***Data Warehousing*** involves ***data cleaning, data integration, and data consolidations***.
* A Data Warehouse has a 3-layer architecture.

1. ***Data Source Layer: -***It defines how the data comes to a Data Warehouse. It involves various data sources and operational transaction systems, flat files, applications, etc.
2. ***Integration Layer: -***It consists of ***Operational Data Store and Staging area***. Staging area is used to perform data cleansing, data transformation and loading data from different sources to a data warehouse. As multiple data sources are available for extraction at different time zones, staging area is used to store the data and later to apply transformations on data.
3. ***Presentation Layer: -***This is used to perform BI reporting by end users. The data in a DW system is accessed by BI users and used for reporting and analysis. The following illustration shows the common architecture of a Data Warehouse System.



### What is an Aggregation?

* We save tables with aggregated data like yearly 1row, quarterly 4rows, and monthly 12rows or so, if someone has to do a year to year comparison, only one row will be processed. However, in an unaggregated table it will compare all the rows. This is called Aggregation.
* There are various Aggregation functions that can be used in an OLAP system like Sum, Avg, Max, Min, etc.
* *For Example −SELECT Avg(salary) FROM employee WHERE title = 'Programmer';*

***Key Differences***

*These are the major differences between an* ***OLAP and an OLTP system****.*

* ***Indexes*** *− An* ***OLTP system*** *has only few indexes while in an* ***OLAP system*** *there are many indexes for performance optimization.*
* ***Joins*** *− In an* ***OLTP system****, large number of joins and data are normalized. However, in an* ***OLAP systems*** *there are less joins and are de-normalized.*
* ***Aggregation*** *− In an* ***OLTP system****, data is not aggregated while in an* ***OLAP database*** *more aggregations are used.*
* ***Normalization*** *− An* ***OLTP system*** *contains normalized data however data is not normalized in an OLAP system.*

**

## What is Data Mining?

* ***Data Mining*** is the ***process of extracting useful information and patterns from enormous data***.
* ***Data Mining*** includes ***collection, extraction, analysis and statistics of data.***
* It is also known as ***Knowledge discovery process, Knowledge Mining*** from Data or data/ pattern analysis.
* ***Data Mining*** is a ***logical process of finding useful information to find out useful data***.
* Once the information and patterns are found it can be used to make decisions for developing the business.
* ***Data mining tools*** can give answers to your various questions related to your business which was too difficult to resolve. They also forecast the future trends which lets the business people to make proactive decisions.
* The information or knowledge extracted so can be used for any of the following applications area:
* Market Analysis
* Fraud Detection
* Customer Retention (maintenance)
* Production Control
* Science Exploration

***Data mining involves three steps. They are***

* ***Exploration*** – In this step the data is cleared and converted into another form. The nature of data is also determined
* ***Pattern Identification*** – The next step is to choose the pattern which will make the best prediction
* ***Deployment*** – The identified patterns are used to get the desired outcome.

***Benefits of Data Mining***

* Automated prediction of trends and behaviors
* It can be implemented on new systems as well as existing platforms
* It can analyze huge database in minutes
* Automated discovery of hidden patterns
* There are a lot of models available to understand complex data easily
* It is of high speed which makes it easy for the users to analyze huge amount of data in less time
* It yields improved predictions

***Data Mining Techniques***

* There are several major data mining techniques have been developing and using in data mining projects recently including ***association, classification, clustering, prediction, sequential patterns and decision tree.*** We will briefly examine those data mining techniques in the following sections.

1. ***Association***
   * In association, ***a pattern is discovered based on a relationship between items in the same transaction.***
   * That’s the reason why ***association technique is also known as relation technique.***
   * The association technique is used in market basket analysis to ***identify a set of products that customers frequently purchase together.***
2. ***Classification***

* Classification is a ***classic data mining technique*** ***based on machine learning***.
* Basically, ***classification is used to classify each item in a set of data into one of a predefined set of classes or groups.***
* Classification method makes ***use of mathematical techniques such as decision trees, linear programming, neural network, and statistics.***
* In classification, we develop the software that can learn how to classify the data items into groups.
  + *For example, we can apply classification in the application that “given all records of employees who left the company; predict who will probably leave the company in a future period.”*
* In this case, we divide the records of employees into two groups that named “leave” and “stay”. And then we can ask our data mining software to classify the employees into separate groups.

1. ***Clustering***

* ***Clustering*** is a data mining technique that ***makes a meaningful or useful cluster of objects which have similar characteristics using the automatic technique***.
* The clustering technique ***defines the classes and puts objects in each class, while in the classification techniques, objects are assigned into predefined classes***.
* To make the concept clearer, we can take book management in the library as an example.
  + *In a library, there is a wide range of books on various topics available.*
  + *The challenge is how to keep those books in a way that readers can take several books on a particular topic without hassle.*
  + *By* ***using the clustering technique****, we can keep books that have some kinds of similarities in one cluster or one shelf and label it with a meaningful name.*
  + *If readers want to grab books in that topic, they would only have to go to that shelf instead of looking for the entire library.*

1. ***Prediction***

* The prediction, as its name implied, is one of a data mining technique that ***discovers the relationship between independent variables and relationship between dependent and independent variables.***
* *For instance, the prediction analysis technique can be used in the sale to predict profit for the future if we consider the sale is an independent variable, profit could be a dependent variable.*
* Then based on the historical sale and profit data, we can draw a fitted regression curve that is used for profit prediction.

1. ***Sequential Patterns***

* Sequential patterns analysis is one of data mining technique that ***seeks to discover or identify similar patterns, regular events or trends in transaction data over a business period.***
* In sales, with historical transaction data, businesses can identify a set of items that customers buy together different times in a year.
* Then businesses can use this information to recommend customers buy it with better deals based on their purchasing frequency in the past.

1. ***Decision trees***

* The A decision tree is one of the most commonly used data mining techniques because its model is ***easy to understand for users.***
* In ***decision tree technique,*** the root of the decision tree is a simple question or condition that has multiple answers.
* Each answer then leads to a set of questions or conditions that help us determine the data so that we can make the final decision based on it.

1. ***Data Mining Decision Tree***

* Starting at the root node, if the outlook is overcast then we should definitely play tennis. If it is rainy, we should only play tennis if the wind is the week. And if it is sunny then we should play tennis in case the humidity is normal.
* We often combine two or more of those data mining techniques together to form an appropriate process that meets the business needs.

# Reference

1. *David M. Kroenke. (1998). Database processing, 6th ed. Prentice Hall*
2. *Thomas M. Connolly and Carolyn E.Begg. (2004). A step by step approach to building databases, 2nded.Pearson Education Limited.*
3. *Jeffrey A. Hoffer. V.Ramesh. Heikki Topi. Modern Database Management, Tenth edition.*
4. *Phill Pratt. Mary Last. Concept of Database Management, 8th edition, Cengage Learning.*
5. *Carlos Coronel and Steven Morris. Database System Design, Implementation and Management, 13th edition.*