Contents

CHAPTER ONE

1. Concepts for Object-Oriented Databases

1.1 Overview of Object-Oriented Concepts

The fundamentals of object databases with a specific focus on conceptual modeling of object database designs. The main concepts are EER (Enhanced Entity Relation), LINQ (Language Integrated Query, object databases, object-oriented databases, object-relational databases, UML (Unified Modeling Language).

An Object Database (ODB) aims to make objects persistent, in addition to support the large number of other features of a database system. These expected database features include: the efficient management of persistent data; transactions, concurrency and recovery control; an ad hoc query language. An ODB has to provide these database features in the context of the complexities introduced by object-orientation. That could be challenging.

What is an object? It refers to an abstract concept that generally represents an entity of interest in the enterprise to be modeled by a database application. An object has to reflect a state and some behavior. The object's state shows the internal structure of the object and the internal structure are the properties of the object. We can view a student as an object. The state of the object has to contain descriptive information such as an identifier, a name, and an address.

The behavior of an object is the set of methods that are used to create, access, and manipulate the object. A student object, for example, may have methods to create the object, to modify the object state, and to delete the object. The object may also have methods to relate the object to other objects, such as enrolling a student in a course or assign a note to a student in a course. Objects having the same state and behavior are described by a class.

Given this fundamental definition of an object, the following object-oriented concepts are typically associated with an ODB:

- ✓ class
- ✓ complex objects
- ✓ object identity
- ✓ object structure
- ✓ object constructor
- ✓ method and persistence
- ✓ encapsulation
- ✓ extensibility
- ✓ polymorphism

- ✓ class hierarchies and inheritance (multiple and selective inheritance)
- ✓ overriding, overloading and late binding

1.2 Object Identity, Object Structure, and Type Constructors

Class: A class essentially defines the type of the object where each object is viewed as an instance of the class. For example, Person is a class and Jacqueline is an instance or an object of this class.

Complex Objects: Complex objects or nested relations are objects which are designed by combining simple objects. These objects used to be created by constructors. An airplane and a car are examples of complex objects because they can be viewed as a higher-level object that is constructed from lower-level objects, such as engine and body (for both), the airplane wings and tail. Each of these objects can be complex objects that are constructed from other simple or complex objects.

Object Identity: An object identity is an internal identifier of object that could not be changed. An object identity (iod) remains constant during object lifetime. The oid is use by object database to uniquely identify the object whose attributes can change in contrast to relational database which does not allow the attributes changing for a tupIe in the table. The identity of an object does not depend on properties value, the iod are unique references and they are useful to construct complex objects [Dietriech and Urban, 2011].

Object Structure: A class object contains information about the state described but attributes and the behavior allowed by methods. An attribute and a method that are for the instances of the class are called instance attribute and an instance method. For example, let Student be a class with following attributes: stid, fname, lname. Each student will have his/her stid, fname, lname. So, two different objects (students) of Student class will have different values of instance attributes.

An attribute and a method that are common to all instances of the class are called instance attribute and an instance method. They are called class attributes and class methods. For example, let Student be a class with following attributes: stid, fname, lname, city. The default value of city can be "Bamako" so that all students will get Bamako as city. So, two different objects (students) of Student class will have the same value of class attributes.

Type Constructors: Also called type generator, a type constructor is particular kind of method in a class. It is called to create an object of this class and initialize its attributes. The constructor has the same name as the class. Unlike to ordinary methods, the constructor has not explicit return type because.

1.3 Encapsulation of Operations, Methods, and Persistence

Encapsulation of Operations: Encapsulation consists to gather data and methods within a structure with interface in order to hide the object implementation. The goal is to prevent access to data by any means other than the allowed services defined through interface. Encapsulation therefore guarantees the integrity of the object data. So, the implementation of a class can changes without affecting the interface that the class provides to the rest of the application. Encapsulation is useful to separate class specification level from class implementation level, this is very important as software engineering approach. User-defined types allow object database supporting the encapsulation [Dietriech and Urban, 2011].

Persistence Storing: permanently the values of properties of an object is called object persistence. In other words, persistence refers to the ability for objects to remain after stopping the process that created them [Dogac et al., 1994]. The properties of persistent object are created and stored in main memory. Object-oriented database systems are based on the concept of persistent objects.

Methods: In object-oriented approach, the behavior of an object is described by a set of operations that are called methods. A method has a signature that describes the name of the method and the names and types of the method parameters. A method is a particular implementation of a method signature. They are defined in object class and allow to relate different others objects and manipulating them.

1.4 Type Hierarchies and Inheritance

In object-oriented approach, there is the possible to define class hierarchies and to express the inheritance of state and behavior. Inherited objects are related by “Is-a” relationships. For example, Circle is a Shape and also Triangle is a Shape and we say Circle and Triangle classes inherited the class Shape. Another class called Equilateral Triangle is a Triangle. But we have two hierarchy levels between Shape and Equilateral Triangle.

The ODB have advantages to completely support class hierarchies and inheritance.

Polymorphism: Also called overloading, polymorphism refers to the capability of an object to behave differently to same message. The ODB support polymorphism. For example, there might be a method for area computing like Area () for all Shape objects. However, Circle and Triangle objects can adapt this method to their particular area computation. So, we will get Area () method for Circle object and Area () for Triangle. That is called polymorphism or overloading.

Multiple Inheritances: When an object inherits from only one class, we have simple inheritance. But, when an object inherits from two or more classes, it is multiple inheritances. For example, a Hybrid Car is a Gasoline Car and also Electric Car. So, there is double inheritance because Hybrid Car inherits Gasoline Car and Electric Car.

Late binding: Polymorphism is used with late binding, which means that the translation of an operation name to its appropriate method implementation must be dynamically resolved, i.e. at run-time (Dietrich and Urban, 2011).

1.5 Object-Oriented DBMS Architecture

One of the most abstract OODBMS is the ANSI/SPARC architecture. It consists of following layers: the external user layer, the layer of a conceptual schema, and the layer of physical data.

Another one is the client/server architecture, where database applications are subdivided into two parts: the database server (executing e.g., SQL statements sent by clients) and one or more client's sending requests to the server. There are also more advanced architectures which include three-tier and multi-tier architecture. Essentially, tiers or layers of a user interface and a database are architecture separated by one (or more) layers devoted to business logic.

1.6 Object-Oriented DBMS versus traditional Database

It is obvious that OODBMS products provide traditional database functionality which include persistence, distribution, integrity, concurrency and recovery. However, they are inclined to object modelling. By taking the object orientation, it means they typically provide permanent, immutable object identifiers to guarantee integrity.

Further, Pure OODBMS also provide transparent distributed database capabilities and other advanced DBMS functionality like support for Web, support for workgroups, administrative tools. Thus, OODBMS go beyond the relational database capabilities thus are well suited to handle complex, highly interrelated data, particularly in cross-platform and distributed environment. Above all, the pure OODBMS are able to perform much better than RDBMS due to the new techniques, such as new caching methods, pointer swizzling, navigation via pointer links instead of performing joins, shifting processing on the client side, and others.

1.7 Object-Oriented DBMS drawbacks

There several perceived misgivings concerning OODBMS. They include: Maturity of the technology; Stability of the vendors; Availability of qualified personnel and Cost of conversion thus difficult leveraging the investment already made in RDBMS.

2. Query processing and Optimization

CHAPTER OUTCOME

Students will be able to:

- ✓ Translating SQL Queries into Relational Algebra
- ✓ Basic Algorithms for Executing Query Operations
- ✓ Using Heuristic in Query Optimization
- ✓ Using Selectivity and Cost Estimates in Query Optimization
- ✓ Semantic Query Optimization

2.1 Introduction

Database performance tuning often requires a deeper understanding of how queries are processed and optimized within the database management system. In this set of notes we provide a general overview of how rule based and cost-based query optimizers operate and then provide some specific examples of optimization in commercial DBMS.

Query optimization is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans. Generally, the query optimizer cannot be accessed directly by users: once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs.

2.2 Basic Algorithms for Executing Query Operations

A query: is a high-level specification of a set of objects of interest from the database. It is usually specified in a special language (Data Manipulation Language - DML) that describes what the result must contain.

Query Optimization: aims to choose an efficient execution strategy for query execution.

Query-processing: in object-oriented databases is almost the same as in relation database with only few changes because of semantic differences between relational and object-oriented queries. Moreover, all the techniques applicable to one are also to another. Indeed, without the class hierarchy, queries have similar structures in both databases.

Therefore, in order to simplify, we are going see queries processing without distinguishing between object databases and relational databases.

Query is processed in two phases: the query-optimization phase and the query-processing phase. In order to facilitate the understanding, we will add the query-compilation phase before the two previous phases because queries are viewed by user as Data Manipulation Language (DML) scripts. So, the figure XXX presents the whole process.

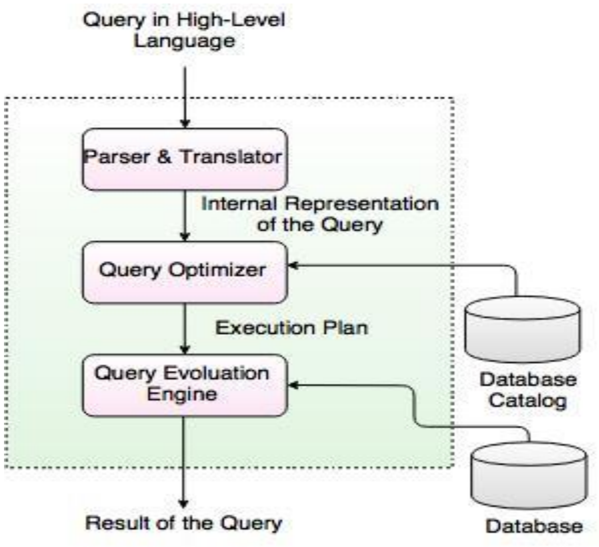


Fig. Query Processing

Query-compilation: DML processor translates DML statements into low-level instructions (compiled query) that the query optimizer can understand.

Query-optimization: the optimizer automatically generates a set of reasonable strategies for processing a given query, and selects one optimal on the basis of the expected cost of each of the strategies generated.

Query-processing: the system executes the query using the optimal strategy generated. In query-optimization, a SQL query is first translated into an equivalent relational algebra expression using a query tree data structure before to be optimized. We will therefore set out below how to pass from a SQL query to an expression in Relational Algebra.

2.3 Translating SQL Queries into Relational Algebra

To translate SQL query in relational algebra, we need to know the different operators in both and the correspondence between them. In module titled "Principle of Database", the relational algebra has been presented in detail. So, we will not do it again.

As in query-optimization, a SQL query is decomposed into query parts also called elementary units. Each part has to be expressed by algebraic operators and optimized. A query unit is a single SELECT-FROM-WHERE-GROUP BY-HAVING expression. Since the two last clauses (GROUP and BY-HAVING) are not necessarily in a query, these clauses do not often appear in query unit. Query having the nested queries is decomposed within separate query units.

Example:

Table 1: Decomposition of a SQL query.

Initial query	Decomposition

SELECT name
FROM clients
WHERE address = (SELECT address
FROM clients
WHERE numClient=76);
	subquery = SELECT address
	FROM clients
	WHERE numClient=76;
	SELECT name
	FROM clients
	WHERE address = subquery;

Among the strategies generated by query optimizer for each unit, one of them is chosen. In above example, the inner unit will be performed only once to get the address of client 76 that will be used by the outer unit.

There are two types of nested queries: uncorrelated and correlated. Uncorrelated nested queries could be performed separately and their results will be used in outer query. Correlated nested queries need information (tuple variable) from outer query in their execution.

The first ones are easier to optimize compared to last ones. The outer query of above example is an uncorrelated one as it can be performed independently from the outer one.

The following table outlines mapping of SQL operators and relational algebra operators

Table 2: From SQL query to Relational Algebra expression.

SQL operators	Relational algebra operators
Select * from table;	table (x1, x2, …, xn)
Select * from table1, table2;	table1(x1, x2, …, xn) · table2(y1, y2, …, ym)
Select Distinct x1, x2 from table;	∠x1,x2table(x1, x2, …, xn)
Select * from table where x1 > x2;	⌠x1, x2table (x1, x2, …, xn)
Select * from table1 UNION Select * from table2;	table1(x1, x2, …, xn) ∪ table2(y1, y2, …, ym)
Select x1, x2, x3 from table1 EXCEPT Select y1, y2, y3 from table2;	table1(x1, x2, x3) − table2(y1, y2, y3)
Select x1, x2, x3 from table1 INTERSECT Select y1, y2, y3 from table2;	table1(x1, x2, x3) ∩ table2(y1, y2, y3)
Select * from table1, table2 where x1•y1;	\x1•y1(table1(x1, x2, x3) − table2(y1, y2, y3))
Select * from table1, table2 where x1=y1;	\x1=y1(table1(x1, x2, x3) − table2(y1, y2, y3))
Select count (*) from table group by x1;	Countx1(table (x1, x2, x3))
Select SUM(x2) from table group by x1;	Sumx1(table (x1, x2, x3), x2

2.4 Using Heuristic in Query Optimization

There several heuristics in query optimization. Some of them are heuristic rules that order operations in a query, and comparing costs of different strategies in order to select one that has minimal cost. Some systems use only heuristics because it is easier and others combine heuristics with partial cost-based optimization. Only heuristic rules are presented as follow.

When we have relative complex query, unary operations (SELECT and PROJECT) must be performed in first. Then binary operations (PRODUCT, JOIN, UNION, DIFFERENCE …) can be performed. Indeed, the size of the result of a binary operation is the product of the sizes of two operands. On another side, unary operations tend to reduce the size of the result. Therefore, the unary operations should be applied before any binary operation.

2.4.1 Process for heuristics optimization

- ✓ The parser of a high-level query generates an initial internal representation;
- ✓ Apply heuristics rules to optimize the internal representation.
- ✓ A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.
    - ➢ 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.

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.

An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.

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. There is only a *single* graph corresponding to each query.

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.

Relation algebra:

$\mathbf{p}_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}}(((\mathbf{s}_{\text{PLOCATION=‘STAFFORD’}}(\text{PROJECT}))\ _{\text{DNUM=DNUMBER}}\ (\text{DEPARTMENT}))\ _{\text{MGRSSN=SSN}}\ (\text{EMPLOYEE}))$

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’;
```

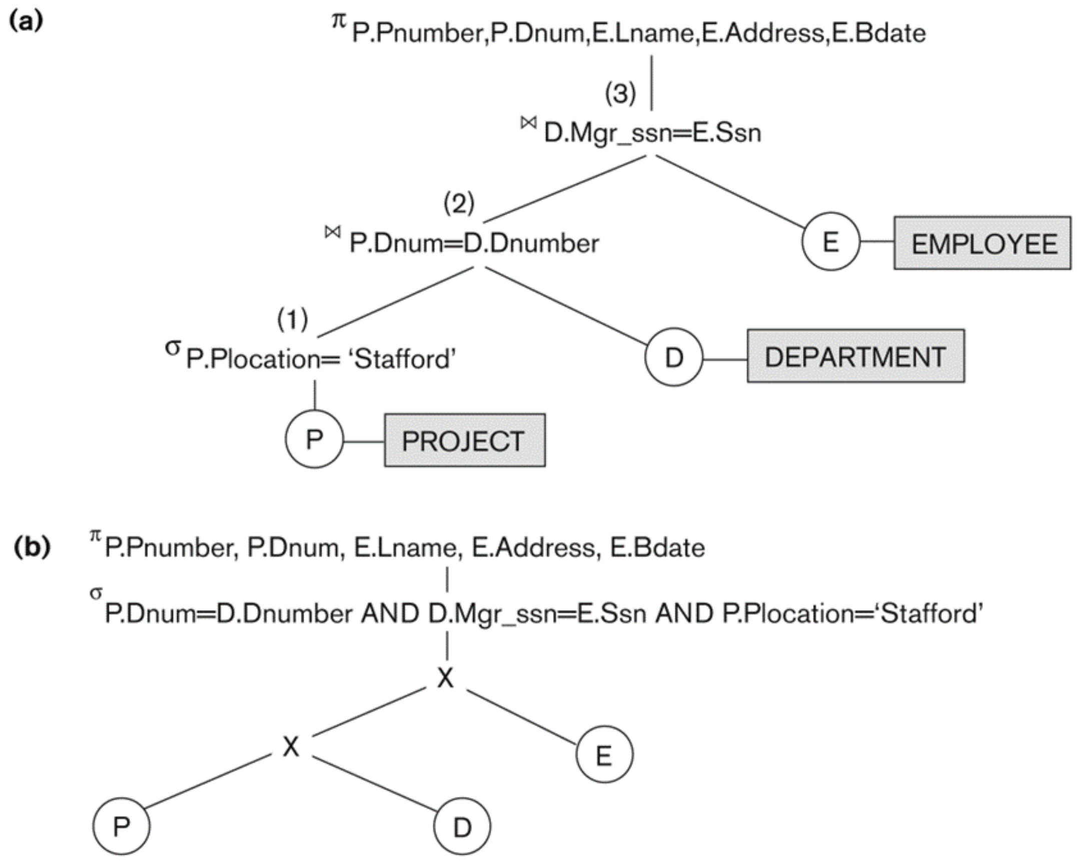

(a)
π P.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate
(3)
⋈ D.Mgr_ssn=E.Ssn
(2)
⋈ P.Dnum=D.Dnumber
E
EMPLOYEE
(1)
σ P.Plocation= 'Stafford'
D
DEPARTMENT
P
PROJECT
(b)
π P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
σ P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND P.Plocation='Stafford'
X
X
E
P
D

Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

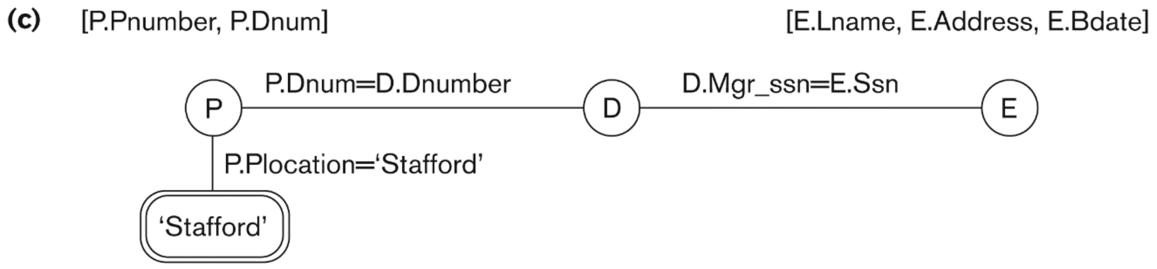


Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

2.4.2 Using Selectivity and Cost Estimates in Query Optimization

- ✓ **Cost-based query optimization**:
  - ➢ Estimate and compare the costs of executing a query using different execution strategies and choose the strategy with the lowest cost estimate.
  - ➢ (Compare to heuristic query optimization)

- ✓ **Issues**
  - ➢ Cost function
  - ➢ Number of execution strategies to be considered
- ✓ **Cost Components for Query Execution**
  1. Access cost to secondary storage
  2. Storage cost
  3. Computation cost
  4. Memory usage cost
  5. 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)

2.4.3 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.

CHAPTER THREE

3. Transaction Processing Concepts

3.1 Transaction

A computer system, like any other system, is subject to various failures like power outage. In this case, the data loss can be catastrophic. The DBMS must then propose a mechanism allowing to recovery of interrupted treatment during execution.

Obviously, it is not convenient, in case of interrupted updating operation, to launch again the new query because in this case the modified tuples before the failure will be modified twice. Also, it is not possible to give up its treatment. In these two cases, the database will become inconsistent.

One of classical examples is an application of travel tickets booking where the number of available seats must be updated by subtracting the number of sold tickets t, and the number of booked seats must be updated also. One of the system constraints is to always check that the sum of the number of seats available seats and the number of booked seats is constant before and after each booking operation. This is the way to ensure that no seat or ticket will be lost. Following orders must be executed:

If the constraint is well respected before and after these two orders, the database will pass by an inconstancy phase between the two orders of UPDATE. A failure between executions of these orders would be dramatic.

The rule to follow is that: the database must be always in a consistent state. It is therefore necessary to define a sequence of operations considered by the system as atomic. This means that either all actions of this sequence are executed, or none performed. In this way, the database can stay consistent according to above constraint. During the execution of these operations, another user might not view the data changing, but only at the end of execution: this is to code isolation. All of these properties lead to transaction concept that aims to preserve the database consistency.

3.1.1 Transaction and Concurrency control

In general, a transaction is any operation on a database system. But, when two transactions are being processed against a database at the same time, they are termed concurrent transactions. Although it may appear to the users that concurrent transactions are being processed simultaneously, this cannot be true because the CPU of the machine processing the database can execute only one instruction at a time. Usually, transactions are interleaved, which means that the operating system switches CPU services among tasks so that some portion of each transaction is carried out in a given interval.

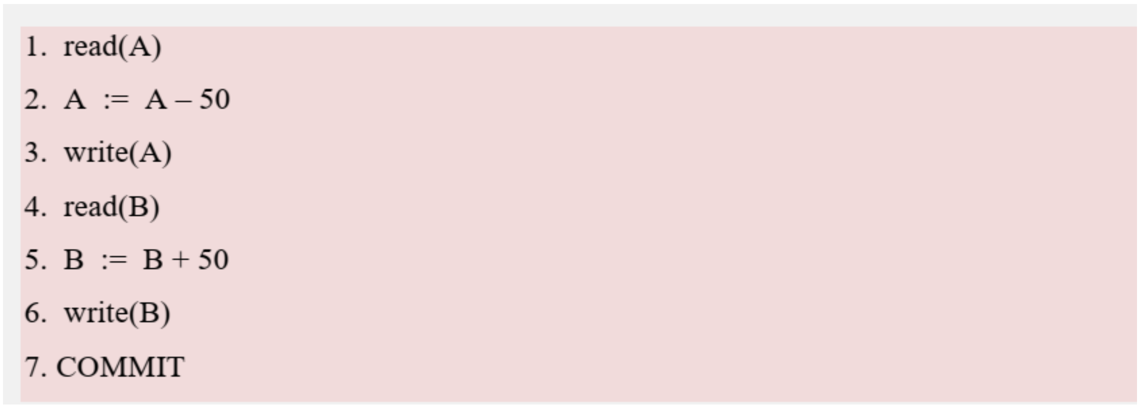
Concurrent execution of transactions is source of inconsistency in the database systems because of interleaving transactions steps. Thus, the timing of individual steps of different transactions needs to be regulated in some manner by a component of DBMS called scheduler (cf. Figure 5).

In order to preserve the correctness of the database in concurrent executing processes, transaction is view as a set of actions that occurs on a consistent database and leave it consistent. Concurrency control is the general process of assuring that transactions preserve consistency when executing simultaneously. A schedule is a sequence of the important actions taken by one or more transactions. The important read and write actions take place in the main-memory buffers, not the disk.

3.2 Properties Of Transaction

3.2.1 Atomicity

- ✓ Either all operations of the transaction are properly reflected in the database or none are.
- ✓ Means either all the operations of a transaction are executed or not a single operation is executed.
- ✓ For example, consider below transaction to transfer Rs. 50 from account A to account B:



```
1. read(A)
2. A := A – 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
7. COMMIT
```

In above transaction if Rs. 50 is deducted from account A then it must be added to account B

3.2.2 Consistency

Execution of a transaction in isolation preserves the consistency of the database. Means our database must remain in consistent state after execution of any transaction. In above example total of A and B must remain same before and after the execution of transaction.

3.2.3 Isolation

Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions.

Intermediate transaction results must be hidden from other concurrently executed transactions. In above example once your transaction start from step one its result should not be access by any other transaction until last step (step 7) is completed.

3.2.4 Durability

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

Once your transaction completed up to step 7 its result must be stored permanently. It should not be removed if system fails.

Transactions, Database items, Read and Write operations and DBMS buffers A transaction is an executing program, forms a logical unit of database processing

- ✓ Txn includes one or more database operations
- ✓ Txn can be embedded in an application program (or it can be a command line query)
- ✓ Txn boundary; begin Txn …..end Txn
- ✓ A single application program can contain many Txns
- ✓ If a Txn is retrieve and no updates, it is called a read only Txn, otherwise read-write
- ✓ Data item can be a record or an entire block (granularity) or could be a single attribute
- ✓ Each data item has a unique name
- ✓ The basic database operation that a Txn can include: read_item(x), write_item(x); x is a program variable
- ✓ The basic access is one disk block from disk to memory

Read_item(x):

1. Find the address of the disk block that contains x
2. Copy the disk block to memory buffer (if not in memory)
3. Copy the item from the buffer to program variable x

Write_item(x):

1. Find the address of the disk block that contains x
2. Copy the disk block to memory buffer (if not in memory)
3. Copy the program variable x into buffer
4. Store the updated buffer to disk

COMMIT_TRANSACTION: This signals a successful end of the transaction so that any Changes (updates) executed by the transaction can be safely committed to the database and Will not be undone

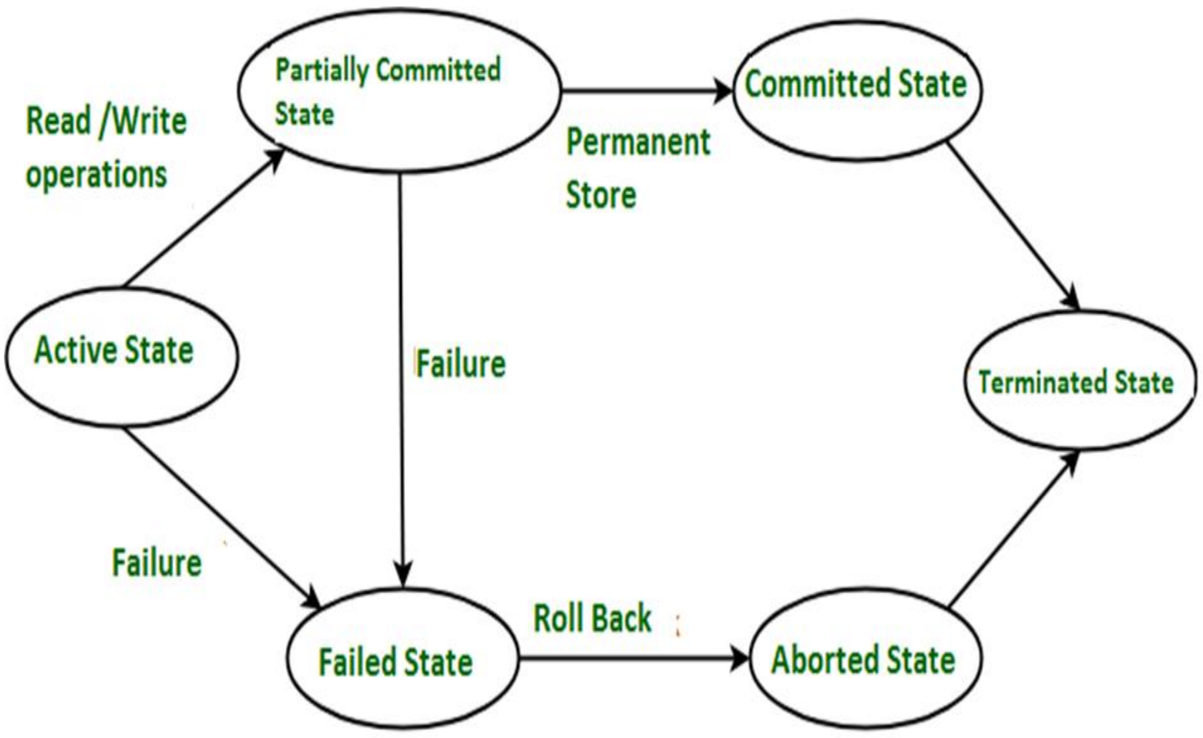
ROLLBACK (or ABORT): This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

State Transition Diagram: Once a transaction is committed, we cannot undo the changes made by the transactions by rolling back the transaction.

Only way to undo the effects of a committed transaction is to execute a compensating transaction. The creating of a compensating transaction can be quite complex and so the task is left to the user and it is not handled by the DBMS. Because failure of transaction may occur, transaction is broken up into states to handle various situations

Following are the different states in transaction processing in database

- ✓ Active
- ✓ Partial committed
- ✓ Failed
- ✓ Aborted
- ✓ Committed



Transaction States in DBMS

Active: This is the initial state. The transaction stay s in this state while it is executing. **Partially Committed**: This is the state after the final statement of the transaction is executed. At this point failure is still possible since changes may have been only done in main memory, a hardware failure could still occur.

The DBMS needs to write out enough information to disk so that, in case of a failure, the system could re-create the updates performed by the transaction once the system is brought back up. After it has written out all the necessary information, it is committed.

Failed: After the discovery that normal execution can no longer proceed. Once a transaction cannot be completed, any changes that it made must be undone rolling it back.

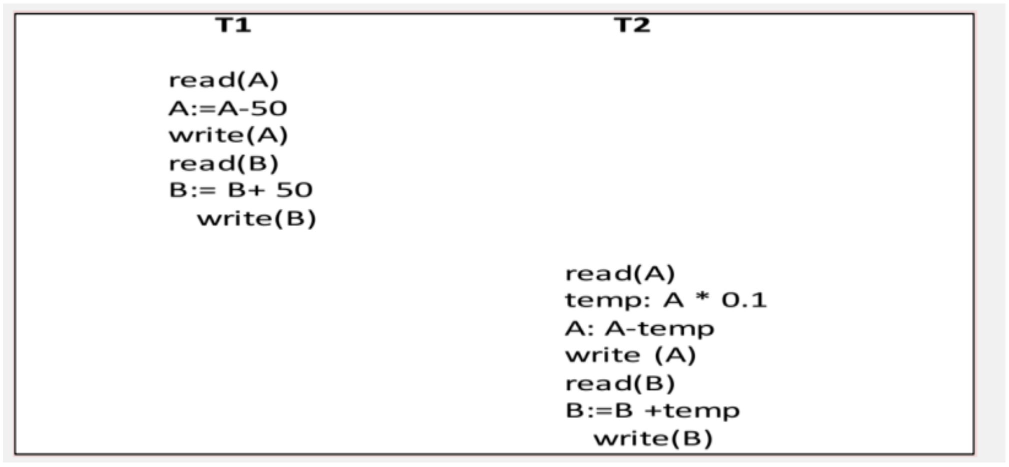
Aborted: The state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Committed: The transaction enters in this state after successful completion of the transaction. We cannot abort or rollback a committed transaction.

3.3 Schedules and Recoverability

3.3.1 Schedule

- ✓ A schedule is the chronological (sequential) order in which instructions are executed I system.
- ✓ A schedule for a set of transaction must consist of all the instruction of those transactions and must preserve the order in which the instructions appear in each individual transaction. Example of schedule (Schedule 1)



T1	T2
read(A)	
A:=A-50	
write(A)	
read(B)	
B:= B+ 50	
write(B)	
	read(A)
	temp: A * 0.1
	A: A-temp
	write (A)
	read(B)
	B:=B +temp
	write(B)

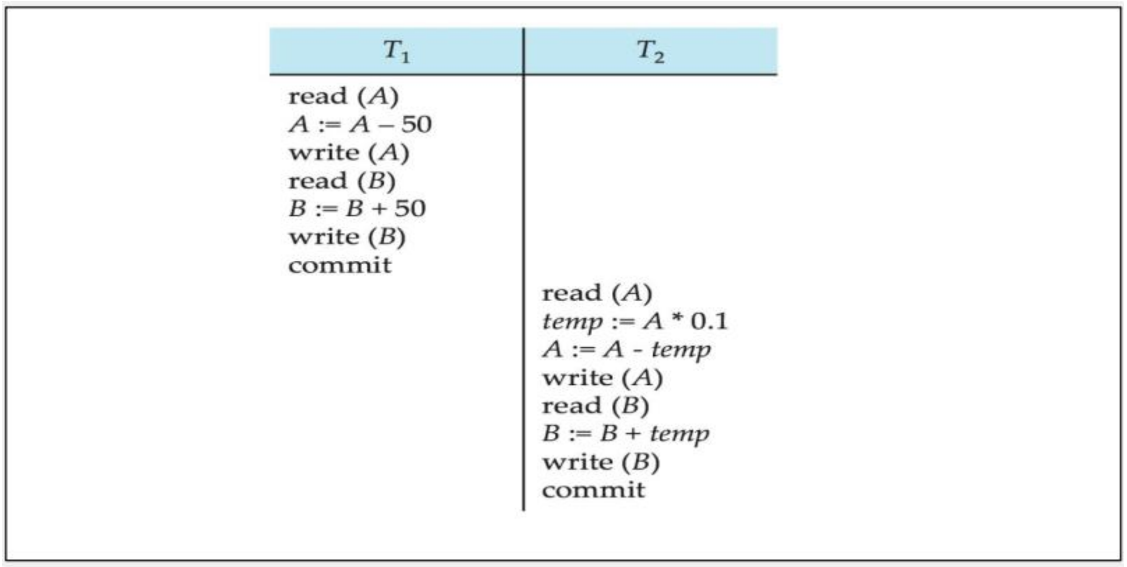
3.3.1.1 Serial schedule

- ✓ Schedule that does not interleave the actions of different transactions.
- ✓ In schedule 1 the all the instructions of T1 are grouped and run together. Then all the instructions of T2 are grouped and run together.

- ✓ Means schedule 2 will not start until all the instructions of schedule 1 are complete. This type of schedules is called serial schedule.

So In Serial schedule, a transaction is executed completely before starting the execution of another transaction. In other words, you can say that in serial schedule, a transaction does not start execution until the currently running transaction finished execution.

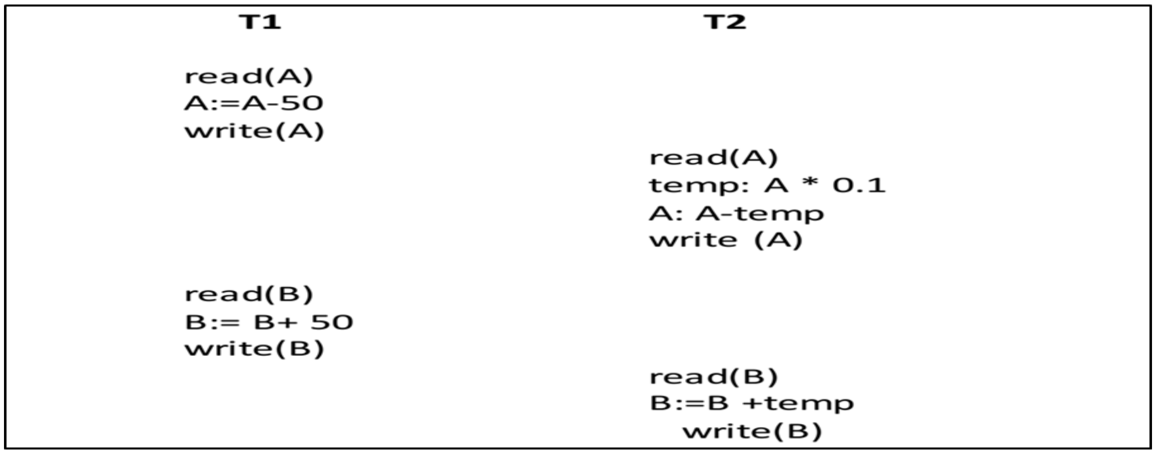
- ✓ This type of execution of transaction is also known as non-interleaved execution.
- ✓ The example we have seen above is the serial schedule.
- ✓ Let T1transfer 50 birr from A to B, and T2 transfer 10% of the balance from A to B. An example of a serial schedule in which T1 is followed by T2



T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

3.3.1.2 Interleaved schedule

Schedule that interleaves the actions of different transactions. Means schedule 2 will start before all instructions of schedule 1 are completed. This type of schedules is called interleaved schedule.



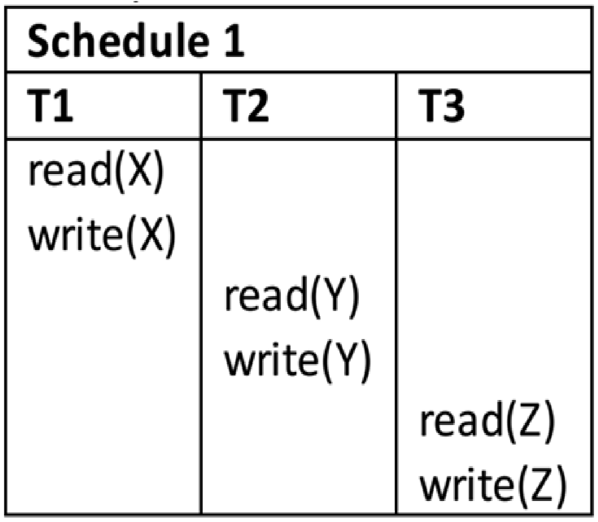
T1	T2
read(A)	
A:=A-50	
write(A)	
	read(A)
	temp: A * 0.1
	A: A-temp
	write (A)
read(B)	
B:= B+ 50	
write(B)	
	read(B)
	B:=B +temp
	write(B)

3.3.1.3 Serializable schedule

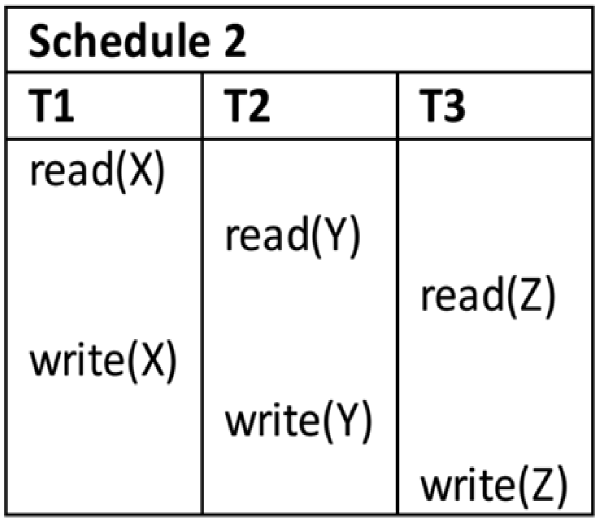
A schedule that is equivalent (in its outcome) to a serial schedule has the serializability property. Two schedules are equivalent schedule if the effect of executing the first schedule is identical (same) to the effect of executing the second schedule.

We can also say that two schedule are equivalent schedule if the output of executing the first schedule is identical (same) to the output of executing the second schedule. A schedule that is equivalent (in its outcome) to a serial schedule has the serializability property.

Example of serializable schedule



Schedule 1		
T1	**T2**	**T3**
read(X)		
write(X)		
	read(Y)	
	write(Y)	
		read(Z)
		write(Z)



Schedule 2		
T1	**T2**	**T3**
read(X)		
	read(Y)	
		read(Z)
write(X)		
	write(Y)	
		write(Z)

In above example there are two schedules as schedule 1 and schedule 2 In schedule 1 and schedule 2 the order in which the instructions of transaction are executed is not the same but whatever the result, we get is same. So, this is known as serializability of transaction.

3.3.1.4 Conflict serializability

Instructions li and lj of transactions Ti and Tj respectively, conflict if and only if there exists some item Q accessed by both li and lj , and at least one of these instructions Wrote Q.

1. If li and lj access different data item then l i and lj don't conflict.
2. li = read(Q), lj = read(Q). li and lj don't conflict.
3. li = read(Q), lj = write(Q). li and lj conflict.
4. li = write(Q), lj = read(Q). li and lj conflict.
5. li = write(Q), lj = write(Q). li and lj conflict.

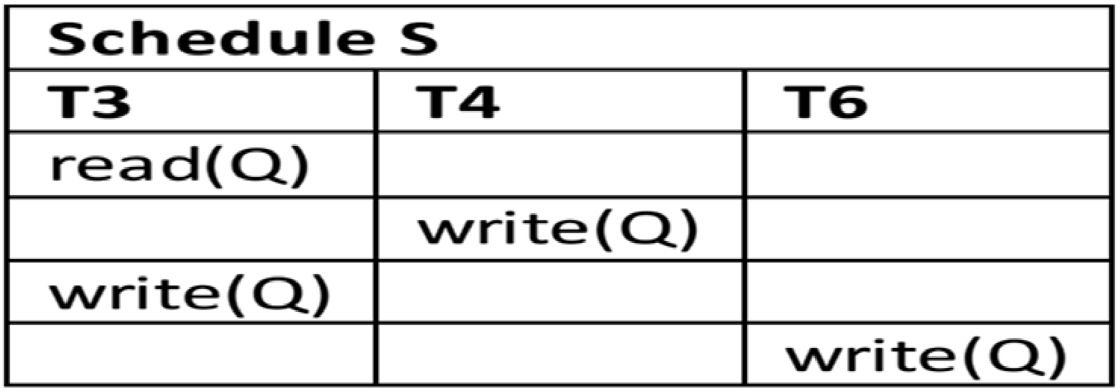
Intuitively, a conflict between li and lj forces a (logical) temporal order between them. If a schedule S can be transformed into a schedule S´ by a series of swaps of non-conflicting instructions, we say that S and S´ are conflict equivalent. We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

View serializability

Let S and S´ be two schedules with the same set of transactions. S and S´ are view Equivalent if the following three conditions are met, for each data item Q,

- ✓ If in schedule S, transaction Ti reads the initial value of Q, then in schedule S' also transaction Ti must read the initial value of Q.
- ✓ If in schedule S transaction Ti executes read(Q), and that value was produced by transaction Tj (if any), then in schedule S' also transaction Ti must read the value of Q that was produced by the same write(Q) operation of transaction Tj .
- ✓ The transaction Ti (if any) that performs the final write(Q) operation in schedule S then in schedule S' also the final write(Q) operation must be performed by Ti .

A schedule S is view serializable if it is view equivalent to a serial schedule. Every conflict serializable schedule is also view serializable but every view serializable is not conflict serializable. Below is a schedule which is view serializable but not conflict serializable.



Schedule S		
T3	**T4**	**T6**
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Above schedule is view serializable but not conflict serializable because all the transactions can use same data item (Q) and all the operations are conflict with each other due to one operation is write on data item (Q) and that's why we cannot interchange any non-conflict operation of any transaction.

Review Questions

1. Discuss what we meant transaction? Why it is interesting? And describe the possible database operation

2. Differentiate Read operation vs Write operation by giving appropriate examples.

3. Compare and contrast transaction properties: Atomicity, Consistency, Isolation and Durability by giving an appropriate example from the real-world scenario.
4. Compare and contrast Transaction state: - Active state, partially committed, Failed state, Aborted State, Committed State by giving an appropriate example from the real-world scenario.
5. Discuss Concurrency Problems in DBMS by using real world example: Dirty Read Problem/uncommitted read, Unrepeatable Read Problem and Lost Update Problem
6. Compare and contrast the Recoverable Schedules: - Cascading Schedule and Cascadeless Schedule use example for each

CHAPTER FOUR

4. Concurrency Control

4.1 Introduction Of Concurrency Controlling

In a multiprogramming environment where more than one transaction can be concurrently executed, there exists a need of protocols to control the concurrency of transaction to ensure atomicity and isolation properties of transactions. Concurrency control protocols, which ensure serializability of transactions, are most desirable.

Concurrency control protocols can be broadly divided into two categories: Concurrency is the ability of a database to allow multiple (more than one) users to access data at the same time.

Concurrency control in database management systems permits many users (assumed to be interactive) to access a database in a multi programmed environment while preserving the illusion that each user has sole access to the system.

Control is needed to coordinate concurrent accesses to a DBMS so that the overall correctness of the database is maintained. For example, users A and B both may wish to read and update the same record in the database at about the same time. The relative timing of the two transactions may have an impact on the state of the database at the end of the transactions. The end result may be an inconsistent database.

Why Concurrent Control is needed?

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

- ✓ The lost update problem: This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of same database item incorrect.

- ✓ The temporary update (or dirty read) problem: This occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.
- ✓ The incorrect summary problem: If one transaction is calculating an aggregate function on a number of records while other transaction is updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Whenever a transaction is submitted to a DBMS for execution, the system must make sure that:

- ✓ All the operations in the transaction are completed successfully and their effect is recorded permanently in the database; or
- ✓ The transaction has no effect whatever on the database or on the other transactions in the case of that a transaction fails after executing some of operations but before executing all of them.

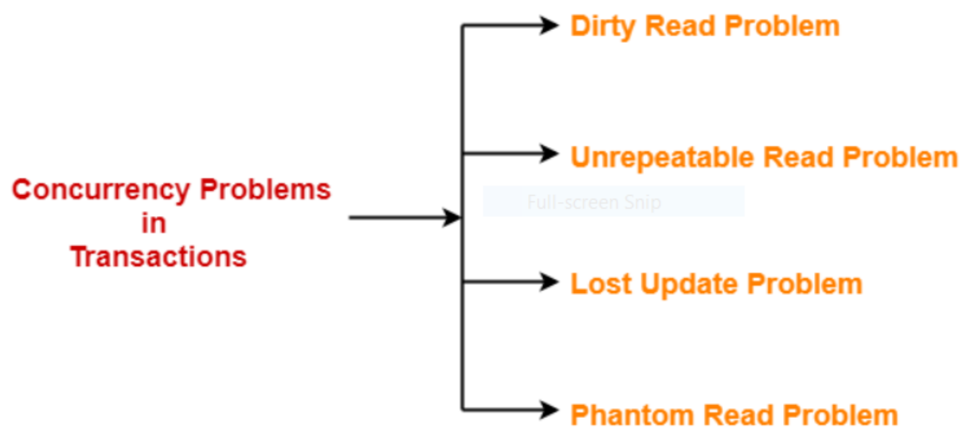
Methods To Control Concurrency (Mechanisms)

Optimistic – Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., serializability and recoverability) until its end, without blocking any of its (read, write) operations and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re -executed, which incurs an obvious overhead. If not, too many transactions are aborted, then being optimistic is usually a good strategy.

Pessimistic – Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.

Semi-optimistic – Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking (if needed) to transaction's end, as done with optimistic.

When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems, such problems are called as concurrency problems, the concurrency problems are Concurrency Problems in DBMS.


Concurrency Problems in Transactions
Dirty Read Problem
Unrepeatable Read Problem
Lost Update Problem
Phantom Read Problem

4.2 Concurrency Control techniques in DBMS

- ✓ Lock-Based Protocols
- ✓ Two Phase Locking Protocol
- ✓ Timestamp-Based Protocols
- ✓ Validation-Based Protocols

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it.

4.2.1 Locking Techniques for Concurrency Control

Types of lock

1. **Shared lock:**
   - ✓ It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
   - ✓ It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.
2. **Exclusive lock:**
   - ✓ In the exclusive lock, the data item can be both reads as well as written by the transaction.
   - ✓ This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

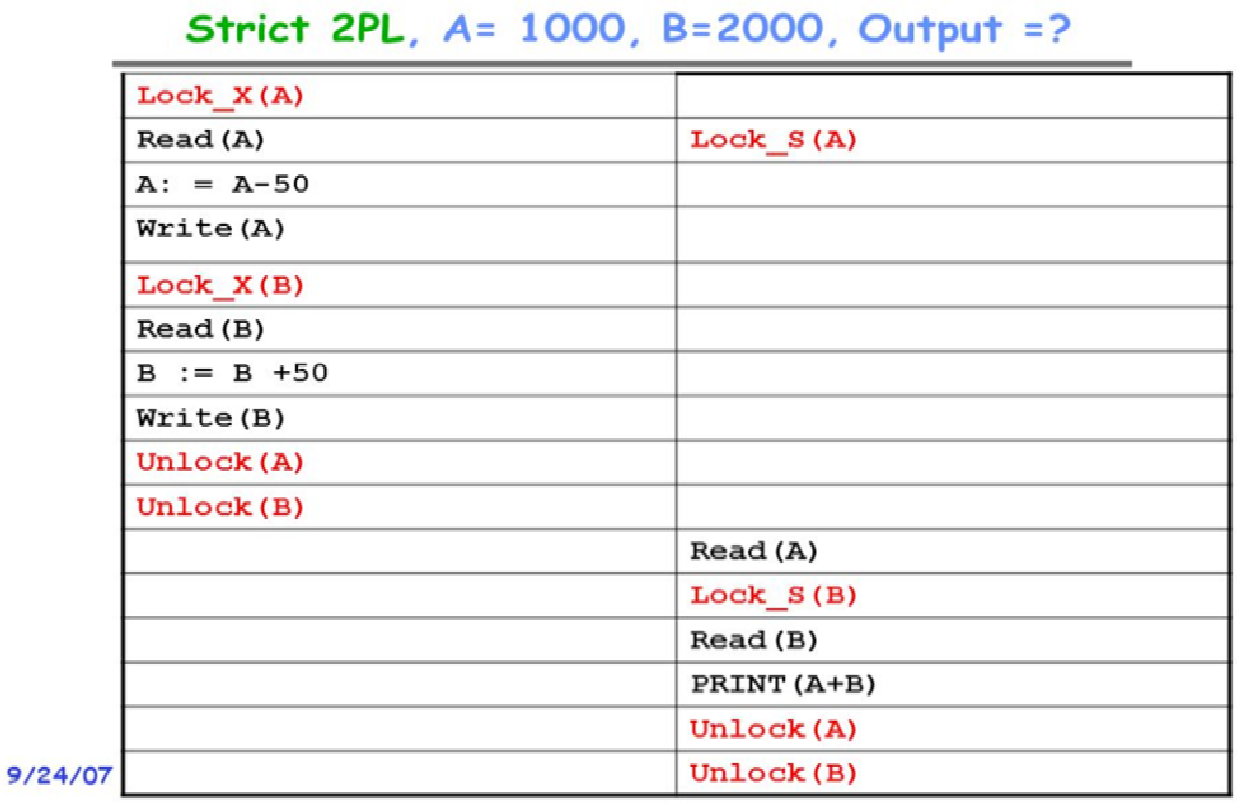
Locking: In order to execute transactions in an interleaved manner it is necessary to have some form of concurrency control. This enables a more efficient use of computer resources. One method of avoiding problems is with the use of locks. When a transaction requires a database object it must obtain a lock.

Locking is necessary in a concurrent environment to assure that one process does not retrieve or update a record that is being updated by another process. Failure to use some controls (locking), would result in inconsistent and corrupt data.

Locks enable a multi-user DBMS to maintain the integrity of transactions by isolating a transaction from others executing concurrently. Locks are particularly critical in write intensive and mixed workload (read/write) environments, because they can prevent the inadvertent loss of data or Consistency problems with reads. In addition to record locking, DBMS implements several other locking mechanisms to ensure the integrity of other data structures that provide shared I/O, communication among different processes in a cluster and automatic recovery in the event of a process or cluster failure.

Aside from their integrity implications, locks can have a significant impact on performance. While it may benefit a given application to lock a large amount of data (perhaps one or more tables) and hold these locks for a long period of time, doing so inhibits concurrency and increases the likelihood that other applications will have to wait for locked resources.

LOCKING RULES: There are various locking rules or mechanisms that are applicable when a user reads or writes a data to a database. The various locking rules or mechanisms are –



Strict 2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Lock_X(B)	
Read(B)	
B := B +50	
Write(B)	
Unlock(A)	
Unlock(B)	
	Read(A)
	Lock_S(B)
	Read(B)
	PRINT(A+B)
	Unlock(A)
	Unlock(B)

9/24/07

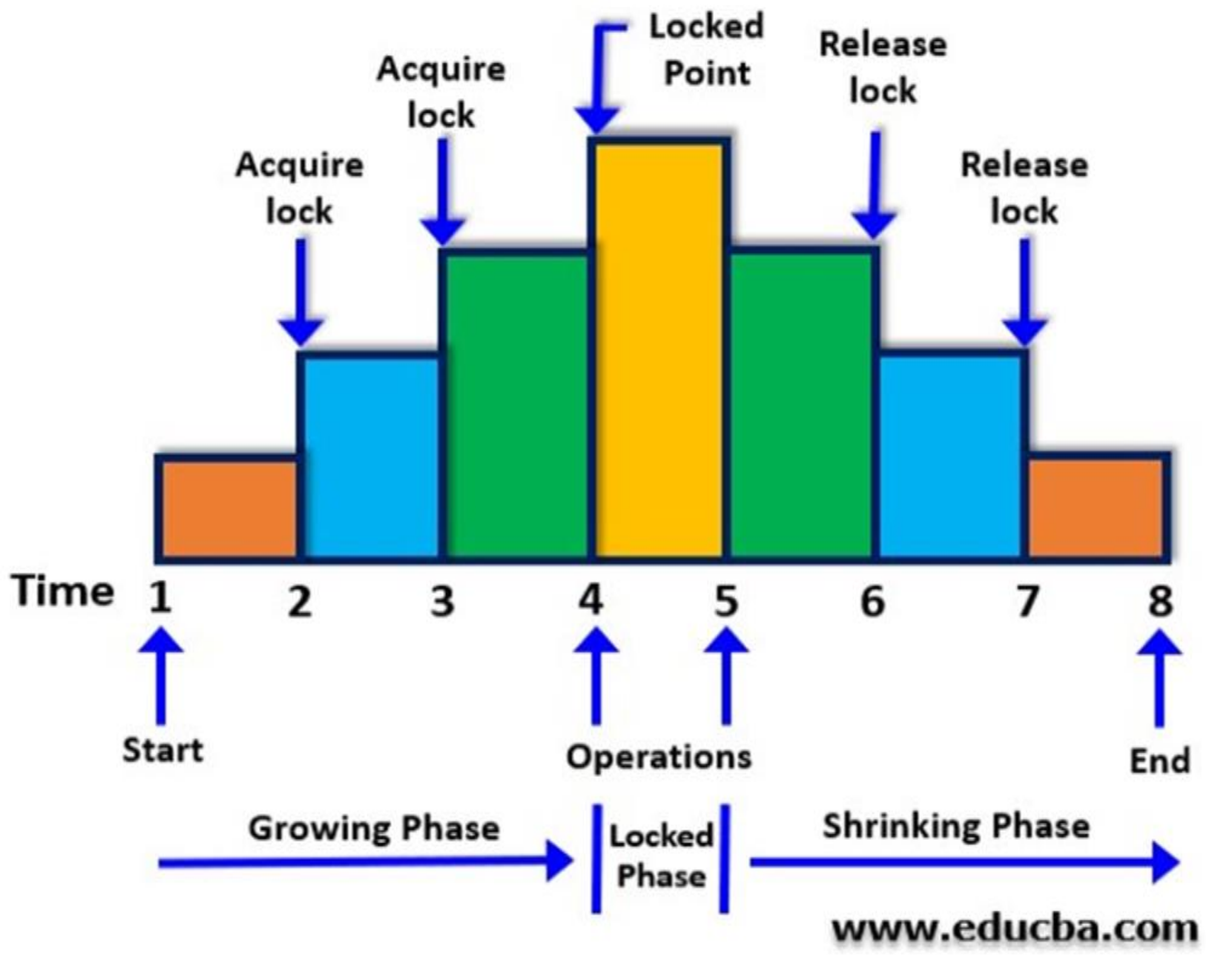
4.2.2 Two-phase locking (2PL)

- ✓ The two-phase locking protocol divides the execution phase of the transaction into three parts.
- ✓ In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- ✓ In the second part, the transaction acquires all the locks.
- ✓ The third phase is started as soon as the transaction releases its first lock.
- ✓ In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.

There are three phases of 2PL: in common

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired



4.2.3 Timestamp-based Protocols

In this Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and writes operations are executed in a timestamp order. The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol. **Example:**

Suppose there are their transactions T1, T2, & T3.

- ✓ T1 has entered the system at time 0010
- ✓ T2 has entered the system at 0020
- ✓ T3 has entered the system at 0030
- ✓ Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.

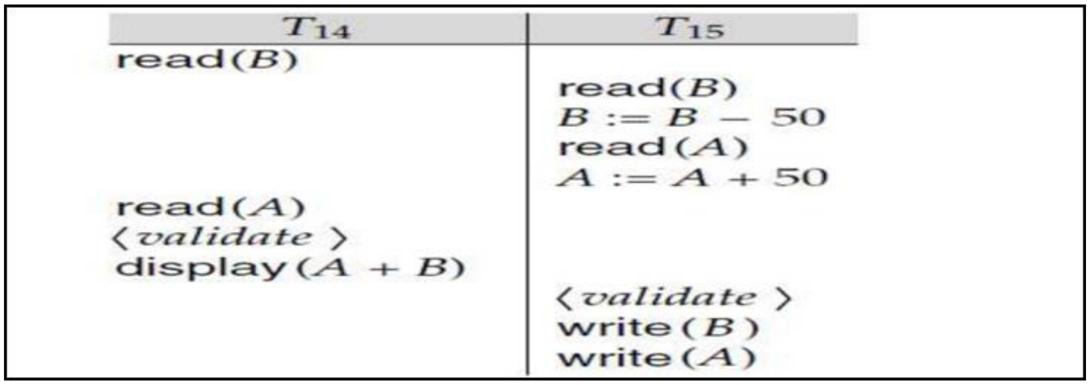
4.2.4 Validation Based Protocol

- ✓ It assumes that multiple transactions can frequently complete without interfering with each other.
- ✓ Before committing, each transaction verifies that no other transaction has modified the data it has read.
- ✓ If check make known conflicting modifications, the committing transaction rolls back and can be restarted.
- ✓ The validation-based protocols require that each transaction Ti executes in two or three different phases in its lifetime.

Read phase: During this phase, 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. Transaction Ti performs a validation test to determine whether it can copy to the database the temporary to local variables that hold the results of write operations without causing a violation of serializability.

Write phase: If transaction Ti succeeds in validation (step 2), then the system applies the actual updates to the database. Otherwise, the system rolls back Ti.

- ✓ Start (Ti), the time when Ti started its execution.
- ✓ Validation (Ti), the time when Ti finished its read phase and started its validation phase.
- ✓ Finish (Ti), the time when Ti finished its write phase.



T_{14}	T_{15}
read(B)	
	read(B)
	$B := B - 50$
	read(A)
	$A := A + 50$
read(A)	
⟨*validate*⟩	
display($A + B$)	
	⟨*validate*⟩
	write(B)
	write(A)

4.3 Multi-version Concurrency Control

Multi-version schemes keep old versions of data item to increase concurrency.

Multi-version 2 phase locking: Each successful write results in the creation of a new version of the data item written. Timestamps are used to label the versions. When a read(X) operation is issued, select an appropriate version of X based on the timestamp of the transaction.

4. Validation Concurrency Control: The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through 2 or 3 phases, referred to as read, validation and write.

- ✓ During read phase, the transaction reads the database, executes the needed computations and makes the updates to a private copy of the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.
- ✓ During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to a write phase. If the validation test is negative, he transaction is restarted and the changes are discarded.
- ✓ During the write phase, the changes are permanently applied to the database.

CHAPTER Five

5. Database Recovery Techniques

5.1 Backup and Recovery Concepts

It is the process of restoring a database to the correct state in the event of before failure. Also, it services that is provided by the DBMS to ensure that the database is reliable and remain in consistent state in case of a failure. In general, recovery refers to the various operations involved in restoring, rolling forward and rolling back a backup.

It the process of restoring the database to the most recent consistence state that exist just before failure. This occur when the database is in the inconsistent state and violate the atomic properties The three states of database recovery are:

- **Precondition**
- **Condition**
- **Post condition**

- ✓ Database recovery is the process of restoring a database to the correct state in the event of a failure
- ✓ Database recovery is a service that is provided by the DBMS to ensure that the database is reliable and remain in consistent state in case of a failure.
- ✓ Restoring a physical backup means reconstructing it and making it available to the database server.
- ✓ To recover a restored backup, data is updated using redo command after the backup was taken.
- ✓ Database server such as SQL server or ORACLE server performs cash recovery and instance recovery automatically after an instance failure.
- ✓ In case of media failure, a database administrator (DBA) must initiate a recovery operation.
- ✓ Recovering a backup involves two distinct operations: rolling the backup forward to a more recent time by applying redo data and rolling back all changes made in uncommitted transactions to their original state.
- ✓ In general, recovery refers to the various operation s involved in restoring, rolling forward and rolling back a backup.

- ✓ Backup and recovery refer to the various strategies and operations involved in Protecting the database against data loss and reconstructing the database

5.2 Log Based Recovery Method

In short Transaction log is a journal or simply a data file, which contains history of all transaction performed and maintained on stable storage. This is used to storing some information about the transaction histories what we done before, when the transaction is fail suddenly it hold what we did before in buffer It used to storing the information about the transaction.

The most widely used structure for recording database modification is the log. The log is a sequence of log records, recording all the update activities in the database.

In short Transaction log is a journal or simply a data file, which contains history of all transaction performed and maintained on stable storage Since the log contains a complete record of all database activity, the volume of data stored in the log may become unreasonable large.

For log records to be useful for recovery from system and disk failures, the log must reside on stable storage.

Log contains:

1. Start of transaction
2. Transaction-id
3. Record-id
4. Type of operation (insert, update, delete)
5. Old value, new value
6. End of transaction that is committed or aborted.

1. All such files are maintained by DBMS itself. Normally these are sequential files.
2. Recovery has two factors Rollback (Undo) and Roll forward (Redo).
3. When transaction Ti starts, it registers itself by writing a <Ti start>log record
4. Before Ti executes write(X), a log record <Ti , X, V1, V2> is written, where V1 is the value of X before the write, and V2 is the value to be written to X.
5. Log record notes that Ti has performed a write on data item Xj
6. Xj had value V1 before the write, and will have value V2 after the write
7. When Ti finishes it last statement, the log record < Ti commit> is written.
8. Two approaches are used in log-based recovery

1. **Deferred database modification**

2. **Immediate database modification**

Log based Recovery Techniques

Once a failure occurs, DBMS retrieves the database using the back-up of database and transaction log. Various log-based recovery techniques used by DBMS are as per below:

1. **Deferred Database Modification**
2. **Immediate Database Modification**

Both of the techniques use transaction logs. These techniques are explained in following sub-sections.

5.2.1 Deferred Database Modification log-based recovery method.

Concept

Updates (changes) to the database are deferred (or postponed) until the transaction commits.

- ✓ During the execution of transaction, updates are recorded only in the transaction log and in buffers. After the transaction commits, these updates are recorded in the database. When failure occurs.
- ✓ If transaction has not committed, then it has not affected the database. And so, no need to do any undoing operations. Just restart the transaction.
- ✓ If transaction has committed, then, still, it may not have modified the database. And so, redo the updates of the transaction.

Transaction Log

- ✓ In this technique, transaction log is used in following ways:
- ✓ Transaction T starts by writing to the log.
- ✓ Any update is recorded as , where V indicates new value for data item X. Here, no need to preserve old value of the changed data item. Also, V is not written to the X in database, but it is deferred.
- ✓ Transaction T commits by writing to the log. Once this is entered in log, actual updates are recorded to the database.
- ✓ If a transaction T aborts, the transaction log record is ignored, and no any updates are recorded to the database.

Example

- ✓ Consider the following two transactions, T 0 and T1 given in figure, where T0 executes before T1. Also consider that initial values for A, B and C are 500, 600 and 700 respectively.

Transaction – T_0	Transaction – T_1
Read (A) A =A -100 Write (A) Read (B) B =B+ 100 Write (B)	Read (C) C=C-200 Write (C)

- ✓ The following figure shows the transaction log for above two transactions at three different instances of time.

Time Instance (a)	Time Instance (b)	Time Instance (c)
<T_0 start> < T_0, A, 400> < T_0, B, 700>	< T_0 start> < T_0, A, 400> < T_0, B, 700> < T_0 commit> <T_1 start> <T_1, C, 500>	< T_0 start> < T_0, A, 400> < T_0, B, 700> < T_0 commit> < T_1 start> < T_1, C, 500> < T_1 commit>

If failure occurs in case of

1. No any REDO actions are required.
2. As Transaction T0 has already committed, it must be redone.
3. As Transactions T0 and T1 have already committed, they must be redone.

5.2.2 Explain Immediate Database Modification log based recovery method.

Concept

- ✓ Updates (changes) to the database are applied immediately as they occur without waiting to reach to the commit point.
- ✓ Also, these updates are recorded in the transaction log.
- ✓ It is possible here that updates of the uncommitted transaction are also written to the database. And, other transactions can access these updated values.

When failure occurs

- ✓ If transaction has not committed, then it may have modified the database. And so, undo the updates of the transaction.

- ✓ If transaction has committed, then still it may not have modified the database. And so, redo the updates of the transaction.

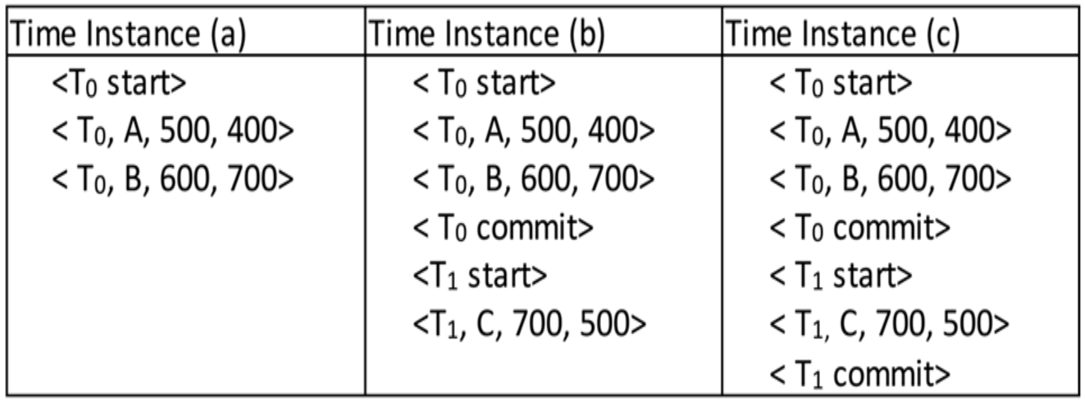
Transaction Log

- ✓ In this technique, transaction log is used in following ways :
- ✓ Transaction T starts by writing to the log.
- ✓ Any update is recorded as where Vold indicates the original value of data item X and Vnew indicates new value for X. Here, as undo operation is required, it requires preserving old value of the changed data item.
- ✓ Transaction T commits by writing to the log.
- ✓ If a transaction T aborts, the transaction log record is consulted, and required undo operations are performed.

Example

Again, consider the two transactions, T 0 and T1, given in figure, where T0 executes before T1. Also consider that initial values for A, B and C are 500, 600 and 700 respectively. The following figure shows the transaction log for above two transactions at three different instances of time. Note that, here, transaction log contains original values also along with new updated values for data items.

If failure occurs in case of – Undo the transaction T0 as it has not committed, and restore A and B to 500 and 600 respectively. Undo the transaction T1, restore C to 700; and, Redo the Transaction T0 set A and B to 400 and 700 respectively. Redo the Transaction T0 and Transaction T0; and, set A and B to 400 and 700 respectively, while set C to 500.



Time Instance (a)	Time Instance (b)	Time Instance (c)
<T_0 start>	< T_0 start>	< T_0 start>
< T_0, A, 500, 400>	< T_0, A, 500, 400>	< T_0, A, 500, 400>
< T_0, B, 600, 700>	< T_0, B, 600, 700>	< T_0, B, 600, 700>
	< T_0 commit>	< T_0 commit>
	<T_1 start>	< T_1 start>
	<T_1, C, 700, 500>	< T_1, C, 700, 500>
		< T_1 commit>

Explain system recovery procedure with Checkpoint record concept

- ✓ Problems with Deferred & Immediate Updates Searching the entire log is time-consuming.
- ✓ It is possible to redo transactions that have already been stored their updates to the database.

Checkpoint

- ✓ A point of synchronization between data base and transaction log file.
- ✓ Specifies that any operations executed before this point are done correctly and stored safely.
- ✓ At this point, all the buffers are force-fully written to the secondary storage.
- ✓ Checkpoints are scheduled at predetermined time intervals Used to limit –
  1. The size of transaction log file
  2. Amount of searching, and
  3. Subsequent processing that is required to carry out on the transaction log file.

  When failure occurs

  - ✓ Find out the nearest checkpoint.
  - ✓ if transaction has already committed before this checkpoint, ignore it
  - ✓ If transaction is active at this point or after this point and has committed before failure, redo that transaction.
  - ✓ If transaction is active at this point or after this point and has not committed, undo that transaction.

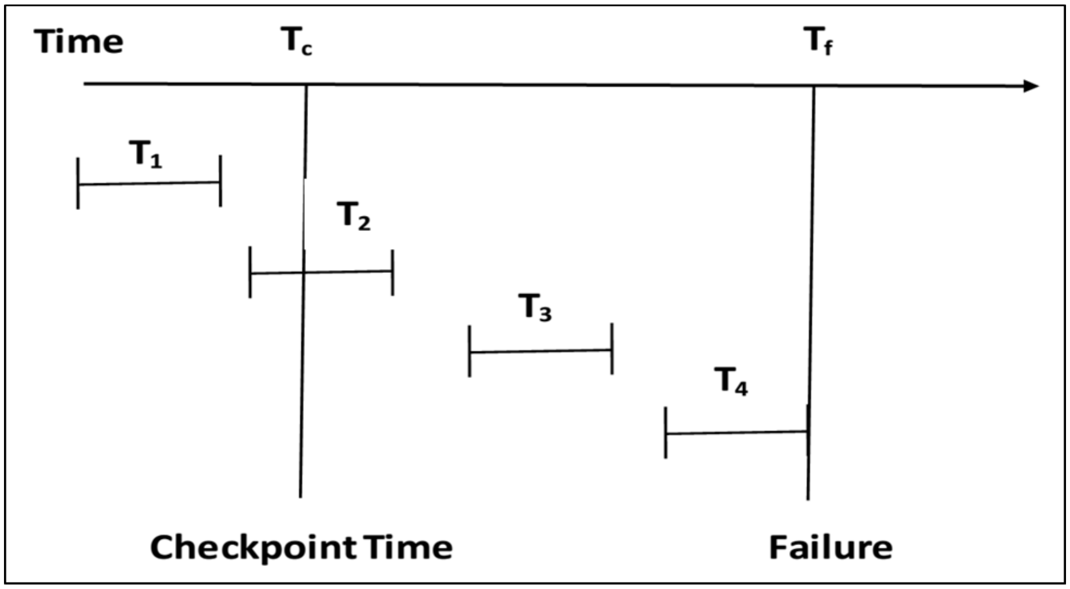
Example:

- ✓ Consider the transactions given in following figure. Here, Tc indicates checkpoint, while Tf indicates failure time. Here, at failure time –

  1. Ignore the transaction T1 as it has already been committed before checkpoint.

  2. Redo transaction T2 and T3 as they are active at/after checkpoint, but have committed before failure.

  3. Undo transaction T4 as it is active after checkpoint and has not committee\d.



5.2.3 Explain Shadow Paging Technique.

Concept

- ✓ Shadow paging is an alternative to transaction-log based recovery techniques.
- ✓ Here, database is considered as made up of fixed size disk blocks, called pages. These pages are mapped to physical storage using a table, called page table.
- ✓ The page table is indexed by a page number of the database. The information about physical pages, in which database pages are stored, is kept in this page table.
- ✓ This technique is similar to paging technique used by Operating Systems to allocate memory, particularly to manage virtual memory.
- ✓ The following figure depicts the concept of shadow paging

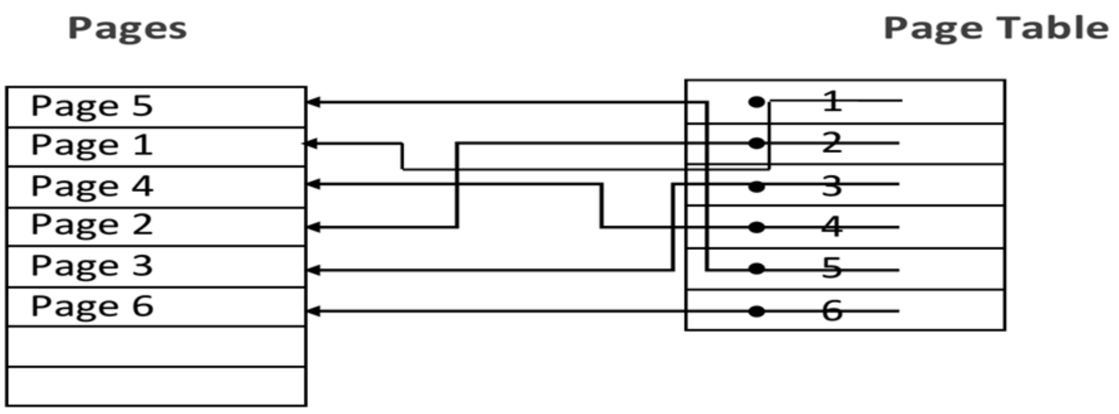
5.2.3.1 Execution of Transaction

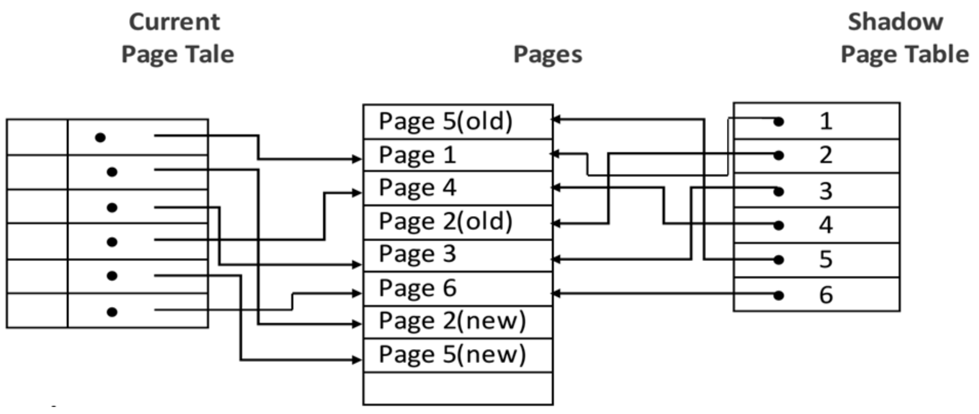
- ✓ During the execution of the transaction, two-page tables are maintained.

1. **Current Page Table:** Used to access data items during transaction execution.
2. **Shadow Page Table:** Original page table, and will not get modified during transaction execution.
   Whenever any page is about to be written for the first time
3. A copy of this page is made onto a free page,

2. The current page table is made to point to the copy,
3. The update is made on this copy.

- ✓ At the start of the transaction, both tables are same and point• to same pages.
- ✓ The shadow page table is never changed, and is used to restore the database in case of any failure occurs. However, current page table entries may change during transaction execution, as it is used to record all updates made to the database.
- ✓ When the transaction completes, the current page table becomes shadow page table. At this time, it is considered that the transaction has committed.
- ✓ The following figure explains working of this technique.
- ✓ As shown in this figure, two pages – page 2 & 5 – are affected by a transaction and copied to new physical pages. The current page table points to these pages.
- ✓ The shadow page table continues to point to old pages which are not changed by the transaction. So, this table and pages are used for undoing the transaction.





Shadow paging technique

Advantages

- ✓ No overhead of maintaining transaction log.
- ✓ Recovery is quite faster, as there is no any redo or undo operations required. Disadvantages
- ✓ Copying the entire page table is very expensive.
- ✓ Data are scattered or fragmented.
- ✓ After each transaction, free pages need to be collected by garbage collector. Difficult to extend this technique to allow concurrent transactions

5.3 ARIES Recovery Algorithm

The previous sections answered the question: what extra information do we maintain to, upon crash, reconstruct the database to the state it was in prior to the crash? In this section, we answer the algorithm question: ***how would we use this information to actually reconstruct the database?***

When a database crashes, all we have are the logs that got saved to disk. The ARIES recovery algorithm has 3 phases that occur in the following order:

1. Analysis: reconstruct transaction and dirty page tables

2. Redo: repeat operations (for durability)
3. Undo: undo operations from transactions in-progress during the crash (for atomicity)

Analysis

In the analysis phase, we scan through the log from the start to reconstruct the transaction and dirty page tables. For each record in the log, we act according to the following rules:

- ✓ If the record is not an END:
    - Add transaction to the transaction table if it does not exist
    - Set transaction's lastLSN to the record's LSN
- ✓ If the record is an UPDATE:
    - If the modified page is not in the DPT, add it to the DPT
    - Set the page's recLSN to the record's LSN
- ✓ If the record is a COMMIT or an ABORT:
    - Change the transaction's status accordingly in the transaction table
- ✓ If the record is an END:
    - Remove the transaction from the transaction table

Something else to consider is that transactions may have been in the process of running, committing, or aborting at the time of the crash. So, we need to do another "final pass" for the transactions in the transaction table. There are two rules to keep in mind for the final pass:

- ✓ If the transaction was in the process of committing, we add an END record to the log and remove it from the transactions table.
- ✓ If the transaction was running, we change its status in the transaction table to aborting and add an ABORT record to the log.
- ✓ Redo

Once we finish the analysis phase, we start the redo phase to get durability. We begin at the smallest recLSN in the dirty page table and redo following transactions.

Intuitively, in this phase, we want to redo all operations that did not make it to disk before the crash. The metadata in the transaction and dirty page tables will help us determine whether the operation made it to disk before the crash.

We redo all UPDATE and CLR operations that do not meet *any* of the following criteria:

- ✓ The page is not in the DPT — meaning the change must have been flushed to disk before the crash
- ✓ The corresponding pageLSN on disk >= current record's LSN — meaning at least the current operation (and possibly some future operation) must have made it to disk before the crash
- ✓ The corresponding recLSN in the dirty page table > current record's LSN — meaning the first operation that dirtied the page occurred after the current operation, so the current operation must have made it to disk before the crash
- ✓ Undo

Finally, we can do the undo phase to ensure atomicity. We start at the *end* of the log and work our way back to the start. Intuitively, we want to undo any UPDATE for any running or aborting transaction at the time of crash. We will only undo UPDATEs that do not have corresponding CLRs (meaning, we will not undo UPDATEs that have already been undone by a CLR).

We have one extra piece of metadata to track in this phase: the *undoNextLSN*. This field stores the LSN of the operation that we want to next undo for that transaction (derived from the prevLSN of the operation currently being undone). Once we undo all operations for a transaction, we can write an END record for that transaction.

Here is the pseudocode for the undo phase (taken from the Berkeley DB notes):

The intuition behind the pseudocode is as follows: we want to undo all the not-undone operations for all the transactions in the transaction table. So, we keep iterating through operations of the transactions, in reverse order (according to LSNs). If the record was a CLR, then we don't have anything to undo in the DB – we just traverse to the next LSN to undo (or write an END if there isn't one). If the record was an UPDATE, we undo it in the DB, log a CLR record, and traverse to the next LSN to undo (or write an END if there isn't one).

5.3.1 Optimization: Checkpointing

In production DBs, it can be terribly inefficient to reconstruct the transaction and dirty page tables from the absolute beginning of the log. So, ARIES uses checkpointing to speed up the analysis phase, which periodically writes the contents of the transaction and dirty page tables to the log, so we can read from the latest checkpoint instead of reconstructing from scratch.

With checkpointing, <BEGIN_CHECKPOINT> and <END_CHECKPOINT> records get written to the log, with the contents of the transaction and dirty page tables in between. When reconstructing, we need to start reading operations from the <BEGIN_CHECKPOINT> —

because the tables saved to the log can be the state at any point between the <BEGIN_CHECKPOINT> and <END_CHECKPOINT> records.

5.4 Recovery in Multi-database Systems

So far, we have implicitly assumed that a transaction accesses a single database. In some cases, a single transaction, called a multi-database transaction, may require access to multiple databases. 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 multi-database 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 multi-database 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 multi-database 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 a 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. Because all the local effects of the transaction and information needed for local recovery have been recorded in the logs of the participating databases, local recovery from failure is now possible. Each participating database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed. Conversely, if one or more of the participating databases or the coordinator have a not OK response, the transaction has failed, and the coordinator sends a message to roll back or UNDO the local effect of the transaction to each participating database. This is done by undoing the local transaction operations, using the log.

The net effect of the two-phase commit protocol is that either all participating data bases 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

CHAPTER SIX

6. Database Security and Authorization

6.1 Introduction

Think of a storage place for a manufacturing organization where it keeps all the important raw materials required for making its products. The organization uses these materials to run its production. Various types of materials form parts of different products. Some types of materials are more critical and sensitive to the production process. The organization cannot perform its day-to-day operations without access to the materials. Will the organization allow anyone to walk into the storage facility off the street and have access to the materials? Unless the place is properly secured, unauthorized persons can enter and steal or destroy the materials. Not even all authorized persons may be permitted access to all parts of the storage area. Sensitive and critical materials must be off-limits to most people

For a modern organization, its database system is even more significant than many other types of assets. Many organizations such as financial institutions and travel companies cannot survive even a single day without their database systems. Any type of destruction of or unauthorized access to the database system has serious impact. Obviously, an organization must ensure that its database system is adequately guarded against accidental breaches of security or theft, misuse, and destruction through malicious intent. Every organization must protect its database system from intentional and unintentional threats. To do so, it must employ both computer-based and other types of controls. The DBMS must include a proper security system to protect the database from unauthorized access.

Security Issues

What are we trying to protect by ensuring database security?

- ✓ What levels of information need to be safeguarded and how?
- ✓ What are the types of problems and threats that deserve special attention?
- ✓ Can we distinguish between threats from outside and internal threats?
- ✓ Do these require different types of protection mechanisms?
- ✓ What are the solution options?

- ✓ How is protection of privacy related to database security?

Let us address these broad questions before getting into specific access control techniques. Many organizations are opening up their database systems for access over the Internet. This openness results in great advantages but, at the same time, makes the database system vulnerable to threats from a much wider area. Web security demands special attention.

Security Problems

Many aspects of security problems require attention in a database environment. Legal, social, and ethical aspects are involved. Does the person requesting for particular information have a legal right to that piece of information? Also, there are policies questions about who decides on what types of access authorizations must be granted to whom and when. Operational and administrative aspects need to be considered How do you allocate passwords, maintain them, and preserve confidentiality?
What about physical controls to prevent problems? Should workstations and servers be guarded with physical lock-and-key schemes? Are hardware controls available in your environment to be used for database security? Are there security schemes in the operating system itself? Finally, what are the security provisions in your DBMS, and to what extent can your environment take advantage of these provisions? To come up with solution options, first it will be worthwhile to classify the types of security problems likely to be encountered.

When you are able to classify the threats, you will be able to find solutions to each type of problem. Broadly, we may classify the types of security exposure in a database environment as follows:

- ✓ **Natural disasters.** Fire, floods, and other such catastrophes.
- ✓ **Human carelessness.** Unintended damage caused by authorized users, especially while running jobs in batch.
- ✓ **Malicious damage**. Sabotage, vandalism, actions of malicious programmers, technical support staff, and persons performing database administration functions.
- ✓ **Crime.** Theft, embezzlement, industrial espionage, and employees selling a company‘s secrets and data for mailing lists.
- ✓ **Privacy invasion**. Casual curiosity, data lookup by competitors, obtaining data for political or legal reasons.

Let us put together the components of the problems of database protection and summarize the potential threats. In this figure presents a summary of threats to database security. Note each component showing the type of threat and its source.

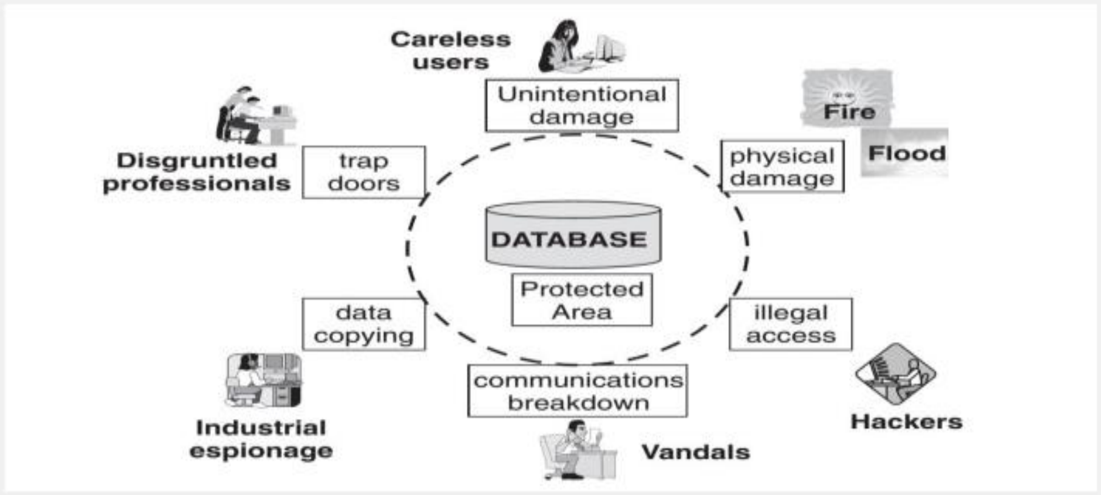


Figure 2. 2 Threats to database security.

6.2 Access Control

Essentially, database security rests on controlling access to the database system. Controlling physical access forms one part of database security. The other major part consists of controlling access through the DBMS. Let us consider two primary dimensions of access control. One dimension of access control deals with levels of data access. A single user or a category of users may be granted access privileges to database objects at various levels of detail.

Another dimension of access control refers to the modes or types of access granted to a single user or to a category of users. How do you grant access privileges to a single user or user category? This leads to the two basic approaches to access control.

As noted above, the DBMS provides two basic approaches to access control:

1. Discretionary control and
2. Mandatory control.

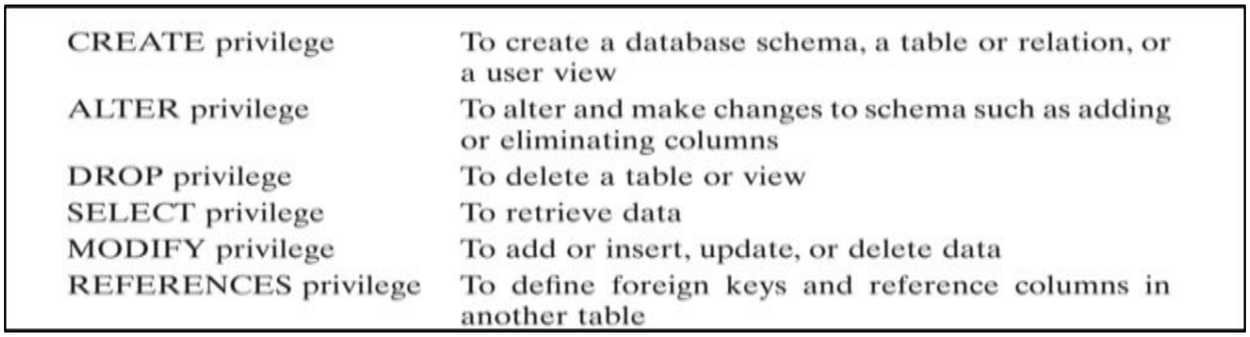
Discretionary access control refers to the granting of privileges or rights to individual users. Although discretionary access control is fairly effective, it is possible for an unauthorized user to gain privileges through an unsuspecting authorized user. Mandatory access control is more effective in overcoming the defects of discretionary access control.

6.2.1 Discretionary Control

As mentioned above, in this approach, individual users are granted privileges or rights to access specific data items in one or more designated modes. On the basis of the specification of privileges, a user is given the discretion to access a data item in the read, update, insert, or delete modes. A user who created a database object automatically derives all privileges to access the object including the passing on of privileges to other users with regard to that object. We introduced the

SQL commands for granting and revoking access privileges. This is how SQL supports discretionary access control. Now we will explore the fundamental concepts of discretionary access control and go over a few more examples.

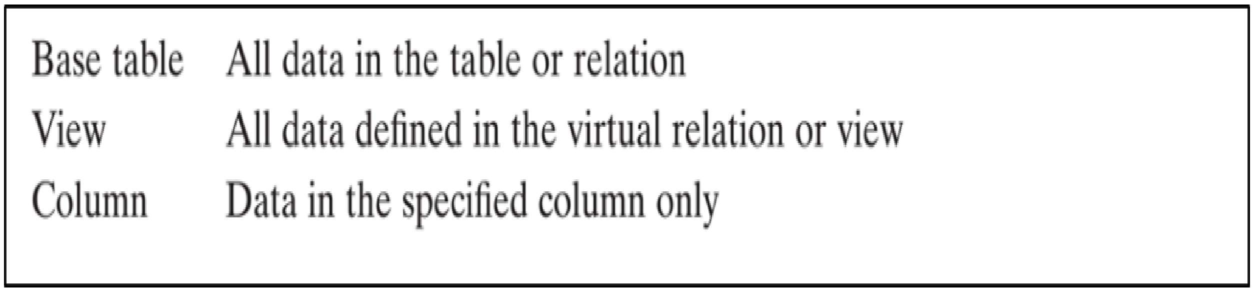
- ✓ Basic Levels There are two basic components or levels for granting or revoking access privileges:
- ✓ Database Objects Users Data item or data element, generally a base table or view A single user or a group of users identifiable with some authorization identifier With these two components, access privileges may be granted as shown in the following general command: GRANT privileges ON database object TO users



CREATE privilege	To create a database schema, a table or relation, or a user view
ALTER privilege	To alter and make changes to schema such as adding or eliminating columns
DROP privilege	To delete a table or view
SELECT privilege	To retrieve data
MODIFY privilege	To add or insert, update, or delete data
REFERENCES privilege	To define foreign keys and reference columns in another table

At the level of users, access privileges include the following:

At the level of database objects, the access privileges listed above apply to the following: Authorization Matrix We can think of the two levels of users and database objects forming a



Base table	All data in the table or relation
View	All data defined in the virtual relation or view
Column	Data in the specified column only

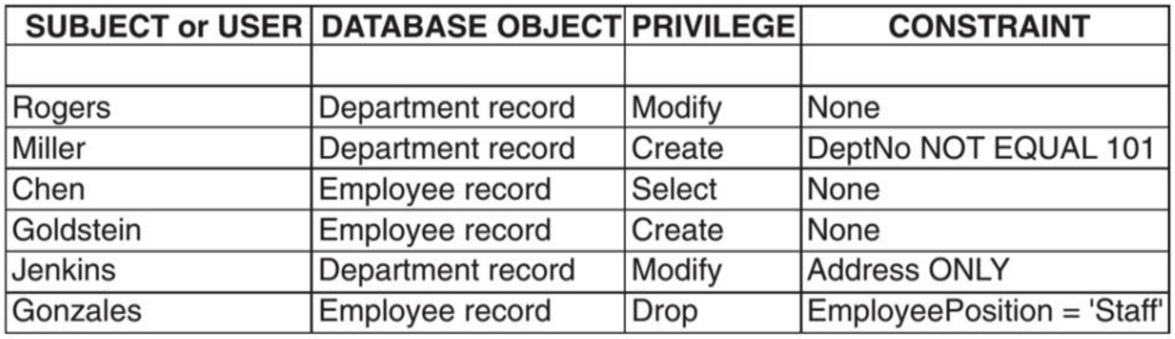
At the level of users, access privileges include the following:

At the level of database objects, the access privileges listed above apply to the following: Authorization Matrix We can think of the two levels of users and database objects forming a

matrix for the purpose of granting access privileges. Set the users as columns and the database objects as rows. Then in the cells formed by the intersection of these columns and rows we can specify the type of privilege granted. Table presents an example of a type of authorization matrix. Note how this type of presentation makes it easy to review the access privileges in a database environment.

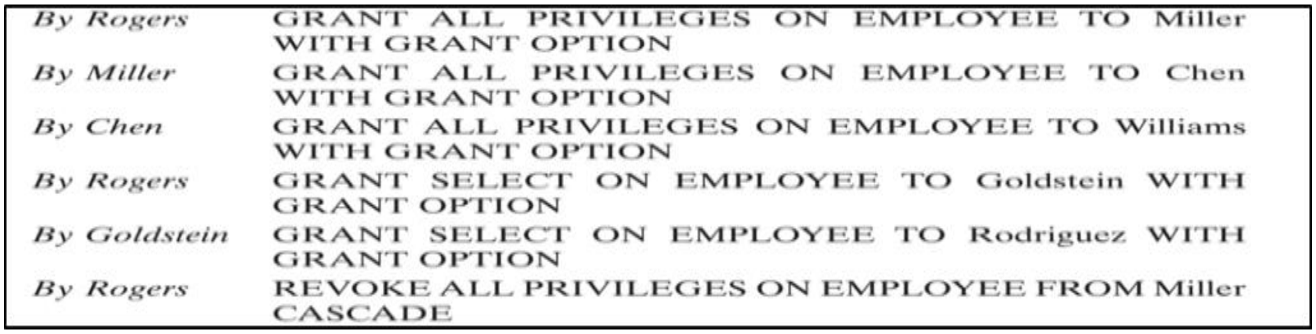
Owner Account. Each database table or relation has an owner. This user account that created the table possesses all access privileges on that table. The DBA can assign an owner to an entire schema and grant the appropriate access privileges.

The owner of a database object can grant privileges to another user. This second user can then pass along the privileges to a third user and so on. The DBMS keeps track of the cycle of granting of privileges.



SUBJECT or USER	DATABASE OBJECT	PRIVILEGE	CONSTRAINT
Rogers	Department record	Modify	None
Miller	Department record	Create	DeptNo NOT EQUAL 101
Chen	Employee record	Select	None
Goldstein	Employee record	Create	None
Jenkins	Department record	Modify	Address ONLY
Gonzales	Employee record	Drop	EmployeePosition = 'Staff'

Here is an example of a cycle of privileges passed along from Rogers, who is the owner of table EMPLOYEE



By Rogers	GRANT ALL PRIVILEGES ON EMPLOYEE TO Miller WITH GRANT OPTION
By Miller	GRANT ALL PRIVILEGES ON EMPLOYEE TO Chen WITH GRANT OPTION
By Chen	GRANT ALL PRIVILEGES ON EMPLOYEE TO Williams WITH GRANT OPTION
By Rogers	GRANT SELECT ON EMPLOYEE TO Goldstein WITH GRANT OPTION
By Goldstein	GRANT SELECT ON EMPLOYEE TO Rodriguez WITH GRANT OPTION
By Rogers	REVOKE ALL PRIVILEGES ON EMPLOYEE FROM Miller CASCADE

The illustrates this cycle of privileges with an authorization graph. Note how the privileges are passed along and how the revoking of privileges with cascade option works. REFERENCES Option the REFERENCES privilege is not the same as the SELECT privilege. Let us take an example. Suppose Nash is the owner of the DEPARTMENT table as indicated below:

DEPARTMENT (DeptNo, DeptName, DeptLocation)

Nash can authorize Miller to create another table EMPLOYEE with a foreign key in that table to refer to the DeptNo column in the DEPARTMENT table. Nash can do this by granting Miller the REFERENCES privilege with respect to the DeptNo column. Note the EMPLOYEE table shown below:

EMPLOYEE (<u>EmployeeNo</u>, FirstName, LastName, Address, Phone, Employee Position, Salary, EmployeeCode, DeptNo)

Foreign Key: DeptNo REFERENCES DEPARTMENT

If Miller loses the REFERENCES privilege with respect to the DeptNo column in the DEPARTMENT table, the foreign key constraint in the EMPLOYEE

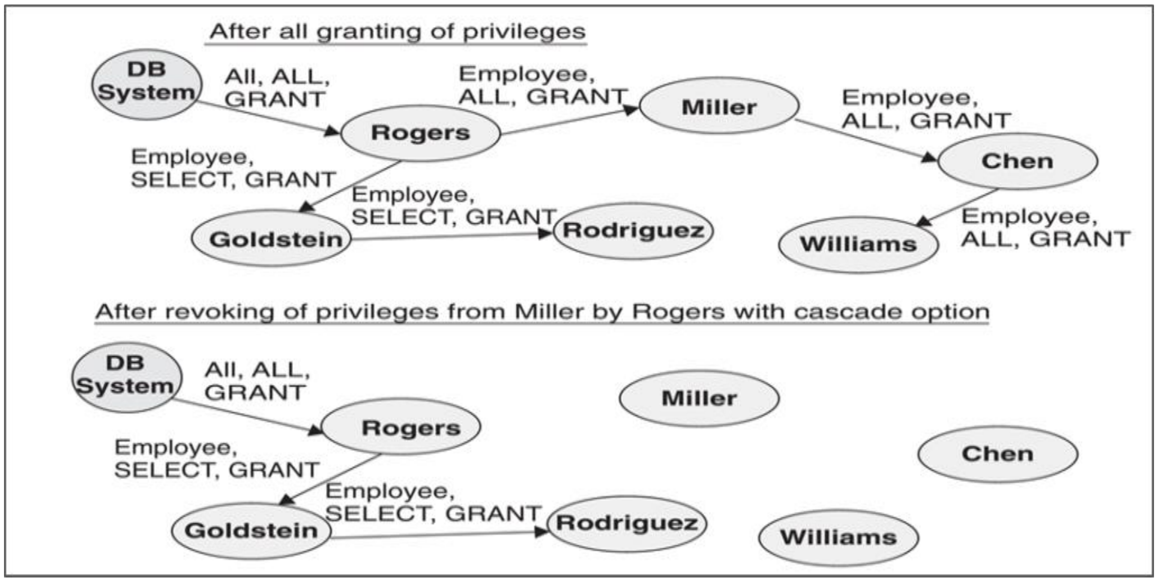


Table will be dropped. The EMPLOYEE table itself, however, will not be dropped. Now suppose Miller has the SELECT privilege on the DeptNo column of the DEPARTMENT table, not the REFERENCES privilege. In this case, Miller will not be allowed to create the EMPLOYEE table with a foreign key column referring to DeptNo in the DEPARTMENT table.

Why not grant Miller the SELECT privilege and allow him to create the EMPLOYEE table with a foreign key column referring to the DeptNo column in the DEPARTMENT table? If this is done, assume that Miller creates the table with a foreign key constraint as follows:

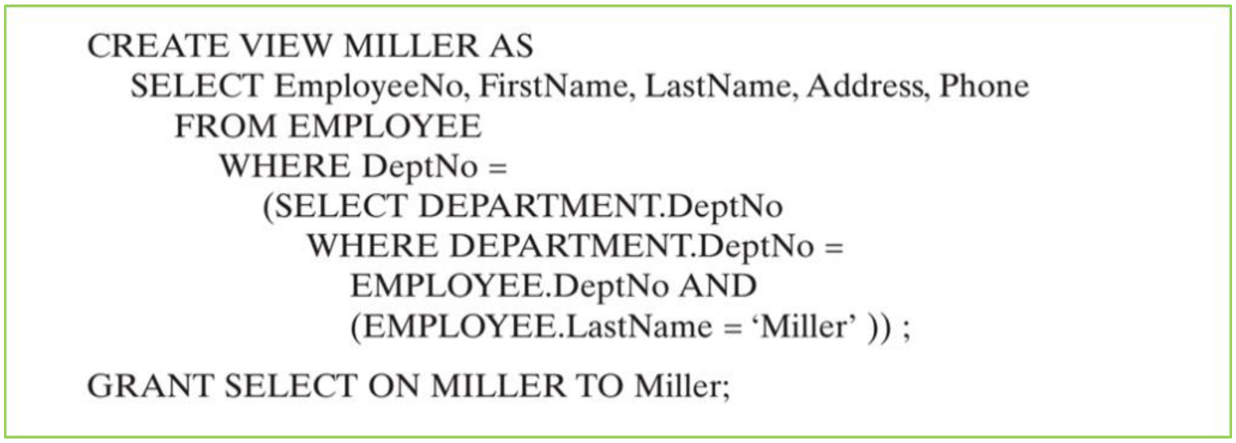
EMPLOYEE (EmployeeNo, FirstName, LastName, Address, Phone, Employee Position, Salary, EmployeeCode, DeptNo)

Foreign Key: DeptNo REFERENCES DEPARTMENT ON DELETE NO ACTION

With the NO ACTION option in the foreign key specification, Nash is prevented from deleting rows from the DEPARTMENT table even though he is the owner. For this reason, whenever such a restrictive privilege needs to be authorized, the more stringent privilege REFERENCES is applied. The SELECT privilege is therefore intended as permission just to read the values.

6.2.1.1 Use of Views

Earlier we had discussions on user views. A user view is like a personalized model of the database tailored for individual groups of users. If a user group, say, in the marketing department, needs to access only some columns of the DEPARTMENT and EMPLOYEE



```
CREATE VIEW MILLER AS
   SELECT EmployeeNo, FirstName, LastName, Address, Phone
      FROM EMPLOYEE
         WHERE DeptNo =
            (SELECT DEPARTMENT.DeptNo
               WHERE DEPARTMENT.DeptNo =
                  EMPLOYEE.DeptNo AND
                  (EMPLOYEE.LastName = 'Miller' )) ;

GRANT SELECT ON MILLER TO Miller;
```

Tables, then you can satisfy their information requirements by creating a view comprising just those columns. This view hides the unnecessary parts of the database from the marketing group and shows them only those columns hey require.

Views are not like tables in the sense that they do not store actual data. You know that views are just like windows into the database tables that store the data. Views are virtual tables. When a user accesses data through a view, he or she is getting the data from the base tables, but only from the columns defined in the view.

Views are intended to present to the user exactly what is needed from the database and to make the rest of the data content transparent to the user. However, views offer a flexible and simple method for granting access privileges in a personalized manner. Views are powerful security tools.

When you grant access privileges to a user for a specific view, the privileges apply only to those data items defined in the views and not to the complete base tables themselves.

Let us review an example of a view and see how it may be used to grant access privileges. For a user to create a view from multiple tables, the user must have access privileges on those base tables.

The view is dropped automatically if the access privileges are dropped. Note the following example granting access privilege to Miller for reading EmployeeNo, FirstName, LastName, Address, and Phone information of employees in the department where Miller works.

SQL Examples

In this we considered a few SQL examples on granting and revoking of access privileges. Now we will study a few more examples. These examples are intended to reinforce your understanding of discretionary access control. We will use the DEPARTMENT and EMPLOYEE tables shown above for our SQL examples.

DBA gives privileges to Miller to create the schema:
GRANT CREATETAB TO Miller;
Miller defines schema, beginning with create schema statement:
CREATE SCHEMA EmployeeDB AUTHORIZATION Miller; (other DDL
Statements follow to define DEPARTMENT and EMPLOYEE tables)
Miller gives privileges to Rodriguez for inserting data in both tables: GRANT INSERT ON DEPARTMENT, EMPLOYEE TO Rodriguez;
Miller gives Goldstein privileges for inserting and deleting rows in both tables, allowing permission to propagate these privileges:
GRANT INSERT, DELETE ON DEPARTMENT, EMPLOYEE TO Goldstein WITH GRANT OPTION;
Goldstein passes on the privileges for inserting and deleting rows in the DEPARTMENT table to Rogers:
GRANT INSERT, DELETE ON DEPARTMENT TO Rogers;
Miller gives Williams the privilege to update only the salary and position columns in the EMPLOYEE table:
GRANT UPDATE ON EMPLOYEE (Salary, EmployeePosition) TO Williams
DBA gives Shady privilege to create tables:
GRANT CREATETAB TO Shady;
Shady creates table MYTABLE and gives Miller privilege to insert rows into MYTABLE:
GRANT INSERT ON MYTABLE TO Miller;

6.2.2 Mandatory Control

Discretionary access control provides fairly adequate protection. This has been the traditional approach in relational databases. A user either has certain access privileges or he or she does not. The discretionary access control method does not support variations based on the sensitivity of parts of the database. Although the method is sufficient in most database environments, it is not bulletproof.

An ingenious professional can drill holes into the protection mechanism and gain unauthorized access. Note the actions of user Shady indicated in the last few statements of the previous subsection. Shady has created a private table MYTABLE of which he is the owner.

He has all privileges on this table. All he has to do is somehow get sensitive data into MYTABLE. Being a clever professional, Shady may temporarily alter one of Miller's programs to take data from the EMPLOYEE data and move the data into MYTABLE. For this purpose, Shady has already given privileges to Miller for inserting rows into the MYTABLE table. This scenario appears as too unlikely and contrived. Nevertheless, it makes the statement that discretionary access control has its limitations.

Mandatory access control overcomes the shortcomings of discretionary access control. In the mandatory access control approach, access privileges cannot be granted or passed on by one user to another in an uncontrolled manner. A well-defined security policy dictates which classes of data may be accessed by users at which clearance levels. The most popular method is known as the **Bell–LaPadula model**. Many of the commercial relational DBMSs do not currently provide for mandatory access control. However, government agencies, defense departments, financial institutions, and intelligence agencies do require security mechanisms based on the mandatory control technique.

6.2.3 Special Security Considerations

We have covered several topics on database security so far. You know the common types of security threats. You have explored solution options. You have reviewed the two major approaches to database access control discretionary and mandatory. Before we finish our discussion of the significant topics, we need to consider just a few more.

In our discussion on granting access privileges, we have been referring to individual users or user groups that need access privileges. Who are these users, and how do you identify them to the database system? This is an important question we need to address. Another obvious question is, where is the DBA in all of these database security provisions, and what is the role of the DBA?

Finally, we will inspect what are known as statistical databases and consider special security problems associated with these.

6.2.3.1 Authorization

The security mechanism protecting a database system is expected to prevent users from performing database operations unless they are authorized to do so. Authorization for data access implies access control. We have discussed discretionary and mandatory access control approaches.

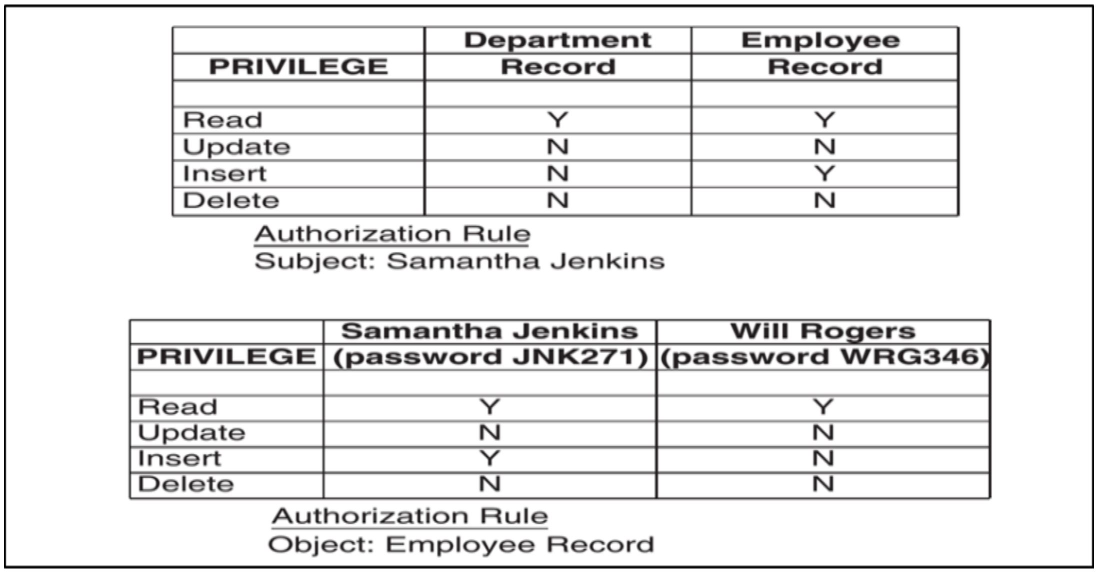
Let us now complete the discussion by touching on a few remaining topics.

- ✓ A Profile To authorize a subject that may be a user, a group of users, a program, or a module, an account is assigned to the subject. Let us confine our discussion to a subject who is a user. User Samantha Jenkins is eligible to have access to the human resources database. So first, Jenkins must be assigned an account or user identification.

  The DBMS maintains a user profile for each user account. The profile for Jenkins includes all the database objects such as tables, views, rows, and columns that she is authorized to access. In the user profile, you will also find the types of access privileges such as read, update, insert, and delete granted to Jenkins.

  Alternatively, the DBMS may maintain an object profile for each database object. An object profile is another way of keeping track of the authorizations. For example, in the object profile for the EMPLOYEE table, you will find all the user accounts that are authorized to access the table. Just like a user profile, an object profile also indicates the types of access privileges.

- ✓ **Authorization Rules** The user profile or the object profile stipulates which user can access which database object and in what way. These are the authorization rules. By examining these rules, the DBMS determines whether a specific user may be permitted to perform the operations of read, update, insert, or delete on a particular database object. You have already looked at an example of an authorization matrix in table 2. This matrix tends to be exhaustive and complex in a large database environment.



	Department	Employee
PRIVILEGE	Record	Record
Read	Y	Y
Update	N	N
Insert	N	Y
Delete	N	N

Authorization Rule
Subject: Samantha Jenkins

	Samantha Jenkins	Will Rogers
PRIVILEGE	(password JNK271)	(password WRG346)
Read	Y	Y
Update	N	N
Insert	Y	N
Delete	N	N

Authorization Rule
Object: Employee Record

6.2.3.2 Authentication

Let us return to the authorization of access privileges to Samantha Jenkins. The authorization matrix contains security authorization rules for her. When she attempts to perform any database operation, the DBMS, through its arbiter module, can verify authorization rules as applicable and either allow or deny the operation. When Samantha Jenkins signs on to the database system with her user-id, in effect, she declares that she is Samantha Jenkins.

All authorization she can have relates to the user known to the system as Samantha Jenkins. Now when she signs on with her user-id and declares that she is Samantha Jenkins, how does the system know that she is really who she says she is? How can the system be sure that it is really Samantha Jenkins and not someone else signing on with her user-id? How can the system authenticate her identity?

Authentication is the determination of whether the user is who he or she claims to be or declares he or she is through the user-id. It is crucial that the authentication mechanism be effective and failsafe. Otherwise, all the effort and sophistication of the authorization rules will be an utter waste. How can you ensure proper authentication? Let us examine a few of the common techniques for authentication.

- ✓ **Passwords.** Passwords, still the most common method, can be effective if properly administered. Passwords must be changed fairly often to deter password thefts. They must be stored in encrypted formats and be masked while being entered. Password formats need to be standardized to avoid easily detectable combinations. A database environment with highly sensitive data may require one-time-use passwords.
- ✓ **Personal information.** The user may be prompted with questions for which the user alone would know the answers such as mother's maiden name, last four digits of social security number, first three letters of the place of birth, and so on.
- ✓ **Biometric verification.** Verification through fingerprints, voiceprints, and retina images, and so on. Smartcards recorded with such biometric data may be used.
- ✓ **Special procedures.** Run a special authentication program and converse with the user. System sends a random number m to the user. The user performs a simple set of operations on the random number and types in the result n. System verify n by performing the same algorithm on m. Of course, m and n will be different each time and it will be hard for a perpetrator to guess the algorithm
- ✓ **Hang-up and call-back**. After input of user-id, the system terminates the input and reinitiates input at the workstation normally associated with that user. If the user is there at that customary workstation and answers stored questions for the user-id, then the system allows the user to continue with the transaction.

6.3 Statistical Databases

Statistical databases pose a great and interesting challenge in the matter of data security. Statistical databases are usually large databases intended to provide statistical information and not information about individual entities. Statistical databases may contain data about large populations of entities.

A census database contains information about the people in specific geographic areas. The database system of a large international bank holds information about the savings and checking account activities of significant strata of the population. Databases of large financial institutions contain profiles of investors.

Databases used in data warehousing and data mining may be considered as statistical databases in some significant sense. Need for Data Access Statistical databases serve critical purposes. They store rich data content providing population statistics by age groups, income levels, household sizes, education levels, and so on. Government statisticians, market research companies, and institutions estimating economic indicators depend on statistical databases. These professionals select records from statistical databases to perform statistical and mathematical functions.

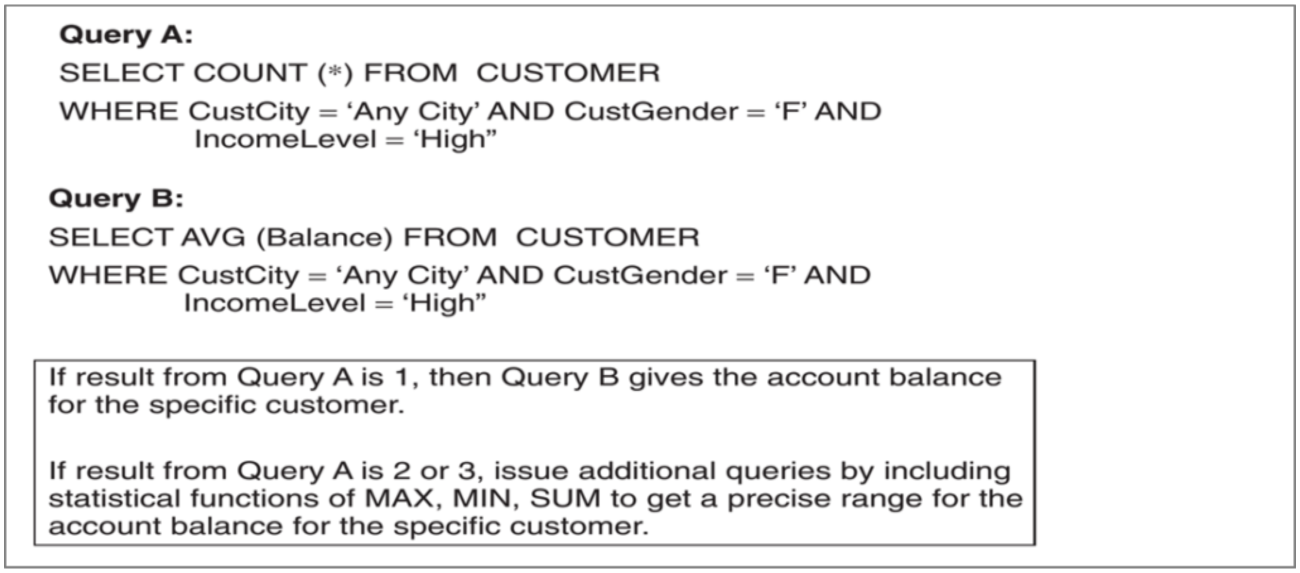
They may count the number of entities in the selected sample of records from a statistical database, add up numbers, take averages, find maximum and minimum amounts, and calculate statistical variances and standard deviations. All such professionals need access to statistical databases.

However, there is one big difference between users of an operational database needing access privileges and professionals requiring access privileges to a statistical database.

Users of an operational database need information to run the day-to-day business to enter an order, to check stock of a single product, to send a single invoice. That is, these users need access privileges to individual records in the database. On the other hand, professionals using statistical databases need access privileges to access groups of records and perform mathematical and statistical calculations from the selected groups.

They are not interested in single records, only in samples containing groups of records Security Challenge So what is the problem with granting access privileges to professionals to use a statistical database just the way you would grant privileges to use any other type of database? Here is the problem: The professionals must be able to read individual records in a select sample group for performing statistical calculations but, at the same time, must not be allowed to find out what is in a particular record.

For example, take the case of the international bank. The bank's statisticians need access to the bank's database to perform statistical calculations. For this purpose, you need to grant them access privileges to read individual records. But, at the same time, you cannot allow them to see Jane Doe's bank account balance. The challenge in the case of the bank is this: How can you grant access privileges to the statisticians without compromising the confidentiality of individual bank customers?
Perhaps one possible method is to grant access privileges to individual records because the statistician needs to read a group of records for the calculations but restrict the queries to perform only mathematical and statistical functions such as COUNT, SUM, AVG, MAX, MIN, variance and standard deviations.



Query A:

```
SELECT COUNT (*) FROM  CUSTOMER
WHERE CustCity = 'Any City' AND CustGender = 'F' AND
        IncomeLevel = 'High"
```

Query B:

```
SELECT AVG (Balance) FROM  CUSTOMER
WHERE CustCity = 'Any City' AND CustGender = 'F' AND
        IncomeLevel = 'High"
```

If result from Query A is 1, then Query B gives the account balance for the specific customer.

If result from Query A is 2 or 3, issue additional queries by including statistical functions of MAX, MIN, SUM to get a precise range for the account balance for the specific customer.

Although this method appears to be adequate to preserve the confidentiality of individual customers, a clever professional can run a series of queries and narrow the intersection of the query result to one customer record.

This person can infer the values in individual rows by running a series of ingenuous queries. Each query produces a result set. Even though only statistical functions are permitted, by combining the different results through a series of clever queries, information about a single entity may be determined.

Figure above illustrates how, by using different predicates in queries from a bank's statistical database, the bank balance of a single Customer Jane Doe may be determined. Assume that the infiltrator knows some basic information about Jane Doe.

- ✓ Solution Options Safeguarding privacy and confidentiality in a statistical database proves to be difficult. The standard method of granting access privileges does not work. In addition to discretionary and mandatory techniques, other restrictions must be enforced on queries. Here is a list of some solution options. None of them is completely satisfactory. Combinations of some of the options seem to be effective. Nevertheless, protection of privacy and confidentiality in statistical databases is becoming more and more essential.
- ✓ **Only statistical functions.** Allow only statistical functions in queries.
- ✓ **Same sample.** Reject series of queries to the same sample set of records.
- ✓ **Query types.** Allow only those queries that contain statistical or mathematical functions.
- ✓ **Number of queries.** Allow only a certain number of queries per user per unit time.
- ✓ **Query thresholds.** Reject queries that produce result sets containing fewer than n records, where n is the query threshold.
- ✓ **Query combinations.** The result set of two queries may have a number of common records referred to as the intersection of the two queries. Impose a restriction saying that no two queries may have an intersection larger than a certain threshold number.
- ✓ **Data pollution.** Adopt data swapping. In the case of the bank database, swap balances between accounts. Even if a user manages to read a single customer's record, the balances may have been swapped with balances in another customer's record.
- ✓ **Introduce noise.** Deliberately introduce slight noise or inaccuracies. Randomly add records to the result set. This is likely to show erroneous individual records, but statistical samples produce approximate responses quite adequate for statistical analysis.
- ✓ **Log queries.** Maintain a log of all queries. Maintain a history of query results and reject queries that use a high number of records identical to those used in previous queries.

CHAPTER SEVEN

7. Distributed Database System

7.1 Distributed Database Concepts

We can define a distributed database (DDB)as a collection of multiple logically interrelated databases distributed over a computer network, and a distributed database management system (DDBMS)as a software system that manages a distributed database while making the distribution transparent to the user.

Distributed databases are different from Internet Web files. Web pages are basically a very large collection of files stored on different nodes in a network the Internet with interrelationships among the files represented via hyperlinks. The common functions of database management, including uniform query processing and transaction processing, do not apply to this scenario yet.

Differences between DDB and Multiprocessor Systems

We need to distinguish distributed databases from multiprocessor systems that use shared storage (primary memory or disk). For a database to be called distributed, the following minimum conditions should be satisfied:

- ✓ Connection of database nodes over a computer network. There is multiple computers, called sites or nodes. These sites must be connected by an underlying communication network to transmit data and commands among sites
- ✓ Logical interrelation of the connected databases. It is essential that the information in the databases be logically related.
- ✓ Absence of homogeneity constraint among connected nodes. It is not necessary that all nodes be identical in terms of data, hardware, and software.

The sites may all be located in physical proximity say, within the same building or a 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 wireless hubs or cables, whereas long-haul networks use telephone lines or satellites. It is also possible to use a combination of networks.

Networks may have different topologies that define the direct communication paths among sites. The type and topology of the network used may have a significant impact on the performance and hence on the strategies for distributed query processing and distributed database design. For high level architectural issues, however, it does not matter what type of network is used; what matters is that each site be able to communicate, directly or indirectly, with every other site.

For the remainder of this chapter, we assume that some type of communication network exists among sites, regardless of any particular topology. We will not address any network specific issues, although it is important to understand that for an efficient operation of a distributed database system (DDBS), network design and performance issues are critical and are an integral part of the overall solution. The details of the underlying communication network are invisible to the end user.

7.1.1 Transparency

The concept of transparency extends the general idea of hiding implementation details from end users. A highly transparent system offers a lot of flexibility to the end user/application developer since it requires little or no awareness of underlying details on their part. In the case of a traditional centralized database, transparency simply pertains to logical and physical data independence for application developers.

- ✓ However, in a DDB scenario, the data and software are distributed over multiple sites connected by a computer network, so additional types of transparencies are introduced. Consider the company database in Figure 3.5 that we have been discussing throughout the book. The EMPLOYEE, PROJECT, and WORKS_ON tables may be fragmented horizontally and stored with possible replication as shown in Figure below.

The following types of transparencies are possible:

7.1.1.1 Data organization transparency (also known as distribution or network transparency).

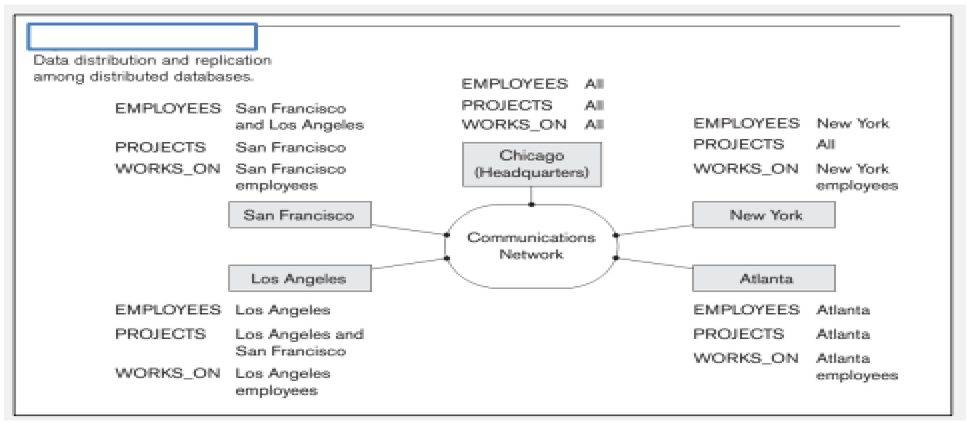
This refers to freedom for the user from the operational details of the network and the placement of the data in the distributed system. It may be divided into location transparency and naming transparency.

- ✓ Location transparency refers to the fact that the command used to perform a task is independent of the location of the data and the location of the node where the command was issued. Naming transparency implies that once a name is associated with an object, the named objects can be accessed unambiguously without additional specification as to where the data is located.

- ✓ Replication transparency. As we show in Figure below, copies of the same data objects may be stored at multiple sites for better availability, performance, and reliability. Replication transparency makes the user unaware of the existence of these copies.

7.1.1.2 Fragmentation transparency

Two types of fragmentation are possible. Horizontal fragmentation distributes a relation (table) into sub relations


Data distribution and replication among distributed databases.
EMPLOYEES All
PROJECTS All
WORKS_ON All
Chicago (Headquarters)
EMPLOYEES San Francisco and Los Angeles
PROJECTS San Francisco
WORKS_ON San Francisco employees
San Francisco
EMPLOYEES New York
PROJECTS All
WORKS_ON New York employees
New York
Communications Network
Los Angeles
Atlanta
EMPLOYEES Los Angeles
PROJECTS Los Angeles and San Francisco
WORKS_ON Los Angeles employees
EMPLOYEES Atlanta
PROJECTS Atlanta
WORKS_ON Atlanta employees

That is subsets of the tuples (rows) in the original relation. Vertical fragmentation distributes a relation into subrelations where each subrelation 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. Other transparencies include design transparency and execution transparency—referring to freedom from knowing how the distributed database is designed and where a transaction executes.

Autonomy

Autonomy determines the extent to which individual nodes or DBs in a connected DDB can operate independently. A high degree of autonomy is desirable for increased flexibility and customized maintenance of an individual node. Autonomy can be applied to design, communication, and execution. Design autonomy refers to independence of data model usage and transaction management techniques among nodes.

Communication autonomy determines the extent to which each node can decide on sharing of information with other nodes. Execution autonomy refers to independence of users to act as they please.

Reliability and Availability

Reliability and availability are two of the most common potential advantages cited for distributed databases. Reliability is broadly defined as the probability that a system is running (not down) at a certain time point, whereas availability is the probability that the system is continuously available during a time interval. We can directly relate reliability and availability of the database to the faults, errors, and failures associated with it.

A failure can be described as a deviation of a system's behavior from that which is specified in order to ensure correct execution of operations. Errors constitute that subset of system states that causes the failure. Fault is the cause of an error. To construct a system that is reliable, we can adopt several approaches. One common approach stress's fault tolerance; it recognizes that faults will occur, and designs mechanisms that can detect and remove faults before they can result in a system failure.

Another more stringent approach attempts to ensure that the final system does not contain any faults. This is done through an exhaustive design process followed by extensive quality control and testing.

A reliable DDBMS tolerates failures of underlying components and processes user requests so long as database consistency is not violated. A DDBMS recovery manager has to deal with failures arising from transactions, hardware, and communication networks. Hardware failures can either be those that result in loss of main memory contents or loss of secondary storage contents. Communication failures occur due to errors associated with messages and line failures. Message errors can include their loss, corruption, or out-of-order arrival at destination.

7.1.2 Advantages of Distributed Databases

Organizations resort to distributed database management for various reasons. Some important advantages are listed below.

- ✓ **Improved ease and flexibility of application development.** Developing and maintaining applications at geographically distributed sites of an organization is facilitated owing to transparency of data distribution and control.
- ✓ **Increased reliability and availability.** This is achieved by the isolation of faults to their site of origin without affecting the other databases connected to the network. When the data and DDBMS 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. If the data in the failed site had been replicated at another site prior to the failure, then the user will not be affected at all.
- ✓ **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.
- ✓ I**n 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 subqueries that execute in parallel. This contributes to improved performance.
- ✓ **Easier expansion.** In a distributed environment, expansion of the system in terms of adding more data, increasing database sizes, or adding more processors is much easier

7.1.3 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 distribution. 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 execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain the 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 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.

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

In a DDB, decisions must be made regarding which site should be used to store which portions of the database. For now, we will assume that there is no replication; that is, each relation or portion of a relation is stored at one site only.

We discuss replication and its effects later in this section. We also use the terminology of relational databases, but similar concepts apply to other data models. We assume that we are starting with a relational database schema and must decide on how to distribute the relations over the various sites. To illustrate our discussion, we use the relational database schema in table as follow

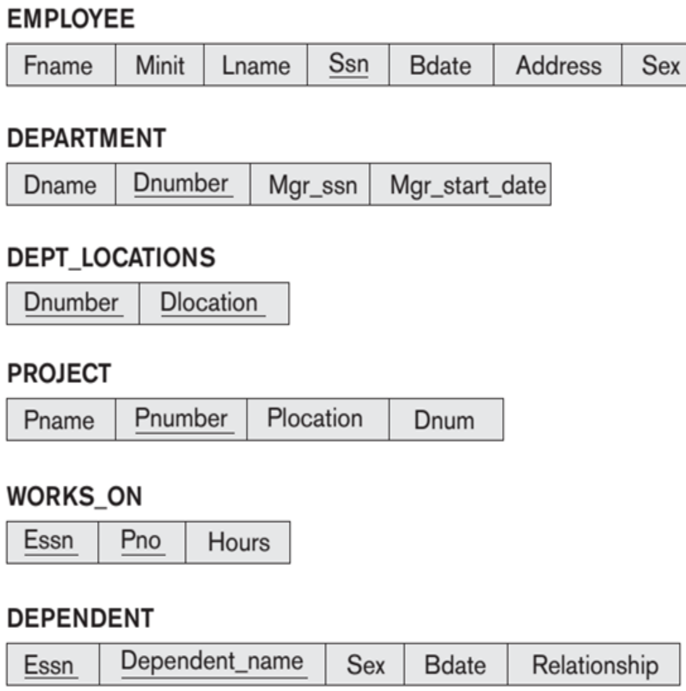

EMPLOYEE
Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno
DEPARTMENT
Dname | Dnumber | Mgr_ssn | Mgr_start_date
DEPT_LOCATIONS
Dnumber | Dlocation
PROJECT
Pname | Pnumber | Plocation | Dnum
WORKS_ON
Essn | Pno | Hours
DEPENDENT
Essn | Dependent_name | Sex | Bdate | Relationship

Figure 3.5
Schema diagram for the COMPANY relational database schema.

Before we decide on how to distribute the data, we must determine the logical units of the database that are to be distributed. The simplest logical units are the relations themselves; that is, each whole relation is to be stored at a particular site. In our example, we must decide on a site to store each of the relations EMPLOYEE, DEPARTMENT, PROJECT, WORKS_ON, and DEPENDENT.

In many cases, however, a relation can be divided into smaller logical units for distribution. For example, consider the company database shown in Figure, and assume there are three computer sites one for each department in the company. We may want to store the database information relating to each department at the computer site for that department. A technique called horizontal fragmentation can be used to partition each relation by department.

7.2.1 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 in Figure 3.6 with the following conditions: (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.

7.2.2 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 include work-related information Ssn, Salary, Super_ssn, and 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. Notice that each horizontal fragment on a relation R can be specified in the relational algebra by a σCi (R) operation.

A set of horizontal fragments whose conditions C 1, C2, …,Cn include all the tuples in R—that is, every tuple in R satisfies (C1 ORC2 OR…ORCn) is called a complete horizontal fragmentation of R. In many cases a complete horizontal fragmentation is also disjoint; that is, no tuple in R satisfies (Ci ANDCj) for any i≠j. Our two earlier examples of horizontal fragmentation for the EMPLOYEE and PROJECT relations were both complete and disjoint. To reconstruct the relation R from a complete horizontal fragmentation, we need to apply the UNION operation to the fragments.

A vertical fragment on a relation R can be specified by a πLi (R) operation in the relational algebra. A set of vertical fragments whose projection lists L1 ,L2, …,Ln include all the attributes in R but share only the primary key attribute of R is called a complete vertical fragmentation of R. In this case the projection lists satisfy the following two conditions:

- ✓ L1∪L2∪…∪Ln=ATTRS(R).
- ✓ Li ∩Lj=PK(R) for any i≠j,where ATTRS(R) is the set of attributes of R and PK(R) is the primary key ofR.

To reconstruct the relation R from a complete vertical fragmentation, we apply the OUTER UNION operation to the vertical fragments (assuming no horizontal fragmentation is used). Notice that we could also apply a FULL OUTER JOIN operation and get the same result for a complete vertical fragmentation, even when some horizontal fragmentation may also have been applied.

The two vertical fragments of the EMPLOYEE relation with projection lists L1 = {Ssn, Name, Bdate, Address, Sex} and L2= {Ssn, Salary, Super_ssn, Dno} constitute a complete vertical fragmentation of EMPLOYEE. Two horizontal fragments that are neither complete nor disjoint are those defined on the EMPLOYEE relation by the conditions (Salary> 50000) and (Dno= 4); they may not include all EMPLOYEE tuples, and they may include common tuples. Two vertical fragments that are not complete are those defined by the attribute lists L1 = {Name, Address} and L2 = {Ssn, Name, Salary}; these lists violate both conditions of a complete vertical fragmentation.

7.2.3 Data Replication and Allocation

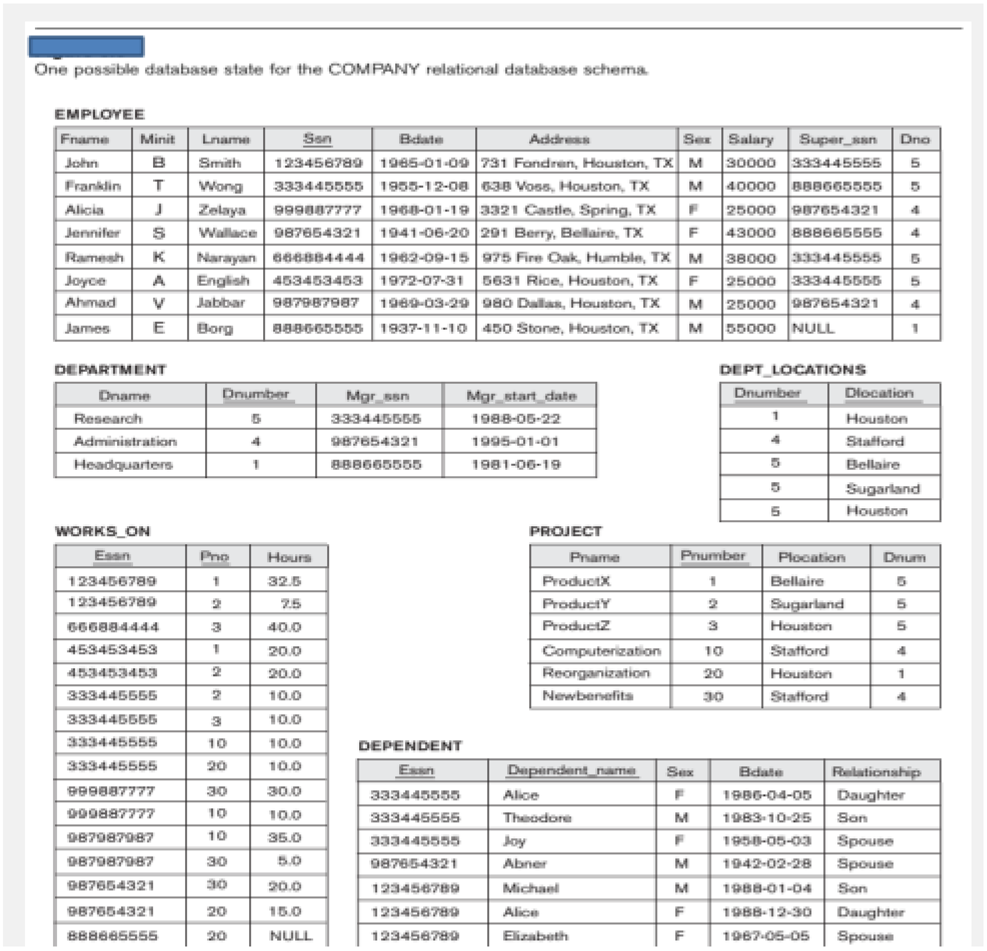
Replication is useful in improving the availability of data. The most extreme case is replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database.
This can improve availability remarkably because the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries because the results of such queries 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 was 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 PDAs 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, transactions can be submitted at any site, and 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.



[redacted]

One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Example of Fragmentation, Allocation, and Replication

Suppose that the company has three computer sites one for each current department. Sites 2 and 3 are for departments 5 and 4, respectively. At each of these sites, we expect frequent access to the EMPLOYEE and PROJECT information for the employees who work in that department and the projects controlled by that department.

Further, we assume that these sites mainly access theName,Ssn,Salary, and Super_ssn attributes of EMPLOYEE. Site 1 is used by company headquarters and accesses all employee and project information regularly, in addition to keeping track of DEPENDENT information for insurance purposes According to these requirements, the whole database in Figure can be stored at site 1.

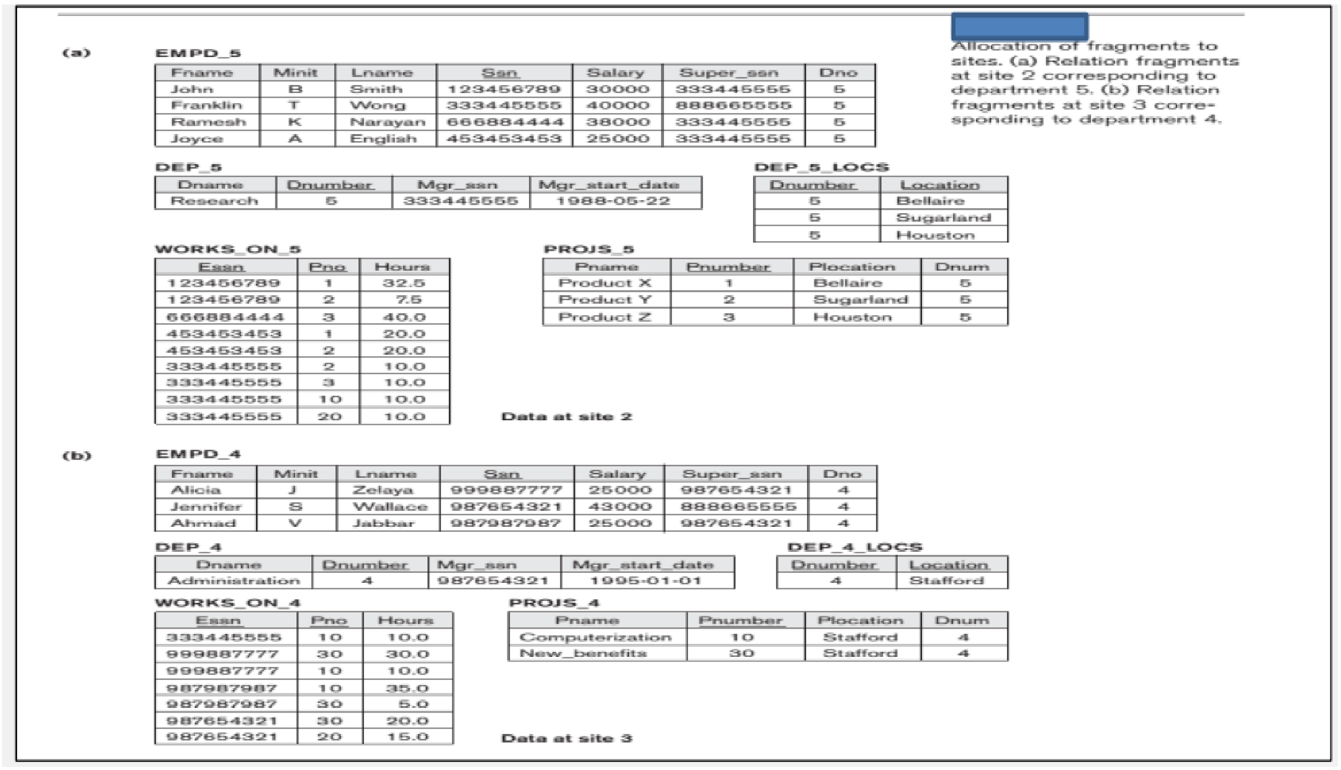
To determine the fragments to be replicated at sites 2 and 3, first we can horizontally fragment DEPARTMENTby its key Dnumber. Then we apply derived fragmentation to the EMPLOYEE, PROJECT, and DEPT_LOCATIONS relations based on their foreign keys for department number—called Dno,Dnum, and Dnumber, respectively, in Figure We can vertically fragment the resulting EMPLOYEE fragments to include only the attributes {Name,Ssn,Salary,Super_ssn,Dno}.

Figure shows the mixed fragments EMPD_5andEMPD_4, which include the EMPLOYEE tuples satisfying the conditions Dno= 5 and Dno= 4, respectively. The horizontal fragments of PROJECT, DEPARTMENT, and DEPT_LOCATIONS are similarly fragmented by department number. All these fragments—stored at sites 2 and 3—are replicated because they are also stored at headquarters site 1.

We must now fragment the WORKS_ON relation and decide which fragments of WORKS_ON to store at sites 2 and 3. We are confronted with the problem that no attribute of WORKS_ONdirectly indicates the department to which each tuple belongs. In fact, each tuple in WORKS_ONrelates an employee to a project P.

We could fragment WORKS_ON based on the department D in which works or based on the department Dthat controls P. Fragmentation becomes easy if we have a constraint stating that D=Dfor all WORKS_ON tuples—that is, if employees can work only on projects controlled by the department they work for. However, there is no such constraint in our database in Figure 3.6. For example, the WORKS_ONtuple <333445555, 10, 10.0> relates an employee who works for department 5 with a project controlled by department 4. In this case, we could fragment WORKS_ON based on the department in which the employee works (which is expressed by the condition C) and then fragment further based on the department that controls the projects that employee is working on.

In Figure, the union of fragments G1, G2, and G3 gives all WORKS_ONtuples for employees who work for department 5. Similarly, the union of fragments G4,G5, and G6 gives all WORKS_ON tuples for employees who work for department 4. On the other hand, the union of fragments G1,G4 , and G7 gives all WORKS_ON tuples for projects controlled by department 5. The condition for each of the fragments G1 through G9 is shown in Figure below The relations that represent M:N relationships, such as WORKS_ON, often have several possible logical fragmentations. In our distribution in Figure below we choose to include all fragments that can be joined to



Allocation of fragments to sites. (a) Relation fragments at site 2 corresponding to department 5. (b) Relation fragments at site 3 corresponding to department 4.

(a)

EMPD_5

Fname	Minit	Lname	Ssn	Salary	Super_ssn	Dno
John	B	Smith	123456789	30000	333445555	5
Franklin	T	Wong	333445555	40000	888665555	5
Ramesh	K	Narayan	666884444	38000	333445555	5
Joyce	A	English	453453453	25000	333445555	5

DEP_5

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22

DEP_5_LOCS

Dnumber	Location
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON_5

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0

PROJS_5

Pname	Pnumber	Plocation	Dnum
Product X	1	Bellaire	5
Product Y	2	Sugarland	5
Product Z	3	Houston	5

Data at site 2

(b)

EMPD_4

Fname	Minit	Lname	Ssn	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	25000	987654321	4
Jennifer	S	Wallace	987654321	43000	888665555	4
Ahmad	V	Jabbar	987987987	25000	987654321	4

DEP_4

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Administration	4	987654321	1995-01-01

DEP_4_LOCS

Dnumber	Location
4	Stafford

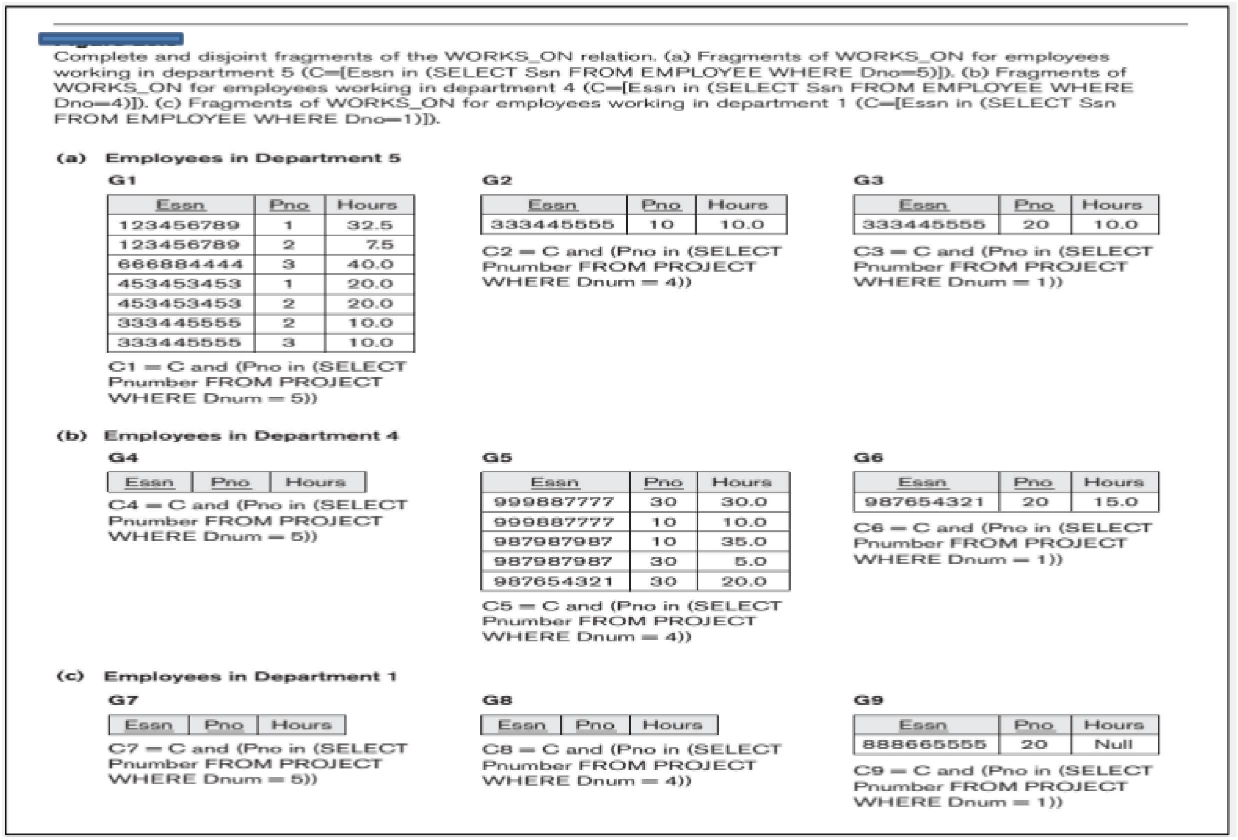
WORKS_ON_4

Essn	Pno	Hours
333445555	10	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0

PROJS_4

Pname	Pnumber	Plocation	Dnum
Computerization	10	Stafford	4
New_benefits	30	Stafford	4

Data at site 3



Complete and disjoint fragments of the WORKS_ON relation. (a) Fragments of WORKS_ON for employees working in department 5 (C=[Essn in (SELECT Ssn FROM EMPLOYEE WHERE Dno=5)]). (b) Fragments of WORKS_ON for employees working in department 4 (C=[Essn in (SELECT Ssn FROM EMPLOYEE WHERE Dno=4)]). (c) Fragments of WORKS_ON for employees working in department 1 (C=[Essn in (SELECT Ssn FROM EMPLOYEE WHERE Dno=1)]).

(a) Employees in Department 5

G1

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0

C1 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 5))

G2

Essn	Pno	Hours
333445555	10	10.0

C2 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 4))

G3

Essn	Pno	Hours
333445555	20	10.0

C3 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 1))

(b) Employees in Department 4

G4

Essn	Pno	Hours

C4 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 5))

G5

Essn	Pno	Hours
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0

C5 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 4))

G6

Essn	Pno	Hours
987654321	20	15.0

C6 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 1))

(c) Employees in Department 1

G7

Essn	Pno	Hours

C7 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 5))

G8

Essn	Pno	Hours

C8 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 4))

G9

Essn	Pno	Hours
888665555	20	Null

C9 = C and (Pno in (SELECT Pnumber FROM PROJECT WHERE Dnum = 1)) either an EMPLOYEE tuple or a PROJECT tuple at sites 2 and 3. Hence, we place the union of fragments G1, G2,G3,G4, and G7at site 2 and the union of fragments G4,G5,G6,G2, and G8 at site3. Notice that fragments G2 andG4 are replicated at both sites.

This allocation strategy permits the join between the local EMPLOYEE or PROJECT fragments at site 2 or site 3 and the local WORKS_ON fragment to be performed completely locally.

This clearly demonstrates how complex the problem of database fragmentation and allocation is for large databases. The Selected Bibliography at the end of this chapter discusses some of the work done in this area.

7.3 Query Processing and Optimization In Distributed Databases

Now we give an overview of how a DDBMS processes and optimizes a query. First, we discuss the steps involved in query processing and then elaborate on the communication costs of processing a distributed query. Finally we discuss a special operation, called a semi join, which is used to optimize some types of queries in a DDBMS.

A detailed discussion about optimization algorithms is beyond the scope of this book. We attempt to illustrate optimization principles using suitable examples.

7.3.1 Distributed Query Processing

A distributed database query is processed in stages as follows:

1. **Query Mapping.** The input query on distributed data is specified formally using a query language. It is then translated into an algebraic query on global relations. This translation is done by referring to the global conceptual schema and does not take into account the actual distribution and replication of data. Hence, this translation is largely identical to the one performed in a centralized DBMS. It is first normalized, analyzed for semantic errors, simplified, and finally restructured into an algebraic query.

2. **Localization.** In a distributed database, fragmentation results in relations being stored in separate sites, with some fragments possibly being replicated. This stage maps the distributed query on the global schema to separate queries on individual fragments using data distribution and replication information.

3. **Global Query Optimization.** Optimization consists of selecting a strategy from a list of candidates that is closest to optimal. A list of candidate queries can be obtained by permuting the ordering of operations within a fragment query generated by the previous stage. Time is the preferred unit for measuring cost. The total cost is a weighted combination of costs such as CPU cost, I/O costs, and communication costs. Since DDBs are connected by a network, often the communication costs over the network are the most significant. This is especially true when the sites are connected through a wide area network (WAN).

4. **Local Query Optimization.** This stage is common to all sites in the DDB. The techniques are similar to those used in centralized systems.The first three stages discussed above are performed at a central control site, while the last stage is performed locally.

6.15, 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 sample queries. Suppose that the EMPLOYEE and DEPARTMENT relations in Figure 3.5 are distributed at two sites as shown in Figure 25.10. We will assume in this example that neither relation is fragmented. According to Figure 25.10, 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’s name and the name of the department for which the employee works. This can be stated as follows in the relational algebra:

$$\text{Q: } \pi_{\text{Fname,Lname,Dname}}(\text{EMPLOYEE} \bowtie_{\text{Dno=Dnumber}} \text{DEPARTMENT})$$

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 + 3,500 = 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 3. 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.

3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3. In this case, 400,000 + 3,500 = 403,500 bytes must be transferred.

If minimizing the amount of data transfer is our optimization criterion, we should choose strategy. 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:

$$Q': \pi_{\text{Fname,Lname,Dname}}(\text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$$

Again, suppose that the query is submitted at site 3. The same three strategies for executing query Q apply to Q, except that the result of Q includes only 100 records, assuming that each department has a manager:

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 + 3,500 = 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 3. The size of the query result is 40 *100 = 4,000 bytes, so 4,000 + 1,000,000 = 1,004,000 bytes must be transferred.

3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3. In this case, 4,000 + 3,500 = 7,500 bytes must be transferred.

Again, we would choose strategy 3 this time by an overwhelming margin over strategies 1 and 2. The preceding three strategies are the most obvious ones for the case where the result site (site 3) is different from all the sites that contain files involved in the query (sites 1 and 2). However, suppose that the result site is site 2; then we have two simple strategies:

1. Transfer the EMPLOYEE relation to site 2, execute the query, and present the result to the user at site 2. Here, the same number of bytes 1,000,000 must be transferred for both Q and Q ‘.

2. Transfer the DEPARTMENT relation to site 1, execute the query at site 1, and send the result back to site 2. In this case 400,000 + 3,500 = 403,500 bytes must be transferred for Q and 4,000 + 3,500 = 7,500 bytes for Q.