SPECIAL EDITION FOR CSEDU STUDENTS

TOUCH-N-PASS EXAM CRAM GUIDE SERIES

DBMS – II

Prepared By

Sharafat Ibn Mollah Mosharraf

CSE, DU

12th Batch (2005-2006)

Table of Contents

CHAPTER 9: OBJECT-BASED DATABASES 1

THEORIES 1
EXCERCISES 6

CHAPTER 15: TRANSACTIONS 11

CHAPTER 16: CONCURRENCY CONTROL 19

CHAPTER 18: DATA ANALYSIS AND MINING 29

DATA ANALYSIS AND OLAP 29
DATA WAREHOUSE 33
DATA MINING 37

CHAPTER 20: DATABASE SYSTEM ARCHITECTURES 42

CHAPTER 21: PARALLEL DATABASES 46

CHAPTER 22: DISTRIBUTED DATABASES 49

CHAPTER 9
OBJECT-BASED DATABASES

Theories

9.1 **What are the major drawbacks of relational data model?**
OR, What is the motivation to use object based data model? [*Marks:* 3]

The major drawbacks of relational data model are:

1. **Limited type system supported by the relational model:**

   Complex application domains require correspondingly complex data types, such as nested record structures, multivalued attributes and inheritance, which are supported by traditional programming languages. Such features had to be translated to simpler relational model data types.

2. **Difficulty in accessing database data from programs written in object-oriented programming languages:**

   Merely extending the type system supported by the database was not enough to solve this problem completely. Differences between the type system of the database and the type system of the programming language make data storage and retrieval more complicated, and need to be minimized. Having to express database access using a language (SQL) which is different from the programming language again makes the job of the programmer harder. It is desirable, for many applications, to have programming language constructs or extensions that permit direct access to data in the database, without having to go through an intermediate language such as SQL.

9.2 **Compare object-oriented and object-relational databases. [2008. *Marks:* 3]**

Object-Oriented Databases	Object-Relational Databases
1. Object-oriented databases are built around persistent programming languages.	1. Object-relational databases are object-oriented databases built on top of the relational model.
2. Targeted towards applications that have high performance requirements.	2. Targeted towards making data modeling and querying easier by using complex data types.
3. Provide low-overhead and thus high performance access to persistent data.	3. Impose significant performance penalty for certain kinds of applications that run primarily in main memory and that perform a large number of accesses to the database.
4. More susceptible to data corruption by programming errors.	4. Provide a good protection of data from programming errors.
5. Typical applications include CAD databases.	5. Typical applications include storage and querying of complex data, including multimedia data.

9.3 **How does the concept of an object in the object-oriented data model differ from the concept of an entity in the entity-relationship model? [*Marks:* 4]**

An entity is simply a collection of variables or data items. An object is an encapsulation of data as well as the methods (code) to operate on the data. The data members of an object are directly visible only to its methods. The outside world can gain access to the object's data only by passing pre-defined messages to it, and these messages are implemented by the methods.

9.4 **How object-relational database features are implemented? [*Marks:* 5]**

Storing complex data types:

The complex data types supported by object-relational systems can be translated to the simpler system of relational databases. The techniques for converting E-R model features to tables can be used, with some extensions, to translate object-relational data to relational data at the storage level.

Storing subtables:

Subtables can be stored in an efficient manner, without replication of all inherited fields, in one of two ways:

- Each table stores the primary key (which may be inherited from a parent table) and the attributes are defined locally. Inherited attributes (other than the primary key) do not need to be stored, and can be derived by means of a join with the supertable, based on the primary key.
- Each table stores all inherited and locally defined attributes. When a tuple is inserted, it is stored only in the table in which it is inserted, and its presence is inferred in each of the supertables. Access to all attributes of a tuple is faster, since a join is not required.

Storing array and multiset types:

Implementations may choose to represent array and multiset types directly, or may choose to use a normalized representation internally. Normalized representations tend to take up more space and require an extra join / grouping cost to collect data in an array or multiset. However, normalized representations may be easier to implement.

9.5 **Compare simple data types in relational model and complex data types in object-relational model. [*Marks:* 3]**

Simple Data Types in Relational Model	Complex Data Types in Object-Relational Model
1. The basic data items are records that are fairly small and whose fields are atomic.	1. Records are large and fields are not atomic.
2. Database is in 1NF.	2. Database is not normalized.
3. There are only a few record types.	3. There are many user-defined records types.

9.6 **Explain Complex Data Types with appropriate examples. [*Marks:* 3]**

Complex data types are structured types which can represent E-R model concepts, such as composite attributes, multivalued attributes, generalization and specialization directly, without a complex translation to the relational model.

For example, we can define the following structured type to represent a composite attribute *name* with component attributes *firstname* and *lastname*:

```
create type Name as (
    firstname varchar(20),
    lastname varchar(20)
)
```

Multivalued attributes can be represented as arrays and multisets:

```
create type Book as (
        title varchar(20),
        author_array varchar(20) array[10],
        pub_date date,
        keyword_set varchar(20) multiset)
)
```

Inheritance of *teacher* from a type *Person* can be defined as follows:

```
create type Person (
        name varchar(20),
        address varchar(20)
```

)

create type *Teacher* **under** *Person* (
salary **integer**,
department **varchar**(20)
)

9.7 **With example explain single and multiple inheritances of types. [*Marks:* 4]**

A single type inheritance of the types *Student* and *Teacher* from *Person* can be defined as follows:

create type *Person* (
name **varchar**(20),
address **varchar**(20)
)

create type *Student* **under** *Person* (
degree **varchar**(20),
department **varchar**(20)
)

create type *Teacher* **under** *Person* (
salary **integer**,
department **varchar**(20)
)

And a multiple inheritance of type *TeachingAssistant* from the types *Student* and *Teacher* can be defined as follows:

create type *TeachingAssistant* **under** *Student, Teacher*

However, the attribute *department* is defined separately in *Student* and *Teacher* and thus conflict in *TeachingAssistant*. To avoid a conflict between the two occurrences of *department*, we can rename them by using an **as** clause:

create type *TeachingAssistant*
under *Student* **with** (*department* **as** *student_dept),*
Teacher **with** (*department* **as** *teacher_dept)*

9.8 **With example explain single and multiple inheritances of tables. [*Marks:* 4]**

Single inheritance of the tables *students* and *teachers* from the table *people* can be created as follows:

create table *people* **of** *Person*
create table *students* **of** *Student* **under** *people*
create table *teachers* **of** *Teacher* **under** *people*

When we declare *students* and *teachers* as subtables of *people*, every tuple present in *students* or *teachers* becomes also implicitly present in *people*. Thus, if a query uses the table *people*, it will find not only tuples directly inserted into that table, but also tuples inserted into its subtables, namely *students* and *teachers*. However, only those attributes that are present in *people* can be accessed.

Multiple inheritance of the table *teaching_assistants* from the tables *students* and *teachers* can be created as follows:

create table *teaching_assistants* **of** *TeachingAssistant* **under** *students, teachers*

As a result of the declaration, every tuple present in the *teaching_assistants* table is also implicitly present in the *teachers* and in the *students* table, and in turn in the *people* table.

9.9 **How arrays and multiset types are defined in SQL? [*Marks:* 4]**

An array of values can be created in SQL in this way:

Array[′Smith′,′Jones′]

Similarly, a multiset of keywords can be constructed as follows:

Multiset['parsing','analysis']

A *book* type using array and multiset can be declared as follows:

```
create type Book as (
        title varchar(20),
        author_array varchar(20) array[10],
        pub_date date,
        publisher Publisher,
        keyword_set varchar(20) multiset)
)
```

A tuple of the relation of type *Book* can be created as:

('Compilers', **array**['Smith','Jones'], **new** *Publisher*('McGraw-Hill','New York'), **multiset**['parsing','analysis'])

9.10 **With example define nesting and unnesting. [*Marks:* 5]**

Unnesting

The transformation of a nested relation into a form with fewer (or no) relation-valued attributes is called ***unnesting***.

Consider the following *books* relation:

title	***author_array***	***publisher* (*name, branch*)**	***keyword_set***
Compilers	[Smith, Jones]	(McGraw-Hill, New York)	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London)	{Internet, Web}

Suppose that we want to convert the relation into a single flat relation, with no nested relations or structured types as attributes. We can use the following query to carry out the task:

select *title, A.author, publisher.name* **as** *pub_name, publisher.branch* **as** *pub_branch, K.keyword*
from *books* **as** *B,* **unnest** (*B.author_array*) **as** *A*(*author*), **unnest** (*B.keyword_set*) **as** *K*(*keyword*)

The result of the preceding query is the following relation which is in 1NF:

title	***author***	***pub_name***	***pub_branch***	***keyword***
Compilers	Smith	McGraw-Hill	New York	parsing
Compilers	Jones	McGraw-Hill	New York	parsing
Compilers	Smith	McGraw-Hill	New York	analysis
Compilers	Jones	McGraw-Hill	New York	analysis
Networks	Jones	Oxford	London	Internet
Networks	Frick	Oxford	London	Internet
Networks	Jones	Oxford	London	Web
Networks	Frick	Oxford	London	Web

Nesting

The reverse process of transforming a 1NF relation into a nested relation is called ***nesting***.

The above 1NF relation can be converted back to the nested relation using the following query:

```
select title, collect(author) as author_set, Publisher(pub_name, pub_branch) as publisher,
        collect(keyword) as keyword_set
from flat_books
group by title, publisher
```

9.11 **Explain the distinction between a type *x* and a reference type ref(*x*). Under what circumstances would you choose to use a reference type?**

If the type of an attribute is *x*, then in each tuple of the table, corresponding to that attribute, there is an actual object of type *x*. If its type is **ref**(*x*), then in each tuple, corresponding to that attribute, there is a reference to some object of type *x*. We choose a reference type for an attribute, if that attribute's intended purpose is to refer to an independent object.

9.12 **What do you mean by persistent programming language? How it differs from high level languages with embedded SQL? [*2008. Marks:* 1 + 3]**

A persistent programming language is a programming language extended with constructs to handle persistent data.

Persistent programming languages differ from high level languages with embedded SQL in the following ways:

1. With an embedded language, the type system of the host language usually differs from the type system of the data-manipulation language. The programmer is responsible for any type conversions between the host language and SQL.

   In contrast, in a persistent programming language, the query language is fully integrated with the host language, and both share the same type system. Objects can be created and stored in the database without any explicit type or format changes; any format changes required are carried out transparently.

2. The programmer using an embedded query language is responsible for writing explicit code to fetch data from the database into memory. If any updates are performed, the programmer must write code explicitly to store the updated data back in the database.

   In contrast, in a persistent programming language, the programmer can manipulate persistent data without writing code explicitly to fetch it into memory or store it back to disk.

9.13 **What are the advantages and drawbacks of persistent programming languages? [*Marks:* 4]**

Advantages of persistent programming languages:

1. Objects can be created and stored in the database without any explicit type or format changes; any format changes required are carried out transparently.
2. The programmer can manipulate persistent data without writing code explicitly to fetch it into memory or store it back to disk.

Disadvantages of persistent programming languages:

1. Since the programming language is usually a powerful one, it is relatively easy to make programming errors that damage the database.
2. The complexity of the language makes automatic high-level optimization, such as to reduce disk I/O, harder.
3. Support for declarative querying is important for many applications, but persistent programming languages currently do not support declarative querying well.

9.14 **Suppose that you have been hired as a consultant to choose a database system for your client's application. For each of the following applications, state what type of database system (relational, persistent programming language–based OODB, object relational; do not specify a commercial product) you would recommend. Justify your recommendation.**

1. **A computer-aided design system for a manufacturer of airplanes.**
2. **A system to track contributions made to candidates for public office.**
3. **An information system to support the making of movies.**

1. An OODB system would be suitable for this. That is because CAD requires complex data

types, and being computation oriented, CAD tools are typically used in a programming language environment needing to access the database.

2. A relational system would be apt for this, as data types are expected to be simple, and a powerful querying mechanism is essential.
3. Here there will be extensive use of multimedia and other complex data types. But queries are probably simple, and thus an object relational system is suitable.

Exercises

9.1 **A car-rental company maintains a vehicle database for all vehicles in its current fleet. For all vehicles, it includes the vehicle identification number, license number, manufacturer, model, date of purchase, and color. Special data are included for certain types of vehicles:**

- **Trucks: cargo capacity**
- **Sports cars: horsepower, renter age requirement**
- **Vans: number of passengers**
- **Off-road vehicles: ground clearance, drivetrain (four- or two-wheel drive)**

Construct an SQL:1999 object-oriented database schema definition for this database. Use inheritance where appropriate. [*2008. Marks: 3*]

```
create type Vehicles (
    vehicle-id int,
    licence-number string,
    manufacturer string,
    model string,
    purchase-date date,
    color string
)

create type Trucks under Vehicles (
    cargo-capacity int
)

create type SportCars under Vehicles (
    horsepower int,
    renter-age-requirement int
)

create type Vanss under Vehicles (
    num-passengers int
)

create type OffRoadVehicles under Vehicles (
    ground-clearance real,
    drivetrain int
)
```

9.2 **Consider a database schema with a relation *Emp* whose attributes are as shown below, with types specified for multivalued attributes.**

Emp = (*ename*, *ChildrenSet* **multiset**(*Children*), *Skillset* **multiset**(*Skills*))
Children = (*name*, *Birthday*)
Birthday = (*day*, *month*, *year*)
Skills = (*type*, *ExamSet* **multiset**(*Exams*))
Exams = (*year*, *city*)

a. Define the above schema with appropriate types for each attribute. [*Marks*: 4]

b. Using the above schema, write the following queries in SQL:

i. Find the names of all employees who have a child who has a birthday in March.

ii. Find those employees who took an examination for the skill type "typing" in the city "Dayton".

iii. List all skill types in the relation *Emp*.

a.

```
create type Exams (
        year char(4),
        city varchar(20)
)
create table exams of type Exams
create type Skills (
        type varchar(20),
        ExamSet Exams multiset
)
create table skills of type Skills
create type Birthday (
        day int(2),
        month char(3),
        year int(4)
)
create type Children (
        name varchar(50),
        Birthday birthday
)
create table children of type Children
create type Emp (
        ename varchar(50),
        ChildrenSet Children multiset,
        SkillSet Skills multiset
)
create table emp of type Emp
```

b.

i.
```
select ename
from emp, emp.ChildrenSet as c
where 'March' in (select birthday.month from c)
```

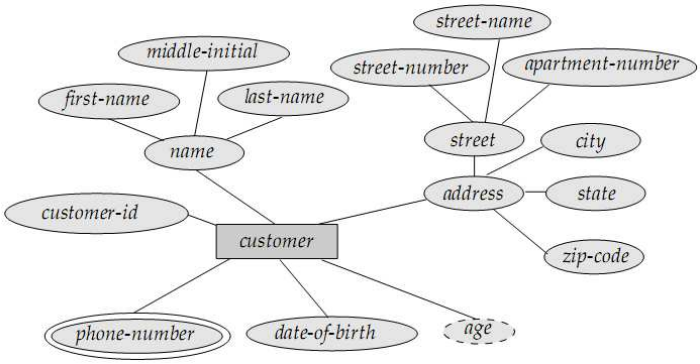
ii.
```
select ename
from emp, emp.SkillSet as s, s.ExamSet as x
where s.type = 'typing' and x.city = 'Dayton'
```

iii.
```
select distinct s.type
from emp, emp.SkillSet as s
```

9.3 Consider the following ERD.

a. Give an SQL:2003 schema definition corresponding to the ERD.

b. Give constructors for each of the structured types defined in (a).



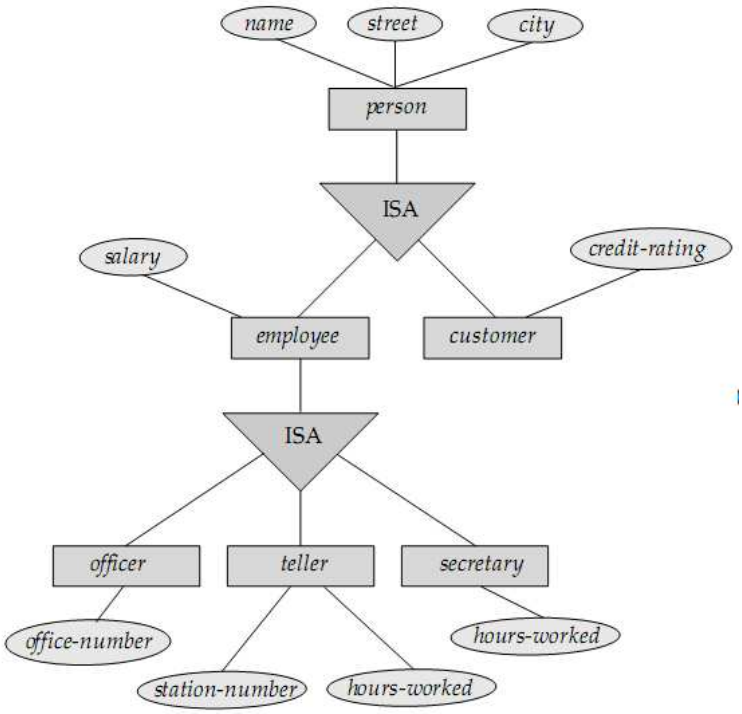
a.

```
create type Name
    (first_name varchar(15),
     middle_initial char,
     last_name varchar(15))
create type Street
    (street_name varchar(15),
     street_number varchar(4),
     apartment_number varchar(7))
create type Address
    (street Street,
     city varchar(15),
     state varchar(15),
     zip_code char(6))
create table customer
    (name Name,
     customer_id varchar(10),
     address Adress,
     phones  char(7)  array[10],
     dob date)
method integer age()
```

b.

```
create function Name (f varchar(15), m char, l varchar(15))
returns Name
begin
    set first_name = f;
    set middle_initial = m;
    set last_name = l;
end
create function Street (sname varchar(15), sno varchar(4), ano varchar(7))
returns Street
begin
    set street_name = sname;
    set street_number = sno;
    set apartment_number =ano;
end
create function Address (s Street, c varchar(15), sta varchar(15), zip varchar(6))
returns Address
begin
    set street = s;
    set city = c;
    set state =sta;
    set zip_code =zip;
end
```

9.4 **Give an SQL:2003 schema definition of the E-R diagram below.**




```
create type Person
    (name varchar(30),
     street varchar(15),
     city varchar(15))
create type Employee
    under Person
    (salary integer)
create type Customer
    under Person
    (credit-rating integer)
create type Officer
    under Employee
    (office-number integer)
create type Teller
    under Employee
    (station-number integer,
    hours-worked integer)
create type Secretary
    under Employee
    (hours-worked integer)
create table person of Person
create table employee of Employee
    under person
create table customer of Customer
    under person
create table officer of Officer
    under employee
create table teller of Teller
    under employee
create table secretary of Secretary
    under employee
```

9.5 **Consider the relational schema below. Give a schema definition in SQL:2003 corresponding to the relational schema, but using references to express foreign-key relationships.**

employee (person_name, street, city)
works (person_name, company_name, salary)
company (company_name, city)
manages (person_name, manager_name)

```
create type Employee
    (person_name varchar(30),
     street varchar(15),
     city varchar(15))
create type Company
    (company_name varchar(15),
    (city varchar(15))
create table employee of Employee
create table company of Company
create type Works
    (person ref(Employee) scope employee,
    comp ref(Company) scope company,
    salary int)
create table works of Works
create type Manages
    (person ref(Employee) scope employee,
    (manager ref(Employee) scope employee)
create table manages of Manages
```

CHAPTER 15
TRANSACTIONS

15.1 **What is transaction? [*2008, In-course 2008-2009. Marks:* 1]**

A *transaction* is a unit of program execution that accesses and possibly updates various data items.

For example, a transfer of funds from a checking account to a savings account consists of several operations from the point of view of the database system. All these operations result into a single transaction.

15.2 **What are the ACID properties of transaction? [*2008, In-course 2008-2009. Marks:* 1]**

OR, Explain the transaction properties that are required for the database system to ensure integrity of the data? [*Marks:* 4]

OR, List the ACID properties. Explain the usefulness of each. [*In-course. Marks: 6*]

To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

- **Atomicity**

  Either all operations of the transaction are reflected properly in the database or none at all.

  This is typically the responsibility of the application programmer who codes the transactions.

- **Consistency**

  Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

  Clearly, lack of atomicity will lead to inconsistency in the database.

- **Isolation**

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

  The user view of a transaction system requires the isolation property, and this is the property that concurrent schedules use to take the system from one consistent state to another. These requirements are satisfied by ensuring that only serializable schedules of individually consistency preserving transactions are allowed.

- **Durability**

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

15.3 **Database-system implementers have paid much more attention to the ACID properties than have file-system implementers. Why might this be the case?**

Database systems usually perform crucial tasks whose effects need to be atomic and durable, and whose outcome affects the real world in a permanent manner. Examples of such tasks are monetary transactions, seat bookings etc. Hence the ACID properties have to be ensured. In contrast, most users of file systems would not be willing to pay the price (monetary, disk space, time) of supporting ACID properties.

15.4 **Draw the state diagram of a transaction and describe each state. [*2008, In-course 2007-2008. Marks:* 3]**

A transaction must be in one of the following states:

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

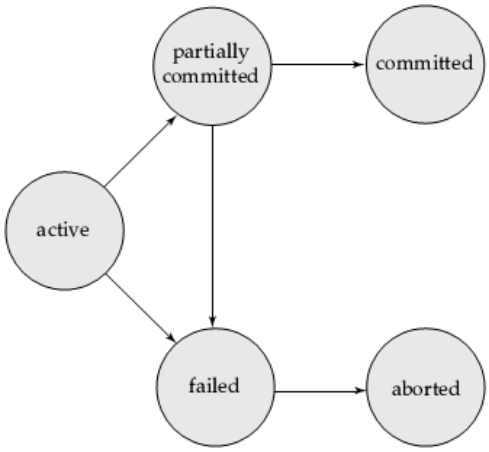


Figure: State diagram of a transaction.

15.5 **During its execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which a transaction may pass. Explain why each state transition may occur.**

The possible sequences of states are:

1. ***Active → partially committed → committed.*** This is the normal sequence a successful transaction will follow. After executing all its statements it enters the partially committed state. After enough recovery information has been written to disk, the transaction finally enters the committed state.
2. ***Active → partially committed → aborted.*** After executing the last statement of the transaction, it enters the partially committed state. But before enough recovery information is written to disk, a hardware failure may occur destroying the memory contents. In this case the changes which it made to the database are undone, and the transaction enters the aborted state.
3. ***Active → failed → aborted.*** After the transaction starts, if it is discovered at some point that normal execution cannot continue (either due to internal program errors or external errors), it enters the failed state. It is then rolled back, after which it enters the aborted state.

15.6 **What are the advantages of concurrent execution of transactions? [2008, *In-course 2007-2008. Marks:* 3]**

OR, Why do database systems support concurrent execution of transactions in spite of the extra programming effort needed to ensure that concurrent execution does not cause any problems?

The advantages of concurrent execution of transactions are:

1. **Improved throughput and resource utilization:** *I/O activity* and *CPU activity* can operate in parallel leading to better transaction *throughput.* One transaction can be using the CPU while another is reading from or writing to the disk. The processor and disk utilization also increase; the processor and disk spend less time idle.
2. **Reduced waiting time and average response time:** Short transactions need not wait behind long ones. If the transactions are operating on different parts of the database, it is better to run them concurrently, sharing the CPU cycle and disk accesses among them. It also reduces the *average response time* - the average time for a transaction to be completed after it has been submitted.

For these reasons, database systems support concurrent execution of transactions in spite of the extra programming effort needed to ensure that concurrent execution does not cause any problems.

15.7 **Justify the following statement: Concurrent execution of transactions is more important when data must be fetched from (slow) disk or when transactions are long, and is less important when data is in memory and transactions are very short. [*In-course 2008-2009. Marks:* 3]**

If a transaction is very long or when it fetches data from a slow disk, it takes a long time to complete. In absence of concurrency, other transactions will have to wait for longer period of time and average response time will increase. Also when the transaction is reading data from disk, CPU is idle. So, resources are not properly utilized. Hence concurrent execution becomes important in this case.

However, when the transactions are short or the data is available in memory, these problems do not occur. Hence concurrent execution is less important in these cases.

15.8 **With example define *schedule* and *serial schedule*. [*Marks:* 4]**

A *schedule* is a sequence of instructions that specify the chronological order in which instructions of transactions are executed.

A *serial schedule* is a schedule which consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule.

For example, the schedules S_1 and S_2 shown below are both schedules; but S_2 is a serial schedule whereas S_1 is not.

Schedule S_1

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule, but not serial schedule.

Schedule S_2

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Serial schedule.

15.9 **Explain the distinction between the terms *serial schedule* and *serializable schedule*. [*In-course 2007-2008. Marks:* 2]**

A schedule in which all the instructions belonging to one single transaction appear together is called a *serial schedule.*

A schedule which is equivalent to a serial schedule is called a *serializable schedule*.

For example, the following schedule S_1 is not a serial schedule, but it is a serializable schedule which is equivalent to the serial schedule S_2 below.

[*See the schedules S_1 and S_2 in question 15.8*]

15.10 **With examples show the differences among *schedule, serial schedule* and *serializable schedule*. [*In-course 2008-2009. Marks:* 5]**

[*Combine the answers of questions 15.8 and 15.9. Try yourself. ☺]*

15.11 **How do you identify conflicting instructions to determine conflict serializability? [*Marks:* 3]**

Let us consider a schedule S in which there are two consecutive instructions, l_i and l_j ($i \neq j$) of transactions T_i and T_j respectively.

If l_i and l_j, refer to *different* data item, then we can swap them without affecting the results of any instruction in the schedule.

However, if l_i and l_j refer to *the same* data item Q, then the order of the two steps may matter.

The following four cases need to be considered:

1. I_i = `read`(Q), I_j = `read`(Q).

   The order of I_i and I_j does not matter, since the same value of Q is read by T_i and T_j, regardless of the order.

2. I_i = `read`(Q), I_j = `write`(Q).

   If I_i comes before I_j, then T_i does not read the value of Q written by T_j in instruction I_j.

   If I_j comes before I_i, then T_i reads the value of Q written by T_j.

   Thus, the order of I_i and I_j matters.

3. I_i = `write`(Q), I_j = `read`(Q).

   The order of I_i and I_j matters for reasons similar to those of the previous case.

4. I_i = `write`(Q), I_j = `write`(Q).

   Since both instructions are write operations, the order of these instructions does not affect either of T_i or T_j.

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

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

Thus, we can say that two instructions in a schedule conflict if and only if there exists *same* item Q accessed by both of them and at least one of these instructions is a write operation on Q.

15.12 **What do you understand by conflict equivalent and view equivalent schedules? Explain with example. [*Marks:* 5]**

OR, Distinguish between conflict serializable schedule and view serializable schedule with proper example. [*In-course. Marks: 6*]

Conflict Equivalent Schedules:

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.*

Conflict Serializable Schedule:

We say that a schedule S is *conflict serializable* if it is conflict equivalent to a serial schedule.

For example, schedule 1 below is conflict equivalent to schedule 2. The sequence of swaps of non-conflicting instructions is also shown.

Schedule 1 **Schedule 2**

T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
read(A)		read(A)		read(A)		read(A)		read(A)	
write(A)		write(A)		write(A)		write(A)		write(A)	
	read(A)		read(A)	read(B)		read(B)		read(B)	
	write(A)	read(B)			read(A)		read(A)	write(B)	
read(B)			write(A)		write(A)	write(B)			read(A)
write(B)		write(B)		write(B)			write(A)		write(A)
	read(B)		read(B)		read(B)		read(B)		read(B)
	write(B)		write(B)		write(B)		write(B)		write(B)

Figure: Transforming Schedule 1 – which is conflict equivalent to serial schedule 2 – into Serial Schedule 2.

View Equivalent Schedules:

Let S and $S´$ be two schedules with the same set of transactions. The schedules S and $S´$ are said to be *view equivalent* if the following three conditions are met:

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S', also read the initial value of Q.
2. For each data item Q, if transaction T_i executes **read**(Q) in schedule S, and that value was produced by a **write**(Q) operation executed by transaction T_j, then the **read**(Q) operation of transaction T_i must, in schedule S', also read the value of Q that was produced by the same **write**(Q) operation in transaction T_j.
3. For each data item Q, the transaction (if any) that performs the final **write**(Q) operation in schedule S must perform the final **write**(Q) operation in schedule S'.

View Serializable Schedule:

We say that a schedule S is *view serializable* if it is view equivalent to a serial schedule.

For example, Schedule 3 below is view serializable to serial schedule 4, but it is not conflict serializable, since every pair of consecutive instructions conflict, and, thus, no swapping of instructions is possible.

Schedule 3

T_3	T_4	T_5
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Schedule 4

T_3	T_4	T_5
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

15.13 **What are the conditions to determine view equivalence? [*Marks:* 3]**

[*See question 15.12*]

15.14 **Give an example of a schedule that is view equivalent but not conflict equivalent. Explain with causes. [*In-course 2008-2009. Marks:* 4]**

As an example, Schedule 1 below is view serializable to schedule 2, but it is not conflict serializable, since every pair of consecutive instructions conflict, and, thus, no swapping of instructions is possible.

Schedule 1

T_3	T_4	T_5
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Schedule 2

T_3	T_4	T_5
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

A schedule is view equivalent but not conflict equivalent when it contains blind writes, i.e. the schedule contains transactions which perform write operations without having performed a read operation.

15.15 **"Every conflict-serializable schedule is view serializable" – explain how? [2007. *Marks:* 3]**

Every conflict-serializable schedule is also view serializable as all the conditions of view equivalence also holds for conflict-serializable schedule. For example, Schedule 1 below is conflict equivalent to Schedule 2.

Schedule 1

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule 2

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Schedule 1 is also view equivalent to Schedule 2 because:

1. T_1 reads the initial values of A and B in both schedules.
2. T_2 reads the value of A and B which were written by T_1 in both schedules.
3. T_2 performs the final **write**(A) and **write**(B) operations in both schedules.

15.16 **Since every conflict-serializable schedule is view serializable, why do we emphasize conflict serializability rather than view serializability? [2008, *In-course 2007-2008. Marks:* 3]**

Most of the concurrency control protocols (protocols for ensuring that only serializable schedules are generated) used in practice are based on conflict serializability — they actually permit only a subset of conflict serializable schedules (to make the schedules recoverable and cascadeless). The general form of view serializability is very expensive to test, and only a very restricted form of it is used for concurrency control. That is why we emphasize conflict serializability rather than view serializability.

15.17 **Consider a database with objects X and Y and assume that there are two transactions T_1 and T_2. Transaction T_1 reads objects X and Y and then writes object X. Transaction T_2 reads objects X and Y and then writes objects X and Y.**

i. Give an example schedule with actions of transactions T_1 and T_2 on objects X and Y that results in a write-read conflict.
ii. Give an example schedule with actions of transactions T_1 and T_2 on objects X and Y that results in a read-write conflict.
iii. Give an example schedule with actions of transactions T_1 and T_2 on objects X and Y that results in a write-write conflict.
iv. For each of the above three schedules, show that Strict Two-Phase Lock (2PL) protocol disallows the schedule. [*2006. Marks: 3 × 2 + 4 = 10*]

The transactions are as follows:

T_1: read(X);
read(Y);
write(X);
T_2: read(X);
read(Y);
write(X);
write(Y);

i.

T_1	T_2
read(X)	
read(Y)	
	read(X)
write(X)	
	read(Y)
	write(X)
	write(Y)

ii.

T_1	T_2
	read(X)
	read(Y)
read(X)	
	write(X)
	write(Y)
read(Y)	
write(X)	

iii.

T_1	T_2
read(X)	
read(Y)	
	read(X)
	read(Y)
write(X)	
	write(X)
	write(Y)

iv. If strict 2PL is used, then in (i) and (iii), T_1 would acquire lock-X(X) and T_2 would not be able to acquire lock-X(X) and subsequently lock-X(Y) until T_1 commits. As for the case in (ii), T_2 would acquire lock-X(X) and hence T_1 would not be able to acquire lock-X(X) until T_2 commits. Thus strict 2PL disallows the above three schedules.

15.18 **What is a recoverable schedule? [*In-course 2007-2008. Marks:* 2]**

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

15.19 **Why is recoverability of schedules desirable?**

Recoverable schedules are desirable because failure of a transaction might otherwise bring the system into an irreversibly inconsistent state.

15.20 **Are there any circumstances under which it would be desirable to allow non-recoverable schedules? Explain in detail. [*Marks:* 2]**

Non-recoverable schedules may sometimes be needed when updates must be made visible early due to time constraints, even if they have not yet been committed, which may be required for very long duration transactions.

15.21 **Suppose that there is a database system that never fails. Is a recovery manager required for this system?**

Even in this case the recovery manager is needed to perform roll-back of aborted transactions.

15.22 **What is cascadeless schedule? [*In-course 2007-2008, 2008-2009. Marks:* 2]**

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

15.23 **What is the importance of cascadeless schedules? [*Marks:* 2]**

OR, Why is cascadelessness of schedules desirable? [*In-course 2008-2009. Marks:* 1]

Cascadeless schedules are desirable because the failure of a transaction does not lead to the aborting of any other transaction. Of course this comes at the cost of less concurrency.

15.24 **Are there any circumstances under which it would be desirable to allow noncascadeless schedules? Explain your answer. [*In-course 2008-2009. Marks:* 1]**

If failures occur rarely so that we can pay the price of cascading aborts for the increased concurrency, noncascadeless schedules might be desirable.

15.25 **How conflict serializability is tested through precedence graph? [*In-course 2008-2009. Marks:* 5]**

Testing for Conflict Serializability:

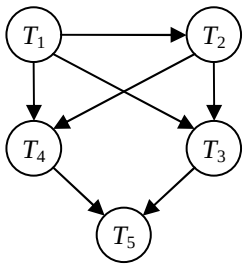
1. Consider a schedule S of a set of transactions $T_1, T_2, ..., T_n$.
2. We construct a directed graph called *precedence graph***,** from S. This graph consists of a pair $G = (V, E)$, where V is the set of vertices and E is a set of edges.

   The set of vertices consists of all the transactions participating in the schedule. The set of edges consists of all the edges $T_i \rightarrow T_j$ for which one of the three conditions holds:

   a. T_i executes write(Q) before T_j executes read(Q).
   b. T_i executes read(Q) before T_j executes write(Q).
   c. T_i executes write(Q) before T_j executes write(Q).

   If an edge $T_i \rightarrow T_j$ exists in the precedence graph, then, in any serial schedule S' equivalent to S, T_i must appear before T_j.
3. If the precedence graph has a cycle, then schedule S is *not* conflict serializable. If no cycle exists, then it is conflict serializable.

15.26 **Consider the following precedence graph. Is the corresponding schedule conflict serializable? Explain your answer. [*2007, In-course 2008-2009. Marks: 4*]**



Yes, the corresponding schedule is conflict serializable, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, $<T_1, T_2, T_3, T_4, T_5>$.

15.27 **Consider the following two transactions:**

T_1: read(A);
read(B);
if $A = 0$ then $B := B + 1$;
write(B);
T_2: read(B);
read(A);
if $B = 0$ then $A := A + 1$;
write(A);

Let the consistency requirement be $A = 0 \vee B = 0$, with $A = B = 0$ the initial values.

a. Show that every serial execution involving these two transactions preserves the consistency of the database.
b. Show a concurrent execution of T_1 and T_2 that produces a nonserializable schedule.
c. Is there a concurrent execution of T_1 and T_2 that produces a serializable schedule?

a. There are two possible executions: T_1T_2 and T_2T_1.

Case 1:

	A	B
Initially	0	0
After T_1	0	1
After T_2	0	1

Consistency met: $(A = 0) \vee (B = 0) \equiv (T \vee F = T)$

Case 2:

	A	B
Initially	0	0
After T_2	1	0
After T_1	1	0

Consistency met: $(A = 0) \vee (B = 0) \equiv (F \vee T = T)$

b. Any interleaving of T_1 and T_2 results in a non-serializable schedule.

T_1	T_2
read(A)	
	read(B)
	read(A)
read(B)	
if $A = 0$ then $B = B + 1$	
	if $B = 0$ then $A = A + 1$
	write(A)
write(B)	

c. There is no parallel execution resulting in a serializable schedule. From part (a), we know that a serializable schedule results in $A = 0 \vee B = 0$. Suppose we start with T_1 read(A). Then when the schedule ends, no matter when we run the steps of T_2, $B = 1$. Now suppose we start executing T_2 prior to completion of T_1. Then T_2 read(B) will give B a value of 0. So when T_2 completes, $A = 1$. Thus $(B = 1 \wedge A = 1) \rightarrow \neg (A = 0 \vee B = 0)$.

Similar case happens for starting with T_2 read(B).

CHAPTER 16
CONCURRENCY CONTROL

16.1 **What is concurrency control scheme? [*In-course 2008-2009. Marks: 2*]**

When several transactions execute concurrently in the database, to ensure the isolation property, the system must control the interaction among the concurrent transactions and this control is achieved through one of the variety of mechanisms called *concurrency-control* schemes.

16.2 **What is locking protocol?**

OR, Define locking protocol. [*In-course 2008-2009. Marks: 2*]

A locking protocol is a set of rules followed by all transactions while requesting and releasing locks.

16.3 **What are the modes of locking in lock-based protocols? [*Marks: 2*]**

Data items can be locked in two modes:

1. **Shared (S) Mode –** data item can only be read. If a transaction T_i has obtained a shared-mode lock on item Q, then T_i can read, but cannot write Q. S-lock is requested using **lock-S** instruction.
2. **Exclusive (X) Mode –** data item can be both read and written. If a transaction T_i has obtained an exclusive-mode lock on item Q, then T_i can both read and write Q. X-lock is requested using **lock-X** instruction.

16.4 **What are the functions of a lock manager? [*In-course 2008-2009. Marks: 4*]**

16.5 **How starvation arises in case of granting of locks? How starvation can be avoided? [*2008. Marks: 1 + 2*]**

Also, How concurrency control manager grants locks so that starvation does not arise? [*2008. Marks: 2*]

A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item. Thus the first transaction never makes progress and starvation arises.

To avoid starvation, when a transaction T_i requests a lock on a data item Q in a particular mode M, the lock can be granted provided that:

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

16.6 **With example define two-phase locking protocol. [*In-course 2008-2009. Marks: 3*]**

OR, How two-phase locking protocol works? Explain with example.

The two-phase locking protocol requires that each transaction issue lock and unlock requests in two phases:

1. Growing phase
   - Transaction may obtain locks
   - Transaction may not release any lock
2. Shrinking phase
   - Transaction may release locks
   - Transaction may not obtain any new locks

Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.

T_1
lock-X(B) read(B) B := B – 50 write(B) lock-X(A) read(A) A := A + 50 write(A) unlock(B) unlock(A)

For example, transaction T_1 in the figure is two-phase.

16.7 **Consider the following two transactions:**

T_1: read(A);
read(B);
if $A = 0$ then $B := B + 1$;
write(B).

T_2: read(B);
read(A);
if $B = 0$ then $A := A + 1$
write(A).

a. Add lock and unlock instructions to transactions T_1 and T_2, so that they observe the two-phase locking protocol. [*2007. Marks: 4*]

b. Can the execution of these transactions result in a deadlock?

a. Lock and unlock instructions:

T_1: lock-S(A)
read(A);
lock-X(B);
read(B);
if $A = 0$ then $B := B + 1$;
write(B);
unlock(A);
unlock(B).

T_2: lock-S(B)
read(B);
lock-X(A);
read(A);
if $B = 0$ then $A := A + 1$;
write(A);
unlock(B);
unlock(A).

b. Execution of these transactions can result in deadlock. For example, consider the following partial schedule:

T_1	T_2
lock-S(A)	
	lock-S(B)
	read(B)
read(A)	
lock-X(B)	
	lock-X(A)

The transactions are now deadlocked.

16.8 **Show that the two-phase locking protocol ensures conflict serializability and that transactions can be serialized according to their lock points. [*Marks: 3*]**

Suppose two-phase locking does not ensure serializability. Then there exists a set of transactions T_0, T_1, ..., T_{n-1} which obey 2PL and which produce a non-serializable schedule. A non-serializable schedule implies a cycle in the precedence graph, and we shall show that 2PL cannot produce such cycles. Without loss of generality, assume the following cycle exists in the precedence graph:

$$T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_{n-1} \rightarrow T_0$$

Let α_i be the time at which T_i obtains its last lock (i.e. T_i's lock point). Then for all transactions such that $T_i \rightarrow T_j$, $\alpha_i < \alpha_j$. Then, for the cycle, we have

	$\alpha_0 < \alpha_1 < \alpha_2 < \ldots < \alpha_{n-1} < \alpha_0$ Since $\alpha_0 < \alpha_0$ is a contradiction, no such cycle can exist. Hence, 2PL cannot produce non-serializable schedules. Because of the property that for all transactions such that $T_i \rightarrow T_j$, $\alpha_i < \alpha_j$, the lock point ordering of the transactions is also a topological sort ordering of the precedence graph. Thus transactions can be serialized according to their lock points.
16.9	**Define compatibility function/lock compatibility matrix. [*Marks: 2*]** Given a set of lock modes, we can define a compatibility function on them as follows. Let *A* and *B* represent arbitrary lock modes. Suppose that a transaction T_i requests a lock of mode *A* on item *Q* on which transaction T_j ($T_i \neq T_j$) currently holds a lock of mode *B*. If transaction T_i can be granted a lock on *Q* immediately, in spite of the presence of the mode *B* lock, then we say mode *A* is compatible with mode *B*. Such a function can be represented conveniently by a matrix, which is called the lock compatibility matrix. For example, the compatibility relation between the *shared* and *exclusive* modes of locking appears in the matrix below. An element (*A*, *B*) of the matrix has the value *true* if and only if mode *A* is compatible with mode *B*. **S** / **X**: S: true, false; X: false, false
16.10	**What are the advantages and drawbacks of two-phase locking protocol? [*In-course 2008-2009. Marks: 3*]** Advantage: Ensures conflict serializability. Drawbacks: 1. Doesn't ensure freedom from deadlock. 2. Cascading rollback might occur. 3. Concurrency might become less.
16.11	**What benefit is provided by strict two-phase locking? What disadvantages result? [*2007. Marks: 3*]** Because strict 2PL produces only cascadeless schedules, recovery is very easy. However, the set of schedules obtainable is a subset of those obtainable from plain two phase locking, thus concurrency is reduced.
16.12	**Most implementations of database systems use strict two-phase locking. Suggest three reasons for the popularity of this protocol.** **OR, Why strict two-phase locking is preferred to other protocols in implementing database systems?** 1. Relatively simple to implement. 2. Imposes low rollback overhead because of cascadeless schedules. 3. Usually allows an acceptable level of concurrency.
16.13	**What benefit does rigorous two-phase locking provide? How does it compare with other forms of two-phase locking?** Rigorous two-phase locking has the advantages of strict 2PL. In addition, it has the property that for two conflicting transactions, their commit order is their serializability order. In some systems users might expect this behavior.
16.14	**Can strict two-phase or rigorous two-phase locking protocols ensure freedom from deadlock? [*In-course 2008-2009. Marks: 1*]** No. No form of 2PL ensures freedom from deadlock.

16.15 **Compare two-phase, strict two-phase and rigorous two-phase protocol. [*2008, In-course 2007-2008. Marks: 3*]**

In 2PL, each transaction issues lock and unlock requests in two phases:

1. Growing Phase – May obtain locks, but may not release locks

2. Shrinking Phase – May release locks, but may not obtain any new locks

In strict 2PL, in addition to above, all exclusive-mode locks taken by a transaction are held until that transaction commits.

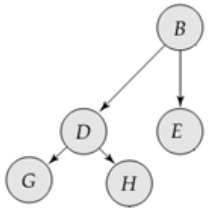
Rigorous 2PL is like strict 2PL except that instead of exclusive-mode locks only, *all* locks taken by a transaction are held until that transaction commits.

16.16 **With example show how tree protocol works? [*Marks: 4*]**

In the tree protocol, the only lock instruction allowed is lock-X. Each transaction T_i can lock a data item at most once, and must observe the following rules:

1. The first lock by T_i may be on any data item.
2. Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.
3. Data items may be unlocked at any time.
4. A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i.

Example:



T_1	T_2
lock-X(B)	
	lock-X(D)
	lock-X(H)
	unlock(D)
lock-X(E)	
lock-X(D)	
unlock(B)	
unlock(E)	
	unlock(H)
unlock(D)	

16.17 **What are the advantages and disadvantages of tree protocol? [*2008, In-course 2007-2008. Marks: 2*]**

Advantages:

1. Ensures conflict serializability.

2. Ensures freedom from deadlock.

Disadvantages:

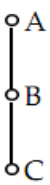
1. Doesn't ensure recoverability and cascadelessness.

2. In some cases, a transaction may have to lock data items that it does not access. This additional locking results in increased locking overhead, the possibility of additional waiting time, and a potential decrease in concurrency.

3. Without prior knowledge of what data items will need to be locked, transactions will have to lock the root of the tree, and that can reduce concurrency greatly.

16.18 **Show by example that there are schedules possible under the tree protocol that are not possible under the two-phase locking protocol and vice-versa. [*In-course 2008-2009. Marks: 3*]**

Consider the tree-structured database graph given below.



Schedule possible under tree protocol but not under 2PL:

T_1	T_2
lock(A)	
lock(B)	
unlock(A)	
	lock(A)
lock(C)	
unlock(B)	
	lock(B)
	unlock(A)
	unlock(B)
unlock(C)	

Schedule possible under 2PL but not under tree protocol:

T_1	T_2
lock(A)	
	lock(B)
lock(C)	
	unlock(B)
unlock(A)	
unlock(C)	

16.19 **Show by example that there are schedules possible under the timestamp protocol that are not possible under the two-phase locking protocol and vice-versa.**

A schedule which is allowed in the two-phase locking protocol but not in the timestamp protocol is:

step	T_0	T_1	Precedence remarks
1	**lock-S**(A)		
2	**read**(A)		
3		**lock-X**(B)	
4		**write**(B)	
5		**unlock**(B)	
6	**lock-S**(B)		
7	**read**(B)		$T_1 \rightarrow T_0$
8	**unlock**(A)		
9	**unlock**(B)		

This schedule is not allowed in the timestamp protocol because at step 7, the W-timestamp of B is 1.

A schedule which is allowed in the timestamp protocol but not in the two-phase locking protocol is:

step	T_0	T_1	T_2
1	**write**(A)		
2		**write**(A)	
3			**write**(A)
4	**write**(B)		
5		**write**(B)	

This schedule cannot have lock instructions added to make it legal under two-phase locking protocol because T_1 must **unlock(*A*)** between steps 2 and 3, and must **lock(*B*)** between steps 4 and 5.

16.20 **What are the advantages and disadvantages of timestamp-ordering protocol? [*In-course 2008-2009. Marks: 4*]**

Advantages:

1. Ensures conflict serializability.
2. Ensures freedom from deadlock as no transaction ever waits.

Disadvantages:

1. Possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction.
2. Generates schedules that are not recoverable and may require cascading rollbacks.

16.21 **In timestamp ordering, W-timestamp(Q) denotes the largest timestamp of any transaction that executed write(Q) successfully. Suppose that, instead, we defined it to be the timestamp of the most recent transaction to execute write(Q) successfully. Would this change in wording make any difference? Explain your answer.**

It would make no difference. The write protocol is such that the most recent transaction to write an item is also the one with the largest timestamp to have done so.

16.22 **When a transaction is rolled back under timestamp ordering, is it assigned a new timestamp? Why can it not simply keep its old timestamp? [*Marks: 2 + 2*]**

A transaction is rolled back because a newer transaction has read or written the data which it was supposed to write. If the rolled-back transaction is re-introduced with the same timestamp, the same reason for rollback is still valid, and the transaction will have to be rolled back again. This will continue indefinitely.

16.23 **Show, with an example, how Thomas' write rule modifies the timestamp-ordering protocol? [*2007. Marks: 3*]**

The timestamp ordering protocol requires that T_i be rolled back if T_i issues write(Q) and TS(T_i) < W-timestamp(Q). However, in Thomas' Write Rule, in those cases where TS(T_i) ≥ R-timestamp(Q), we ignore the obsolete write.

For example, in the following schedule, in case of timestamp-ordering protocol, T_i would be rolled back. However, if Thomas' write rule is applied, the write(Q) operation in T_i is ignored only.

T_i	T_{i+1}
read(Q)	
	write(Q)
write(Q)	

16.24 **Under a modified version of the timestamp protocol, we require that a commit bit be tested to see whether a read request must wait. Explain how the commit bit can prevent cascading abort. Why is this test not necessary for write requests?**

Using the commit bit, a read request is made to wait if the transaction which wrote the data item has not yet committed. Therefore, if the writing transaction fails before commit, we can abort that transaction alone. The waiting read will then access the earlier version in case of a multiversion system, or the restored value of the data item after abort in case of a single-version system. For writes, this commit bit checking is unnecessary. That is because either the write is a "blind" write and thus independent of the old value of the data item or there was a prior read, in which case the test was already applied.

16.25 **Consider the validation-based concurrency-control scheme. Show that by choosing Validation(T_i), rather than Start(T_i), as the timestamp of transaction T_i, we can expect better response time provided that conflict rates among transactions are indeed low.**

In the validation-based concurrency control scheme, choosing Start(T_i) as the timestamp of T_i gives a subset of the schedules allowed by choosing Validation(T_i) as the timestamp. Using Start(T_i) means that whoever started first must finish first. Clearly transactions could enter the validation phase in the same order in which they began executing, but this is overly restrictive. Since choosing Validation(T_i) causes fewer non-conflicting transactions to restart, it gives the better response times.

16.26 **Why multiple granularities are used in case of locking? What is intention mode lock? [*Marks: 2 + 2*]**

There are circumstances where it would be advantageous to group several data items, and to treat them as one individual synchronization unit. For example, if a transaction T_i needs to access the entire database, and a locking protocol is used, then T_i must lock each item in the database. Clearly, executing these locks is time consuming. It would be better if T_i could issue a single lock request to lock the entire database. What is needed is a mechanism to allow the system to define multiple levels of granularity. Hence multiple granularities are used in case of locking.

Intention-mode lock is a class of locks using which the system using multiple granularities can determine if a node can be locked. If a node is locked in an intention mode, it means explicit locking is being done at a lower level of the tree.

16.27 **In multiple-granularity locking, what is the difference between implicit and explicit locking? [*In-course 2007-2008. Marks: 2*]**

When a transaction explicitly locks a node in shared or exclusive mode, it implicitly locks all the descendents of that node in the same mode. The transaction need not explicitly lock the descendent nodes. There is no difference in the functionalities of these locks, the only difference is in the way they are acquired, and their presence tested.

16.28 **Although SIX mode is useful in multiple-granularity locking, an exclusive and intend-shared (XIS) mode is of no use. Why is it useless?**

An exclusive lock is incompatible with any other lock kind. Once a node is locked in exclusive mode, none of the descendents can be simultaneously accessed by any other transaction in any mode. Therefore an exclusive and intend-shared declaration has no meaning.

16.29 **What is deadlock? [*In-course 2007-2008. Marks: 1*]**

A system is said to be in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. More precisely, there exists a set of waiting transactions $\{T_0 , T_1 , . . ., T_n\}$ such that T_0 is waiting for a data item that T_1 holds, and T_1 is waiting for a data item that T_2 holds, and …, and T_{n-1} is waiting for a data item that T_n holds, and T_n is waiting for a data item that T_0 holds. None of the transactions can make progress in such a situation.

16.30 **What are the methods to deal with deadlock problem? [*Marks: 4*]**

There are two principal methods for dealing with deadlock problem

1. We can use a *deadlock prevention* protocol to ensure that the system will never enter a deadlock state.

   There are two approaches to deadlock prevention:

   1. Ensuring that no cycle waits can occur by ordering the requests for locks, or requiring all locks to be acquired together.
   2. Using preemption and transaction rollbacks.

      The wait-die and wound-wait schemes use this approach.

2. Alternatively, we can allow the system to enter a deadlock state, and then try to recover by using a *deadlock detection and deadlock recovery* scheme.

   The wait-for graph is used to detect deadlocked transactions and they are rolled back using the following actions:

   1. Selection of a victim
   2. Rollback
   3. Handling starvation

There exists another method for dealing with deadlock problem – called the timeout-based scheme. In this approach, a transaction that has requested a lock waits for at most a specified amount of time. If the lock has not been granted within that time, the transaction is said to time out, and it rolls itself back and restarts.

16.31 **Under what conditions is it less expensive to avoid deadlock than to allow deadlocks to occur and then to detect them?**

Deadlock avoidance is preferable if the consequences of abort are serious (as in interactive transactions), and if there is high contention and a resulting high probability of deadlock.

16.32 **Describe some of the prevention strategies/approach to restrict deadlock. [*Marks: 5*]**

There are two approaches to deadlock prevention:

1. Ensuring that no cycle waits can occur by ordering the requests for locks, or requiring all

locks to be acquired together.

2. Using preemption and transaction rollbacks.

   The wait-die and wound-wait schemes use this approach.

   [*Now describe the wait-die and wound-wait schemes as in the following answer (16.32).*]

16.33 **Describe wait-die and wound-wait schemes for deadlock prevention.**

OR, What are the deadlock prevention schemes using timestamps?

OR, How deadlock can be prevented using timestamps? [*In-course 2008-2009. Marks: 4*]

1. **Wait-Die Scheme**

   When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (i.e. T_i is older than T_j). Otherwise, T_i is rolled back (dies).

   For example, suppose that transactions T_{22}, T_{23} and T_{24} have timestamps 5, 10 and 15 respectively. If T_{22} requests a data item held by T_{23}, then T_{22} will wait. If T_{24} requests a data item held by T_{23}, then T_{24} will be rolled back.

2. **Wound-Wait Scheme**

   When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j (i.e. T_i is younger than T_j). Otherwise, T_j is rolled back (T_j is wounded by T_i).

   Returning to our example, with transactions T_{22}, T_{23} and T_{24}, if T_{22} requests a data item held by T_{23}, then the data item will be preempted from T_{23}, and T_{23} will be rolled back. If T_{24} requests a data item held by T_{23}, then T_{24} will wait.

16.34 **If deadlock is avoided by deadlock avoidance schemes, is starvation still possible? Explain your answer.**

A transaction may become the victim of deadlock-prevention roll-back arbitrarily many times, thus creating a potential starvation situation.

16.35 **How wait-for graph is used for deadlock detection? [*In-course 2007-2008. Marks: 2*]**

A deadlock exists in the system if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, the system needs to maintain the wait-for graph, and periodically to invoke an algorithm that searches for a cycle in the graph.

16.36 **What are the steps used for deadlock recovery? [*Marks: 5*]**

OR, What is the process of recovering from deadlock? [*2008. Marks: 2*]

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

1. Selection of a Victim

Given a set of deadlocked transactions, we must determine which transaction (or transactions) to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one. Many factors may determine the cost of a rollback, including

a. How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task.
b. How many data items the transaction has used.
c. How many more data items the transaction needs for it to complete.
d. How many transactions will be involved in the rollback.

2. Rollback

Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back.

1. **Total Rollback:** Abort transaction and then restart it.
2. **Partial Rollback:** Roll back the transaction only as far as necessary to break the deadlock.

3. Starvation

In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result, this transaction never completes its designated task, thus there is starvation. We must ensure that transaction can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

16.37 **What are the factors for selecting a victim during deadlock recovery? [*Marks: 5*]**

a. How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task.
b. How many data items the transaction has used.
c. How many more data items the transaction needs for it to complete.
d. How many transactions will be involved in the rollback.

16.38 **For each of the following protocols, describe aspects of practical applications that would lead you to suggest using the protocol, and aspects that would suggest not using the protocol:**

- **Two-phase locking**
- **Two-phase locking with multiple-granularity locking**
- **The tree protocol**
- **Timestamp ordering**
- **Validation**
- **Multiversion timestamp ordering**
- **Multiversion two-phase locking**

- **Two-phase locking:** Use for simple applications where a single granularity is acceptable. If there are large read-only transactions, multiversion protocols would do better. Also, if deadlocks must be avoided at all costs, the tree protocol would be preferable.
- **Two-phase locking with multiple granularity locking:** Use for an application mix where some applications access individual records and others access whole relations or substantial parts thereof. The drawbacks of 2PL mentioned above also apply to this one.
- **The tree protocol:** Use if all applications tend to access data items in an order consistent with a particular partial order. This protocol is free of deadlocks, but transactions will often have to lock unwanted nodes in order to access the desired nodes.
- **Timestamp ordering:** Use if the application demands a concurrent execution that is equivalent to a particular serial ordering (say, the order of arrival), rather than any serial ordering. But conflicts are handled by roll-back of transactions rather than waiting, and schedules are not recoverable. To make them recoverable, additional overheads and increased response time have to be tolerated. Not suitable if there are long read-only transactions, since they will starve. Deadlocks are absent.
- **Validation:** If the probability that two concurrently executing transactions conflict is low, this protocol can be used advantageously to get better concurrency and good response times with low overheads. Not suitable under high contention, when a lot of wasted work will be done.
- **Multiversion timestamp ordering:** Use if timestamp ordering is appropriate but it is desirable

	for read requests to never wait. Shares the other disadvantages of the timestamp ordering protocol. • **Multiversion two-phase locking:** This protocol allows read-only transactions to always commit without ever waiting. Update transactions follow 2PL, thus allowing recoverable schedules with conflicts solved by waiting rather than roll-back. But the problem of deadlocks comes back, though read-only transactions cannot get involved in them. Keeping multiple versions adds space and time overheads though; therefore plain 2PL may be preferable in low conflict situations.

CHAPTER 18
DATA ANALYSIS AND MINING

Data Analysis and OLAP

18.1 **What is the difference between transaction processing system and decision support system? [*Marks:* 3]**

TPS	DSS
1. Transaction-processing systems are systems that record information about transactions.	1. Decision-support systems aim to get high-level information out of the detailed information stored in transaction-processing systems, and to use the high-level information to make a variety of decisions.
2. Contains day-to-day transaction data.	2. Contains historical data.
3. Maintains data required for business operations.	3. Maintains data required for analysis to make business decisions.

18.2 **Compare ROLAP, MOLAP and HOLAP. [*Marks:* 3]**

OR, Define the terms: OLAP (Online Analytical Processing), ROLAP (Relational OLAP) and MOLAP (Multidimensional OLAP). [*Marks:* 6]

OLAP is an approach to swiftly answer multi-dimensional analytical queries. OLAP is part of the broader category of business intelligence (a.k.a. decision-support system), which also encompasses relational reporting and data mining.

MOLAP is the *classic* form of OLAP which stores data cubes in optimized multi-dimensional array storages rather than in a relational database.

MOLAP generally delivers better performance due to specialized indexing and storage optimizations. MOLAP also needs less storage space compared to ROLAP because the specialized storage typically includes compression techniques.

ROLAP works directly with relational databases. The base data and the dimension tables are stored as relational tables and new tables are created to hold the aggregated information. This methodology relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality.

ROLAP is generally more scalable. However, large volume pre-processing is difficult to implement efficiently so it is frequently skipped. ROLAP query performance can therefore suffer tremendously. Again, since ROLAP relies more on the database to perform calculations, it has more limitations in the specialized functions it can use.

Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called HOLAP (Hybrid OLAP) systems.

HOLAP encompasses a range of solutions that attempt to mix the best of ROLAP and MOLAP. It can generally pre-process swiftly, scale well, and offer good function support.

18.3 **What is OLAP? What operations can be done using OLAP? [*2008, In-course 2008-2009. Marks:* 1 + 3]**

OLAP is an approach to swiftly answer multi-dimensional analytical queries. OLAP is part of the broader category of business intelligence (a.k.a. decision-support system), which also encompasses relational reporting and data mining.

The following operations can be done using OLAP:

1. Visualization of data as cross-tabs and data cubes.
2. Pivoting, i.e. changing the dimensions used in a cross-tab and getting responses online, within

a few seconds.

3. Displaying data at any desired level of granularity.
4. Drill down, rollup, slicing and dicing operations.

18.4 **What are the differences between Online Analytical Processing (OLAP) and Online Transaction Processing (OLTP) systems? [*Marks:* 4]**

	OLAP	OLTP
Time Scale	This stores History data for analysis.	This stores current data.
Indexing	Optimizes ad hoc queries by including lots of indices.	Optimizes update performance by minimizing the number of indices.
Normalization	Possibly partially denormalized for performance reasons as this is used for reporting.	This is fully normalized.
Organization	Data stored revolves around information topics.	Data stored revolves around business functions.
Stored Values	Stores descriptive data.	Stores typically coded data.
Homogeneity	Centralized in data warehouse.	Scattered among different databases or DBMS.
Data Present Format	Data is present in multidimension format.	Data is present in 2D format.

18.5 **What do you mean by *measure attribute* and *dimension attribute* of a given relation? [*2006. Marks:* 1]**

OR, For online analytical processing, define measure and dimension attributes using example. [*In-course 2008-2009. Marks:* 3]

Given a relation used for data analysis, we can identify some of its attributes as measure attributes, since they measure some value, and can be aggregated upon. For instance, consider the following *sales* relation:

sales(*item-name*, *color*, *size*, *number*)

The attribute *number* of the *sales* relation is a measure attribute, since it measures the number of units sold.

Some (or all) of the other attributes of the relation are identified as dimension attributes, since they define the dimensions on which measure attributes and summaries of measure attributes are viewed. In the *sales* relation, *item-name*, *color*, and *size* are dimension attributes.

18.6 **What is the multidimensional data model? [*Marks*: 1]**

Multidimensional data model is a model where data can be modeled as dimension attributes and measure attributes.

18.7 **With example define cross-tabulation/cross-tab/pivot table. [*In-course 2008-2009. Marks:* 3]**

A cross-tab is a table where values for one attribute (say *A*) form the row headers, values for another attribute (say *B*) form the column headers, and the values in an individual cell are derived as follows:

1. Each cell can be identified by (a_i, b_j), where a_i is a value for *A* and b_j a value for *B*.
2. If there is at most one tuple with any (a_i, b_j) value, the value in the cell is derived from that single tuple (if any); for instance, it could be the value of one or more other attributes of the tuple.
3. If there can be multiple tuples with an (a_i, b_j) value, the value in the cell must be derived by aggregation on the tuples with that value.

18.8 **What is data cube? [*In-course 2008-2009. Marks*: 2]**

The generalization of a cross-tab, which is 2-dimensional, to *n* dimensions can be visualized as an *n*-dimensional cube, called the *data cube*.

18.9 **Design a data cube from a multidimensional data. [*Marks*: 4]**

Consider the following relation:

sales(item-name, color, size, number)

Following are cross-tabs of *sales* by *item-name* and *color* with *size* = small and *size* = large respectively:

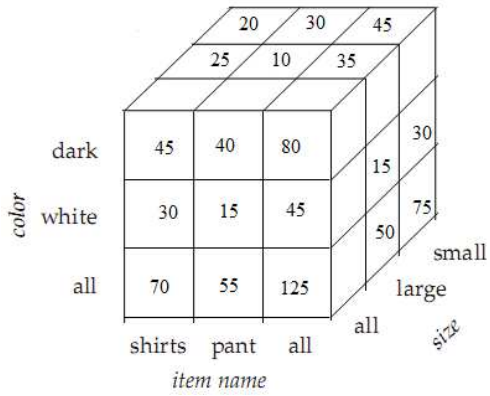
color

item-name	white	dark	Total
shirt	20	15	35
pant	10	30	40
Total	30	45	75

color

item-name	white	dark	Total
shirt	10	25	35
pant	5	10	15
Total	15	35	50

The data cube is as follows:



18.10 **Explain with suitable example the role of cross-tabulation for analyzing multidimensional data. [*2006. Marks:* 2]**

Cross-tabulation provides an easy, visual direct interpretation of multidimensional data.

For example, consider the following relation:

sales(item-name, color, size, number)

From the following cross-tab of *sales* by *item-name* and *color* with *size* = small, an analyst can easily determine which items are selling quite well and which ones are not much popular.

color

item-name	white	dark	Total
shirt	20	5	25
pant	2	30	32
Total	22	35	57

18.11 **What do you mean by concept hierarchy? [*Marks*: 2]**

A hierarchically organized collection of different levels of detail for an attribute (or concept) is called a concept hierarchy.

18.12 **Given a relation schema *sales(item-name, color, size, number)*, differentiate the *cube* and *rollup* operations. [*In-course 2008-2009. Marks*: 4]**

The cube operation computes the union of all possible different groupings of the relation, whereas rollup generates aggregates at multiple levels of a hierarchy.

For example, a cube operation on the sales relation is as follows:

select *item-name, color, size*, **sum**(*number*)
from *sales*

group by cube(*item-name, color, size*)

This query computes the union of eight different groupings of the sales relation:

{(*item-name, color, size*), (*item-name, color*), (*item-name, size*), (*color, size*), (*item-name*), (*color*), (*size*), ()}

A representative rollup construct is

select *item-name, color, size*, **sum**(*number*)
from *sales*
group by rollup(*item-name, color, size*)

Here, only four groupings are generated:

{(*item-name, color, size*), (*item-name, color*), (*item-name*), ()}

18.13 **Give an example of a pair of grouping that cannot be expressed by using a single `group by` clause with `cube` and `rollup`. [*Marks:* 6]**

Consider an example of hierarchies on dimensions in the figure beside.

We cannot express a query to seek aggregation on groups (*City, Hour of day*) and (*City, Date*) using a single **group by** clause with **cube** and **rollup**.

Any single **group by** clause with **cube** and **rollup** that computes these two groups would also compute other groups also.

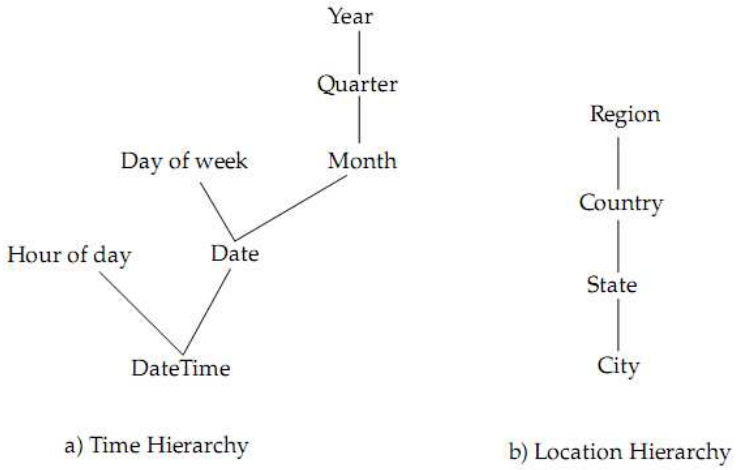


Figure Hierarchies on dimensions.

18.14 **Show how to express `group by cube`(*a, b, c, d*) using `rollup`; your answer should have only one `group by` clause. [*2008, In-course 2008-2009. Marks: 2*]**

group by rollup(*a*), rollup(*b*), rollup(*c*), rollup(*d*)

18.15 **Given a relation *S*(*Student, Subject, Marks*), write a query to rank the top *n* students based on the total marks by using ranking. [*2008, In-course 2008-2009. Marks:* 2]**

We assume that multiple students do not have the same marks since otherwise the question is not deterministic; the query below deterministically returns all students with the same marks as the *n* student, so it may return more than *n* students.

select *student*, **sum**(*marks*) **as** *total*, **rank() over (order by (***total***) desc) as** *t_rank*
from *S*
group by *student*
having *t_rank* <= *n*

18.16 **Write a query to find cumulative balances, equivalent to that shown below, but without using the extended SQL windowing constructs.**

select *account_number, date_time,*
sum(*value*) **over**
(partition by *account_number*
order by *date_time*
Rows unbounded preceding)
as *balance*
from *transaction*
order by *account_number, date_time*

[The query gives the cumulative balances on each account just before each transaction on the account; the cumulative balance of the account is the sum of values of all earlier transactions

on the account.]

```
select t1.account-number, t1.date-time, sum(t2.value)
from transaction as t1, transaction as t2
where t1.account-number = t2.account-number and t2.date-time < t1.date-time
group by t1.account-number, t1.date-time
order by t1.account-number, t1.date-time
```

18.17 **Consider the *sales* relation below.**

sales(item-name, color, size, number)

Write an SQL query to compute the *cube* operation on the relation, giving the relation in the figure beside. Do not use the *with cube* construct.

```
(select color, size, sum(number)
 from sales
 group by color, size
)
union
(select color, 'all', sum(number)
 from sales
 group by color
)
union
(select 'all', size, sum(number)
 from sales
 group by size
)
union
(select 'all', 'all', sum(number)
 from sales
)
```

item-name	*color*	*number*
skirt	dark	8
skirt	pastel	35
skirt	white	10
skirt	**all**	53
dress	dark	20
dress	pastel	10
dress	white	5
dress	**all**	35
shirt	dark	14
shirt	pastel	7
shirt	white	28
shirt	**all**	49
pant	dark	20
pant	pastel	2
pant	white	5
pant	**all**	27
all	dark	62
all	pastel	54
all	white	48
all	**all**	164

Data Warehouse

18.18 **What is a data warehouse? [*2008, In-course 2008-2009. Marks:* 1]**

A data warehouse is a repository (or archive) of information gathered from multiple sources, stored under a unified schema, at a single site.

18.19 **Sketch the data warehouse architecture. [*2008. Marks:* 1]**

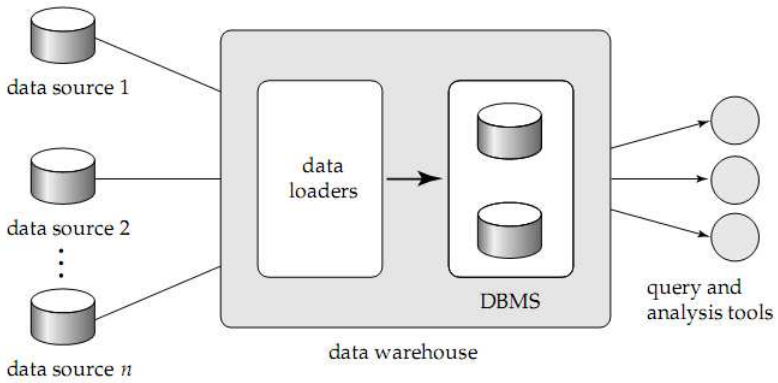


Figure Data-warehouse architecture.

18.20 **What are the components of a data warehouse? [*2006, Marks: 1. In-course 2008-2009, Marks:* 5]**

Followings are the components of a data warehouse[1]:

- **Database Servers:** Operational data accumulated during standard business must be extracted and stored into a database. Most companies use a relational database stored on a mainframe server.
- **Queries/Reports:** Querying is a broad term that encompasses all the activities of requesting data from a data warehouse for analysis. Reports are then generated to display the results for the specified query. Querying, obviously, is the whole point of using the data warehouse.
- **OLAP/Multi-dimensional analysis:** Relational databases store data in a two dimensional format; tables of data represented by rows and columns. Multi-dimensional analysis, commonly referred to as On-Line Analytical Processing (OLAP), offer an extension to the relational model to provide a multi-dimensional view of the data. These tools allow users to drill down from summary data sets into the specific data underlying the summaries. Statistical analysis tools provide summary information and help determine the degree of relationship between two factors.
- **Data Mining:** Data mining is the process of analyzing business data in the data warehouse to find unknown patterns or rules of information that one can use to tailor business operations. Data mining predicts future trends and behaviors, allowing businesses to make proactive, knowledge driven decisions.

18.21 **What are the features of a data warehouse? [*Marks:* 3]**

A data warehouse has the following features[2]:

- **Subject-oriented**

  In operational database, data is stored by individual applications. In the data sets for an order processing application, the data for that particular application is kept. These data sets provide the data for all the functions for entering orders, checking stock, verifying customer's credit, and assigning the order for shipment. But these data sets contain only the data that is needed for those functions relating to this particular application.

  In striking contrast, in the data warehouse, data is stored by business subjects, not by applications. Business subjects differ from enterprise to enterprise. These are the subjects critical for the enterprise. For a manufacturing company, sales, shipments, and inventory are critical business subjects. For a retail store, sales at the check-out counter is a critical subject.

- **Integrated**

  For proper decision making, all the relevant data from the various applications need to be pulled together. The data in the data warehouse come from several operational systems.

- **Time-variant**

  For an operational system, the stored data contains the current values. A data warehouse, because of the very nature of its purpose – which is analysis and decision making, has to contain historical data, not just current values.

- **Nonvolatile**

  In the data warehouse, the data are kept as snapshots over time. The data warehouse is not updated every time a single order is processed. Data from the operational systems are moved into the data warehouse at specific intervals.

[1] **Collected From:** http://www.pharmacyinformatics.org/datawarehouse.htm#comp [Last accessed: 30-08-2010 07:15pm]

[2] **Collected From:** http://www.datawarehousesolution.net/defining-features-of-data-warehouse/Data_Warehouse_for_Beginners [Last accessed: 30-08-2010 07:15pm]

- **Data Granularity**

  Data granularity in a data warehouse refers to the level of detail. In a data warehouse, data are kept summarized at different levels. Depending on the query, a user can then go to the particular level of detail and satisfy the query.

18.22 **Describe benefits and drawbacks of a source-driven architecture for gathering of data at a data warehouse as compared to destination-driven architecture. [*In-course 2008-2009. Marks:* 3]**

In a destination-driven architecture for gathering data, data transfers from the data sources to the data-warehouse are based on demand from the warehouse, whereas in a source-driven architecture, the transfers are initiated by each source.

The benefits of a source-driven architecture are:

- Data can be propagated to the destination as soon as it becomes available. For a destination-driven architecture to collect data as soon as it is available, the warehouse would have to probe the sources frequently, leading to a high overhead.
- The source does not have to keep historical information. As soon as data is updated, the source can send an update message to the destination and forget the history of the updates. In contrast, in a destination-driven architecture, each source has to maintain a history of data which have not yet been collected by the data warehouse. Thus storage requirements at the source are lower for a source-driven architecture.

On the other hand, a destination-driven architecture has the following advantages:

- In a source-driven architecture, the source has to be active and must handle error conditions such as not being able to contact the warehouse for some time. It is easier to implement passive sources, and a single active warehouse. In a destination-driven architecture, each source is required to provide only a basic functionality of executing queries.
- The warehouse has more control on when to carry out data gathering activities, and when to process user queries; it is not a good idea to perform both simultaneously, since they may conflict on locks.

18.23 **Describe the steps of building a data warehouse. [*Marks:* 5]**

The major steps in developing a data warehouse are[3]:

1. **Identify the data source:**

   The very first step before starting to develop a data warehouse, the data source should be identified. The data that are required to be put into the data warehouse need to be figured out.

2. **Build customized ETL[4] tool**

   Each data warehouse has different requirements. Therefore, a customized ETL tool is the better solution in order to fulfill the requirements.

3. **Extraction**

   This can be the most time consuming part where the data from various data source need to be grabbed and store into the staging database. Much of the time and effort are needed in writing a custom program to transfer the data from sources into staging database. As a result, during extraction, the database system that will be used for the staging area needs to be determined and also the necessary data that are needed needs to be figured out before grabbing them.

[3] **Collected From:** http://www.dotnetspider.com/attachments/Resources/38797-5129-term-paper-on-New-Trends-in-data-warehousing.pdf [Last accessed: 28-08-2010 10:45pm]

[4] **ETL Tool:** The different steps involved in getting data into a data warehouse are called as extract, transform and load or ETL tasks; extraction refers to getting data from the sources, while load refers to loading the data into the data warehouse.

4. Transformation

After extracting the data from various data sources, transformation is needed to ensure the data consistency. In order to transform the data into data warehouse properly, a way of mapping the external data sources fields to the data warehouse fields needs to be figured out. Transformation can be performed during data extraction or while loading the data into data warehouse.

5. Loading

Once the extracting process, transforming and cleansing has been done, the data are loaded into the data warehouse. The loading of data can be categorized into two types: the loading of data that currently contain in the operational database and the loading of the updates to the data warehouse from the changes that have occurred in the operational database.

18.24 **Describe the functions a user can perform on a data warehouse. [*Marks:* 3]**

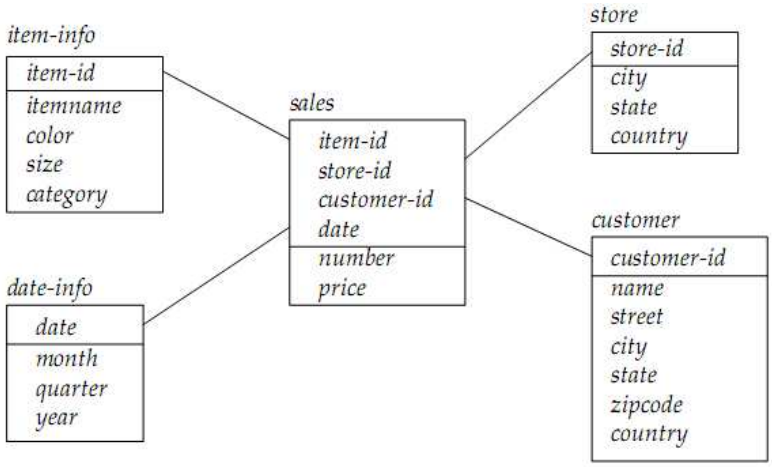
18.25 **Distinguish between**

i. Fact Table and Dimension Table
ii. Star Schema and Snow-Flake Schema [*2006. Marks:* 3]

i. Tables containing multidimensional data are called *fact tables*.

To minimize storage requirements, dimension attributes are usually short identifiers that are foreign keys into other tables called *dimension tables*.

For example, a fact table *sales* would have dimension attributes *item-id*, *store-id*, *customer-id* and *date*, and measure attributes *number* and *price*. The attribute *store-id* is a foreign key into a dimension table *store*, which has other attributes such as store location (*city*, *state*, *country*).



ii. A schema with a fact table, multiple dimension tables, and foreign keys from the fact table to the dimension tables, is called a *star schema*.

More complex data warehouse designs may have multiple levels of dimension tables; for instance, the item-info table may have an attribute manufacturer-id that is a foreign key into another table giving details of the manufacturer. Such schemas are called *snowflake schemas*.

18.26 **What are the schemas used for data warehouse? Give example of any one of them. [*Marks:* 2 + 2]**

The schemas used for data warehouse are:

1. Star Schema
2. Snowflake Schema

Example of star schema:

[*See the figure in question 18.25*]

Data Mining

18.27 **What is data mining? [*In-course 2008-2009. Marks*: 1]**

The term data mining refers loosely to the process of semi-automatically analyzing large databases to find useful patterns.

Like knowledge discovery in artificial intelligence (also called machine learning), or statistical analysis, data mining attempts to discover rules and patterns from data.

18.28 **What are the applications of data mining techniques? [*Marks*: 4]**

OR, What are the major areas of data mining? [*Marks*: 2]

The applications / major areas of data mining are as follows:

1. The most widely used applications are those that require some sort of prediction.

   For instance, when a person applies for a credit card, the credit-card company wants to predict if the person is a good credit risk. The prediction is to be based on known attributes of the person, such as age, income, debts, and past debt repayment history. Rules for making the prediction are derived from the same attributes of past and current credit card holders, along with their observed behavior, such as whether they defaulted on their credit-card dues.

   Other types of prediction include predicting which customers may switch over to a competitor (these customers may be offered special discounts to tempt them not to switch), predicting which people are likely to respond to promotional mail ("junk mail"), or predicting what types of phone calling card usage are likely to be fraudulent.

2. Another class of applications looks for associations.

   For instance, books those tend to be bought together. If a customer buys a book, an online bookstore may suggest other associated books. If a person buys a camera, the system may suggest accessories that tend to be bought along with cameras.

   Other types of associations may lead to discovery of causation. For instance, discovery of unexpected associations between a newly introduced medicine and cardiac problems led to the finding that the medicine may cause cardiac problems in some people. The medicine was then withdrawn from the market.

18.29 **Explain a situation when data mining may be useful. [*2007. Marks:* 2]**

For instance, when a person applies for a credit card, the credit-card company wants to predict if the person is a good credit risk. The prediction is to be based on known attributes of the person, such as age, income, debts, and past debt repayment history. Rules for making the prediction are derived from the same attributes of past and current credit card holders, along with their observed behavior, such as whether they defaulted on their credit-card dues. Data mining would be useful to automate this rules derivation process.

18.30 **Define *support* and *confidence* of association rules in data mining with suitable example. [*2006, In-course 2008-2009. Marks*: 3]**

Support is a measure of what fraction of the population satisfies both the antecedent and the consequent of the rule. For instance, suppose only 0.001 percent of all purchases include milk and screwdrivers. Thus, the support for the rule *milk* $\Rightarrow$ *screwdrivers* is low.

Confidence is a measure of how often the consequent is true when the antecedent is true. For instance, the rule *bread* $\Rightarrow$ *milk* has a confidence of 80% if 80% of the purchases that include bread also include milk.

18.31 **Define minimum support (*minsup*) and minimum confidence (*minconf*). [*Marks*: 3]**

An itemset *A* satisfies a *minimum support* if its frequency count is greater than or equal to a specific support count, *min_sup*.

For example, suppose *min_sup* is 50%.

If $sup(A, C) = 50\% \geq 50\%$, (A, C) is frequent
If $sup(A, D) = 25\% \leq 50\%$, (A, D) is non-frequent

Similarly, an itemset *A* satisfies a *minimum confidence* if its frequency count is greater than or equal to a specific confidence count, *min_conf*.

18.32 **What is the process of finding association rules using apriori algorithm? [*Marks*: 8]**

There are two steps for finding association rules using apriori algorithm:

1. Find all frequent itemsets – the sets of item that have minimum support.
2. Generate strong association rules from the frequent itemsets found that have minimum confidence.

Apriori performs a level-wise iterative search to find frequent itemset L_k from L_{k-1} where L_i is a frequent itemset having *i* items. While finding a frequent itemset, the apriori property is maintained, that is, all non-empty subsets of a frequent itemset must also be frequent. Thus, the first step itself is divided into two steps:

1. **Join Step:** C_k is generated by joining $L_{k-1}$with itself
2. **Prune Step:** Any $(k - 1)$-itemset that is not frequent cannot be a subset of a frequent *k*-itemset

The pseudocode is as follows:

C_k: Candidate itemset of size *k*
L_k : frequent itemset of size *k*

L_1 = {frequent 1-itemset};
for (k = 1; L_k != ∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction *t* in database **do**
    increment the count of all candidates in C_{k+1} that are contained in *t*;
  end
  L_{k+1} = candidates in C_{k+1} with min_support;
end
return $\cup_k L_k$;

The process of finding association rules is as follows:

1. For each frequent itemset *I*, generate all nonempty subsets of *I*.
2. For each non-empty subset *S* of *I*, output the rule $S \Rightarrow (I - S)$ if support_count(*I*) / support_count(*S*) ≥ minimum_confidence.

18.33 **What are the major characteristics and limitations of Apriori algorithm in connection with association rule mining? [*In-course 2008-2009. Marks:* 5]**

Major characteristics of Apriori algorithm:

1. Uses frequent $(k - 1)$-itemsets to generate candidate *k*-itemsets.
2. Uses database scan and pattern matching to collect counts for the candidate itemsets.

Major limitations of Apriori algorithm:

1. Generates huge candidate sets. For example, 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets. Again, to discover a frequent pattern of size 100, e.g. $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
2. Needs multiple scans of database. Specifically, needs $(n + 1)$ scans, where *n* is the length of the longest pattern.

18.34 **Suppose half of all the transactions in a clothes shop purchase jeans, and one-third of all transactions in the shop purchase T-shirts. Suppose also that half of the transactions that purchase jeans also purchase T-shirts. Write down all the (nontrivial) association rules you can deduce from the above information, giving support and confidence of each rule. [*2007. Marks:* 4]**

The rules are as follows. The last rule can be deduced from the previous ones.

Rule	Support	Confidence
$\forall$ *transactions T,* **true** $\Rightarrow$ *buys*(*T, jeans*)	50%	50%
$\forall$ *transactions T,* **true** $\Rightarrow$ *buys*(*T, t-shirts*)	33%	33%
$\forall$ *transactions T, buys*(*T, jeans*) $\Rightarrow$ *buys*(*T, t-shirts*)	25%	50%
$\forall$ *transactions T, buys*(*T, t-shirts*) $\Rightarrow$ *buys*(*T, jeans*)	25%	75%

18.35 **Consider the following transactional data set:**

TID	List of Items
T100	**I1, I2, I5**
T200	**I2, I4**
T300	**I2, I3**
T400	**I1, I2, I4**
T500	**I1, I3**
T600	**I2, I3**
T700	**I1, I3**
T800	**I1, I2, I3, I5**
T900	**I1, I2, I3**

i) Use Apriori algorithm to find the frequent itemsets with their support count when minimum support count is 2 (i.e. *minsup* = 2/9 = 22%).

ii) Generate the association rules with their corresponding confidences. [*Marks:* 12 + 4]

i) Generating 1-itemset frequent pattern:

C_1

Item Set	Supp. Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

Item Set	Supp. Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Generating 2-itemset frequent pattern:

C_2

Item Set	Supp. Count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2

Item Set	Supp. Count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

Generating 3-itemset frequent pattern:

C_3

Item Set	Supp. Count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

Item Set	Supp. Count
{I1, I2, I3}	2
{I1, I2, I5}	2

Generating 4-itemset frequent pattern: C4 = φ

∴ $L_1 \cup L_2 \cup L_3$ is the frequent itemset with minimum support count of 2.

ii) Taking {I1, I2, I5}, its nonempty subsets are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, {I5}.

The resulting association rules with corresponding confidences are as follows:

Association Rule	**Confidence**
I1 ∧ I2 ⇒ I5	sc{I1, I2, I5} / sc{I1, I2} = 2 / 4 = 50%
I1 ∧ I5 ⇒ I2	2 / 2 = 100%
I2 ∧ I5 ⇒ I1	2 / 2 = 100%
I1 ⇒ I2 ∧ I5	2 / 6 = 33%
I2 ⇒ I1 ∧ I5	2 / 7 = 29%
I5 ⇒ I1 ∧ I2	2 / 2 = 100%

Taking {I1, I2, I3}, its nonempty subsets are {I1, I2}, {I1, I3}, {I2, I3}, {I1}, {I2}, {I3}.

The resulting association rules with corresponding confidences are as follows:

Association Rule	**Confidence**
I1 ∧ I2 ⇒ I3	2 / 4 = 50%
I1 ∧ I3 ⇒ I2	2 / 4 = 50%
I2 ∧ I3 ⇒ I1	2 / 4 = 50%
I1 ⇒ I2 ∧ I3	2 / 6 = 33%
I2 ⇒ I1 ∧ I3	2 / 7 = 29%
I3 ⇒ I1 ∧ I2	2 / 6 = 33%

Similarly, for the rest of the itemsets,

Taking	**Association Rule**	**Confidence**
{I1, I2}	I1 ⇒ I2	4 / 6 = 67%
	I2 ⇒ I1	4 / 7 = 57%
{I1, I3}	I1 ⇒ I3	4 / 6 = 67%
	I3 ⇒ I1	4 / 6 = 67%
{I1, I5}	I1 ⇒ I5	2 / 6 = 33%
	I5 ⇒ I1	2 / 2 = 100%
{I2, I3}	I2 ⇒ I3	4 / 7 = 57%
	I3 ⇒ I2	4 / 6 = 67%
{I2, I4}	I2 ⇒ I4	2 / 7 = 29%
	I4 ⇒ I2	2 / 2 = 100%
{I2, I5}	I2 ⇒ I5	2 / 7 = 29%
	I5 ⇒ I2	2 / 2 = 100%

18.36 **Apply the Apriori algorithm to the following data set:**

TID	Items Purchased
101	**milk, bread, eggs**
102	**milk, juice**
103	**juice, butter**
104	**milk, bread, eggs**
105	**coffee, eggs**
106	**coffee**
107	**coffee, juice**
108	**milk, bread, cookies, eggs**
109	**cookies, butter**
110	**milk, bread**

The set of items is { milk, bread, cookies, eggs, butter, coffee, juice }.

Use 0.2 for the minimum support value. [*Marks:* 6]

[*Similar to 18.35. Try yourself…* ☺]

18.37 **What is decision tree? [*Marks:* 2]**

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

CHAPTER 20
DATABASE SYSTEM ARCHITECTURES

20.1 **Describe the transaction server process structure.**

A typical transaction server system today consists of multiple processes accessing data in shared memory, as in the figure. The processes that form part of the database system include

- **Server processes:** These are processes that receive user queries (transactions), execute them, and send the results back.
- **Lock manager process**: This process implements lock manager functionality, which includes lock grant, lock release, and deadlock detection.
- **Database writer process:** There are one or more processes that output modified buffer blocks back to disk on a continuous basis.
- **Log writer process:** This process outputs log records from the log record buffer to stable storage.
- **Checkpoint process:** This process performs periodic checkpoints.
- **Process monitor process:** This process monitors other processes, and if any of them fails, it takes recovery actions for the process, such as aborting any transaction being executed by the failed process, and then restarting the process.

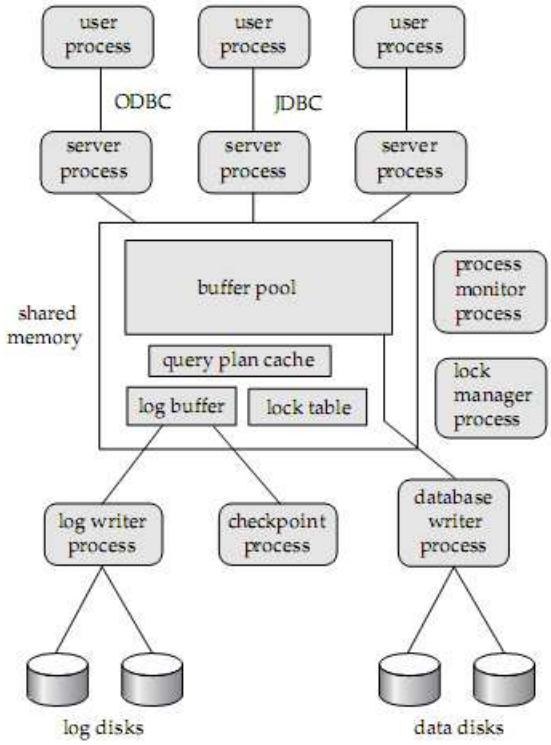


Figure 18.4 Shared memory and process structure.

The shared memory contains all shared data, such as:

- Buffer pool
- Lock table
- Log buffer, containing log records waiting to be output to the log on stable storage
- Cached query plans, which can be reused if the same query is submitted again

20.2 **Define speedup and scaleup with respect to parallel database system. What are the factors that affect parallel operations as well as speedup and scaleup? [*2008, In-course 2008-2009. Marks: 3 + 2*]**

Running a given task in less time by increasing the degree of parallelism is called *speedup*. For example, if the original system took 60 seconds to perform a task, and two parallel systems took 30 seconds, then the value of speedup would be equal to 2.

Handling larger tasks by increasing the degree of parallelism is called *scaleup*. For example, if the original system can process 100 transactions in a given amount of time, and the parallel system can process 200 transactions in this amount of time, then the value of scaleup would be equal to 2.[5]

Factors that affect parallel operations as well as speedup and scaleup:

1. Startup costs
2. Interference
3. Skew

[5] Improved response time is *speedup*, and enhanced throughput is *scaleup*.

20.3 **Mention several architecture models for parallel machines.**

- **Shared memory –** All the processors share a common memory (Figure 18.8a).
- **Shared disk –** All the processors share a common set of disks (Figure 18.8b).
- **Shared nothing –** The processors share neither a common memory nor common disk (Figure 18.8c).
- **Hierarchical –** This model is a hybrid of the preceding three architectures (Figure 18.8d).

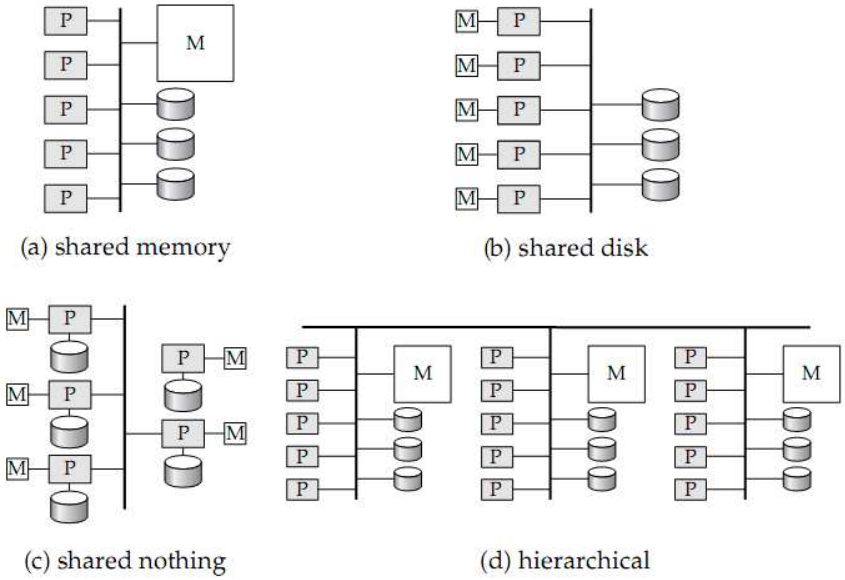


Figure 18.8 Parallel database architectures.

20.4 **Mention the characteristics of shared-memory parallel database architecture.**

In shared-memory architecture, the processors and disks have access to a common memory, typically via a bus or through an interconnection network.

Benefit:

Extremely efficient communication between processors.

Drawbacks:

1. The architecture is not scalable beyond 32 or 64 processors because the bus or the interconnection network becomes a bottleneck (since it is shared by all processors).
2. Maintaining cache-coherency becomes an increasing overhead with increasing number of processors.

20.5 **Mention the characteristics of shared-disk parallel database architecture. [*2008. Marks: 2*]**

In the shared-disk model, all processors can access all disks directly via an interconnection network, but the processors have private memories.

Advantages:

1. Since each processor has its own memory, the memory bus is not a bottleneck.
2. It offers a cheap way to provide a degree of fault tolerance: If a processor (or its memory) fails, the other processors can take over its tasks, since the database is resident on disks that are accessible from all processors.

Disadvantage:

Scalability – although the memory bus is no longer a bottleneck, the interconnection to the disk subsystem is now a bottleneck.

20.6 **Mention the characteristics of shared-nothing parallel database architectures.**

In a shared-nothing system, each node of the machine consists of a processor, memory, and one or more disks. The processors at one node may communicate with another processor at another node by a high-speed interconnection network.

Advantages:

1. Since local disk references are serviced by local disks at each processor, the shared-nothing model overcomes the disadvantage of requiring all I/O to go through a single interconnection network; only queries, accesses to nonlocal disks, and result relations pass through the network.
2. The interconnection networks for shared-nothing systems are usually designed to be scalable, so that their transmission capacity increases as more nodes are added. Consequently, shared-nothing architectures are more scalable and can easily support a large number of processors.

Disadvantages:

1. High costs of communication.
2. High costs of nonlocal disk access, which are higher than in shared-memory or shared-disk architecture since sending data involves software interaction at both ends.

20.7 **Compare different parallel database architectures. [*In-course 2008-2009. Marks: 7*]**

1. Architecture:

In shared-memory architecture, the processors and disks have access to a common memory, typically via a bus or through an interconnection network.

In the shared-disk model, all processors can access all disks directly via an interconnection network, but the processors have private memories.

In a shared-nothing system, each node of the machine consists of a processor, memory, and one or more disks. The processors at one node may communicate with another processor at another node by a high-speed interconnection network.

2. Scalability:

Shared-memory architecture is not scalable beyond 32 or 64 processors because the bus or the interconnection network becomes a bottleneck (since it is shared by all processors).

In shared-disk model, although the memory bus is no longer a bottleneck, the interconnection to the disk subsystem becomes a bottleneck.

The shared-nothing architecture is scalable. However, the communication between nodes and nonlocal disk access costs are very high.

20.8 **What are the advantages and disadvantages of distributed database system? [*2008. Marks: 3*]**

Advantages:

1. **Sharing data** – Users at one site may be able to access the data residing at other sites.
2. **Autonomy** – Each site is able to retain a degree of control over data that are stored locally.
3. **Availability** – If one site fails in a distributed system, the remaining sites may be able to continue operating if data items are replicated in several sites.

Disadvantages:

1. Added complexity (such as more costly software development, greater potential for bugs, increasing processing overhead to achieve intersite coordination etc.) required to ensure proper coordination among the sites.
2. Recovery from failure is more complex in distributed systems than in centralized systems.

20.9 **Why is it relatively easy to port a database from a single processor machine to a multiprocessor machine if individual queries need not be parallelized?**

Porting is relatively easy to a shared memory multiprocessor machine. Databases designed for single-processor machines already provide multitasking, allowing multiple processes to run on the same processor in a time-shared manner, giving a view to the user of multiple processes running in parallel. Thus, coarse-granularity parallel machines logically appear to be identical to single-processor machines, making the porting relatively easy.

Porting a database to a shared disk or shared nothing multiprocessor architecture is a little harder.

20.10 **Transaction server architectures are popular for client-server relational databases, where transactions are short. On the other hand, data server architectures are popular for client-server object-oriented database systems, where transactions are expected to be relatively long. Give two reasons why data servers may be popular for object-oriented databases but not for relational databases.**

Data servers are good if data transfer is small with respect to computation, which is often the case in applications of OODBs such as computer aided design. In contrast, in typical relational database applications such as transaction processing, a transaction performs little computation but may touch several pages, which will result in a lot of data transfer with little benefit in a data server architecture.

Another reason is that structures such as indices are heavily used in relational databases, and will become spots of contention in data server architecture, requiring frequent data transfer. There are no such points of frequent contention in typical current-day OODB applications such as computer aided design.

20.11 **Instead of storing shared structures in shared memory, an alternative architecture would be to store them in the local memory of a special process, and access the shared data by interprocess communication with the process. What would be the drawback of such an architecture?**

The drawbacks would be:

1. Two interprocess messages would be required to acquire locks, one for the request and one to confirm grant. Interprocess communication is much more expensive than memory access, so the cost of locking would increase.
2. The process storing the shared structures could also become a bottleneck.

20.12 **What are the factors that can work against linear scaleup in a transaction processing system? Which of the factors are likely to be the most important in each of the following architectures: shared memory, shared disk, and shared nothing?**

Increasing contention for shared resources prevents linear scale-up with increasing parallelism.

In a shared memory system, contention for memory (which implies bus contention) will result in falling scale-up with increasing parallelism.

In a shared disk system, it is contention for disk and bus access which affects scale-up.

In a shared-nothing system, inter-process communication overheads will be the main impeding factor. Since there is no shared memory, acquiring locks, and other activities requiring message passing between processes will take more time with increased parallelism.

CHAPTER 21
PARALLEL DATABASES

21.1 **Discuss three basic data partitioning strategies. [*2008. Marks: 4*]**

The three basic data partitioning strategies are as follows, assuming that there are n disks, D_0, D_1, …, D_{n-1}, across which the data are to be partitioned.

1. Round-Robin:

   This strategy scans the relation in any order and sends the i-th tuple to disk number $D_{i \bmod n}$. The round-robin scheme ensures an even distribution of tuples across disks.

2. Hash partitioning:

   This declustering strategy designates one or more attributes from the given relation's schema as the partitioning attributes. A hash function is chosen whose range is {0, 1, . . . , n - 1}. Each tuple of the original relation is hashed on the partitioning attributes. If the hash function returns i, then the tuple is placed on disk D_i.

3. Range partitioning:

   This strategy distributes contiguous attribute-value ranges to each disk. It chooses a partitioning attribute, A, as a partitioning vector. The relation is partitioned as follows. Let $[v_0, v_1, \ldots, v_{n-2}]$ denote the partitioning vector, such that, if $i < j$, then $v_i < v_j$. Consider a tuple t such that $t[A] = x$. If $x < v_0$, then t goes on disk D_0. If $x \geq v_{n-2}$, then t goes on disk D_{n-1}. If $v_i \leq x < v_{i+1}$, then t goes on disk D_{i+1}.

21.2 **For each of the three partitioning techniques, namely round-robin, hash partitioning, and range partitioning, give an example of a query for which that partitioning technique would provide the fastest response.**

Round robin partitioning: When relations are large and queries read entire relations, round-robin gives good speed-up and fast response time.

Hash partitioning: For point queries, this gives the fastest response, as each disk can process a query simultaneously. If the hash partitioning is uniform, even entire relation scans can be performed efficiently.

Range partitioning: For range queries which access a few tuples, this gives fast response.

21.3 **In a range selection on a range-partitioned attribute, it is possible that only one disk may need to be accessed. Describe the benefits and drawbacks of this property.**

If there are few tuples in the queried range, then each query can be processed quickly on a single disk. This allows parallel execution of queries with reduced overhead of initiating queries on multiple disks.

On the other hand, if there are many tuples in the queried range, each query takes a long time to execute as there is no parallelism within its execution. Also, some of the disks can become hot-spots, further increasing response time.

21.4 **What factors could result in skew when a relation is partitioned on one of its attributes by:**

a. Hash partitioning
b. Range partitioning

In each case, what can be done to reduce the skew? [*2008, 2007. Marks: 3*]

a. Hash-partitioning:

   Too many records with the same value for the hashing attribute, or a poorly chosen hash function without the properties of randomness and uniformity, can result in a skewed partition.

To improve the situation, we should experiment with better hashing functions for that relation.

b. Range-partitioning:

Non-uniform distribution of values for the partitioning attribute (including duplicate values for the partitioning attribute) which are not taken into account by a bad partitioning vector is the main reason for skewed partitions.

Sorting the relation on the partitioning attribute and then dividing it into *n* ranges with equal number of tuples per range will give a good partitioning vector with very low skew.

21.5 **What form of parallelism (interquery, interoperation, or intraoperation) is likely to be the most important for each of the following tasks:**

a. Increasing the throughput of a system with many small queries
b. Increasing the throughput of a system with a few large queries, when the number of disks and processors is large [*2007. Marks: 4*]

a. When there are many small queries, inter-query parallelism gives good throughput. Parallelizing each of these small queries would increase the initiation overhead, without any significant reduction in response time.

b. With a few large queries, intra-query parallelism is essential to get fast response times. Given that there are large number of processors and disks, only intra-operation parallelism can take advantage of the parallel hardware – for queries typically have few operations, but each one needs to process a large number of tuples.

21.6 **With pipelined parallelism, it is often a good idea to perform several operations in a pipeline on a single processor, even when many processors are available.**

a. Explain why.
b. Would the arguments you advanced in part a hold if the machine has a shared-memory architecture? Explain why or why not.
c. Would the arguments in part a hold with independent parallelism? (That is, are there cases where, even if the operations are not pipelined and there are many processors available, it is still a good idea to perform several operations on the same processor?)

a. The speed-up obtained by parallelizing the operations would be offset by the data transfer overhead, as each tuple produced by an operator would have to be transferred to its consumer, which is running on a different processor.

b. In shared-memory architecture, transferring the tuples is very efficient. So the above argument does not hold to any significant degree.

c. Even if two operations are independent, it may be that they both supply their outputs to a common third operator. In that case, running all three on the same processor may be better than transferring tuples across processors.

21.7 **Give two examples of a join that is not a simple equi-join for which partitioned parallelism can be used. What attributes should be used for partitioning?**

a. $r \bowtie_{(r.A = s.B) \wedge (r.A < s.C)} s$

Here we have extra conditions which can be checked after the join. Hence partitioned parallelism is useful.

b. $r \bowtie_{(r.A \geq s.B * 20) \wedge (r.A < s.B + 10)} s$

This is a query in which an *r* tuple and an *s* tuple join with each other if they fall into the same range of values. Hence partitioned parallelism applies naturally to this scenario.

For both the queries, *r* should be partitioned on attribute *A* and *s* on attribute *B.*

21.8	**Suppose you have a histogram where values are between 1 and 100, and are partitioned into 10 ranges, 1– 10, 11–20, ..., 91– 100, with frequencies 15, 5, 20, 10, 10, 5, 5, 20, 5, and 5, respectively. Give a load-balanced range partitioning function to divide the values into 5 partitions. [*2007. Marks: 3*]** A partitioning vector which gives 5 partitions with 20 tuples in each partition is: [21, 31, 51, 76]. The 5 partitions obtained are 1 - 20, 21 - 30, 31 - 50, 51 - 75 and 76 - 100. The assumption made in arriving at this partitioning vector is that within a histogram range, each value is equally likely.
21.9	**Describe the benefits and drawbacks of pipelined parallelism.** **Benefit:** No need to write intermediate relations to disk only to read them back immediately. **Drawbacks:** 1. Cannot take advantage of high degrees of parallelism, as typical queries do not have large number of operations. 2. Not possible to pipeline operators which need to look at all the input before producing any output. 3. Since each operation executes on a single processor, the most expensive ones take a long time to finish. Thus speed-up will be low in spite of parallelism.
21.10	**Some parallel database systems store an extra copy of each data item on disks attached to a different processor, to avoid loss of data if one of the processors fails. Why is it a good idea to partition the copies of the data items of a processor across multiple processors?** The copies of the data items at a processor should be partitioned across multiple other processors, rather than stored in a single processor, for the following reasons: 1. To better distribute the work that should have been done by the failed processor, among the remaining processors. 2. Even when there is no failure, this technique can to some extent deal with hot-spots created by read only transactions.

CHAPTER 22
DISTRIBUTED DATABASES

22.1 **Discuss the relative advantages of centralized and distributed databases.**

A distributed database allows a user convenient and transparent access to data which is not stored at the site, while allowing each site control over its own local data. A distributed database can be made more reliable than a centralized system because if one site fails, the database can continue functioning, but if the centralized system fails, the database can no longer continue with its normal operation. Also, a distributed database allows parallel execution of queries and possibly splitting one query into many parts to increase throughput.

A centralized system is easier to design and implement. A centralized system is cheaper to operate because messages do not have to be sent.

22.2 **Explain how the followings differ: fragmentation transparency, replication transparency, and location transparency.**

With fragmentation transparency, the user of the system is unaware of any fragmentation the system has implemented. A user may formulate queries against global relations and the system will perform the necessary transformation to generate correct output.

With replication transparency, the user is unaware of any replicated data. The system must prevent inconsistent operations on the data. This requires more complex concurrency control algorithms.

Location transparency means the user is unaware of where data are stored. The system must route data requests to the appropriate sites.

22.3 **How might a distributed database designed for a local-area network differ from one designed for a wide-area network?**

Data transfer on a local-area network (LAN) is much faster than on a wide-area network (WAN). Thus replication and fragmentation will not increase throughput and speed-up on a LAN, as much as in a WAN. But even in a LAN, replication has its uses in increasing reliability and availability.

22.4 **When is it useful to have replication or fragmentation of data? Explain your answer.**

Replication is useful when there are many read-only transactions at different sites wanting access to the same data. They can all execute quickly in parallel, accessing local data. But updates become difficult with replication.

Fragmentation is useful if transactions on different sites tend to access different parts of the database.

22.5 **Explain the notions of transparency and autonomy. Why are these notions desirable from a human-factors standpoint?**

Autonomy is the amount of control a single site has over the local database. It is important because users at that site want quick and correct access to local data items. This is especially true when one considers that local data will be most frequently accessed in a database. Transparency hides the distributed nature of the database. This is important because users should not be required to know about location, replication, fragmentation or other implementation aspects of the database.

22.6 **To build a highly available distributed system, you must know what kinds of failures can occur.**

a. List possible types of failure in a distributed system.
b. Which items in your list from part a are also applicable to a centralized system?

a. The types of failure that can occur in a distributed system include

i. Computer failure (site failure).

	ii. Disk failure. iii. Communication failure. b. The first two failure types can also occur on centralized systems.
22.7	**Consider a distributed system with two sites, A and B. Can site A distinguish among the following?** • **B goes down.** • **The link between A and B goes down.** • **B is extremely overloaded and response time is 100 times longer than normal.** **What implications does your answer have for recovery in distributed systems? [*2007. Marks: 2*]** Site A cannot distinguish between the three cases until communication has resumed with site B. The action which it performs while B is inaccessible must be correct irrespective of which of these situations has actually occurred, and must be such that B can re-integrate consistently into the distributed system once communication is restored.
22.8	**Explain the difference between data replication in a distributed system and the maintenance of a remote backup site.** In remote backup systems all transactions are performed at the primary site and the data is replicated at the remote backup site. The remote backup site is kept synchronized with the updates at the primary site by sending all log records. Whenever the primary site fails, the remote backup site takes over processing. The distributed systems offer greater availability by having multiple copies of the data at different sites whereas the remote backup systems offer lesser availability at lower cost and execution overhead. In a distributed system, transaction code runs at all the sites whereas in a remote backup system it runs only at the primary site. The distributed system transactions follow two-phase commit to have the data in consistent state whereas a remote backup system does not follow two-phase commit and avoids related overhead.
22.9	**What are the advantages and disadvantages of data replication? [*2007. Marks: 4*]** **Advantages of Data Replication:** 1. **Availability**: failure of site containing relation *r* does not result in unavailability of *r* as replicas exist. 2. **Parallelism**: queries on *r* may be processed by several nodes in parallel. 3. **Reduced data transfer**: The more replicas of *r* there are, the greater the chance that the needed data will be found in the site where the transaction is executing. Hence, data replocation minimizes movement of data between sites. **Disadvantages of Data Replication:** 1. **Increased overhead on update**: each replica of relation *r* must be updated. 2. **Increased complexity of concurrency control**: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
22.10	**Consider a relation that is fragmented horizontally by *plant-number*:** ***employee* (*name, address, salary, plant-number*)** **Assume that each fragment has two replicas: one stored at the New York site and one stored locally at the plant site. Describe a good processing strategy for the following queries entered at the San Jose site.** **a. Find all employees at the Boca plant.** **b. Find the average salary of all employees.** **c. Find the highest-paid employee at each of the following sites: Toronto, Edmonton,**

Vancouver, Montreal.

d. Find the lowest-paid employee in the company.

c. i. Send the query $\Pi_{name}(employee)$ to the Boca plant.
ii. Have the Boca location send back the answer.

d. i. Compute average at New York.
ii. Send answer to San Jose.

e. i. Send the query to find the highest salaried employee to Toronto, Edmonton, Vancouver, and Montreal.
ii. Compute the queries at those sites.
iii. Return answers to San Jose.

f. i. Send the query to find the lowest salaried employee to New York.
ii. Compute the query at New York.
iii. Send answer to San Jose.

22.11 **Consider the relations**

***employee* (*name, address, salary, plant-number*)**
***machine* (*machine-number, type, plant-number*)**

Assume that the employee relation is fragmented horizontally by *plant-number*, and that each fragment is stored locally at its corresponding plant site. Assume that the *machine* relation is stored in its entirety at the Armonk site. Describe a good strategy for processing each of the following queries. [*2007. Marks: 4*]

a. Find all employees at the plant that contains machine number 1130.
b. Find all employees at plants that contain machines whose type is "milling machine."
c. Find all machines at the Almaden plant.
d. Find *employee* ⋈ *machine*.

c. i. Perform $\Pi_{plant\text{-}number}(\sigma_{machine\text{-}number\ =\ 1130}(machine))$ at Armonk.
ii. Send the query $\Pi_{name}(employee)$ to all site(s) which are in the result of the previous query.
iii. Those sites compute the answers.
iv. Union the answers at the destination site.

b. i. This strategy is the same as (a), except the first step should be to perform
$\Pi_{plant\text{-}number}(\sigma_{type\ =\ \text{"milling machine"}}(machine))$ at Armonk

c. i. Perform $\sigma_{plant\text{-}number\ =\ x}(machine)$ at Armonk, where *x* is the plant-number for Almaden.
ii. Send the answers to the destination site.

d. **Strategy 1:**
i. Group machine at Armonk by plant number.
ii. Send the groups to the sites with the corresponding plant-number.
iii. Perform a local join between the local data and the received data.
iv. Union the results at the destination site.

Strategy 2:
Send the machine relation at Armonk, and all the fragments of the employee relation to the destination site. Then perform the join at the destination site.

There is parallelism in the join computation according to the first strategy but not in the second. Nevertheless, in a WAN the amount of data to be shipped is the main cost factor. We expect that each plant will have more than one machine, hence the result of the local join at each site will be a cross-product of the employee tuples and machines at that plant. This cross-product's size is greater than the size of the employee fragment at that site. As a result the second strategy will result in less data shipping, and will be more efficient.

22.12 **For each of the strategies in the previous question-answer, state how your choice of a strategy depends on:**

a. The site at which the query was entered
b. The site at which the result is desired

a. Assuming that the cost of shipping the query itself is minimal, the site at which the query was submitted does not affect our strategy for query evaluation.

b. For the first query, we find out the plant numbers where the machine number 1130 is present, at Armonk. Then the employee tuples at all those plants are shipped to the destination site. We can see that this strategy is more or less independent of the destination site. The same can be said of the second query. For the third query, the selection is performed at Armonk and results shipped to the destination site. This strategy is obviously independent of the destination site.

For the fourth query, we have two strategies. The first one performs local joins at all the plant sites and their results are unioned at the destination site. In the second strategy, the machine relation at Armonk as well as all the fragments of the employee relation are first shipped to the destination, where the join operation is performed. There is no obvious way to optimize these two strategies based on the destination site. In the answer to the previous question, we saw the reason why the second strategy is expected to result in less data shipping than the first. That reason is independent of destination site, and hence we can in general prefer strategy two to strategy one, regardless of the destination site.

22.13 **Compute $r \ltimes s$ for the relations of the following figure: [*2007. Marks: 4*]**

A	*B*	*C*
1	2	3
4	5	6
1	2	4
5	3	2
8	9	7

r

C	*D*	*E*
3	4	5
3	6	8
2	3	2
1	4	1
1	2	3

s

Is $r_i \ltimes r_j$ necessarily equal to $r_j \ltimes r_i$? Under what conditions does $r_i \ltimes r_j = r_j \ltimes r_i$ hold?

$r \ltimes s$ =

A	B	C
1	2	3
5	3	2

In general, $r_i \ltimes r_j \neq r_j \ltimes r_i$. This can be easily seen here where $r \ltimes s \neq s \ltimes r$. $r \ltimes s$ is as above while

$s \ltimes r$ =

C	D	E
3	4	5
3	6	8
2	3	2

By definition, $r_i \ltimes r_j = \Pi_{Ri}(r_i \bowtie r_j)$ and $r_j \ltimes r_i = \Pi_{Rj}(r_i \bowtie r_j)$, where R_i and R_j are the schemas of r_i and r_j respectively. For $\Pi_{Ri}(r_i \bowtie r_j)$ to be always equal to $\Pi_{Rj}(r_i \bowtie r_j)$, the schemas R_i and R_j must be the same.