

## Advance Database Management System

Q.No.1.a. Comparison between RDBMS, OODBMS AND ORDBMS

05

Ans:

	rdbms	ordbms	oodbms
1) Object	Not used	Transient object is used	Object is used
2) Inheritance	Not possible	Applicable for defined data types and tables	used
3) Declaration	table	table	table
4) Query language	SQL 3	SQL 3	OQL
5) Id	Primary key	OID is used through 'ref' keyword	OID is used
6) Integrity	used	used	used
7) Data models	relational	relational	object

Q.No.1.d. Write a short note on query processing in distributed database.

10

ans:

first is the cost of transferring data over the network. This data includes intermediate files that are transferred to other sites for further processing, as well as the final result files that may have to be transferred to the site where the query result is needed. Although these costs may not be very high if the sites are connected via a high performance local area network, they become quite significant in other types of networks.

Hence, DDBMS query optimization algorithms consider the goal of reducing the *amount of data transfer* as an optimization criterion in choosing a distributed query execution strategy.

the size of

the EMPLOYEE relation is  $100 * 10,000 = 10^6$  bytes, and the size of the DEPARTMENT relation is

Advance Database Management System

---

35 \* 100 = 3500 bytes. Consider the query Q: "For each employee, retrieve the employee

SITE1:

EMPLOYEE

10,000records

eachrecord is 100byteslong

SSNfieldis 9 byteslong

DNOfieldis 4 byteslong

SITE2:

DEPARTMENT

FNAMEfieldis 15byteslong

LNAMEfieldis 15byteslong

I DNAME I DNUMBER , MGRSSN I MGRSTARTDATE

100records

eachrecord is35 byteslong

DNUMBER fieldis 4 byteslong DNAME fieldis 10byteslong

MGRSSN fieldis 9 byteslong

Example to illustrate *volume* of data transferred.

name and the name of the department for which the employee works." This can be stated as follows in the relational algebra:

Q: '1TFNAME, LNAME,DNAME (EMPLOYEE ~ DNO~DNUMBER DEPARTMENT)

The result of this query will include 10,000 records, assuming that every employee is related to a department. Suppose that each record in the query result is 40 *bytes long*. The

query is submitted at a distinct site 3, which is called the result site because the query

result is needed there. Neither the EMPLOYEE nor the DEPARTMENT relations reside at site 3.

There are three simple strategies for executing this distributed query:

1. Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3. In this case a total of  $1,000,000 + 3500 = 1,003,500$  bytes

must be transferred.

## Advance Database Management System

---

2. Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3. The size of the query result is  $40 * 10,000 = 400,000$  bytes, so  $400,000 + 1,000,000 = 1,400,000$  bytes must be transferred.

3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3. In this case  $400,000 + 3500 = 403,500$  bytes must be transferred.

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

$Q'$ :  $\pi_{TFNAME, LNAME, DNAME}(\sigma_{MGRSSN \sim SSN}(\text{DEPARTMENT} \bowtie \text{EMPLOYEE}))$

### 25.4 Query Processing in Distributed Databases

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

1. Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3. In this case a total of  $1,000,000 + 3500 = 1,003,500$  bytes must be transferred.

2. Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3. The size of the query result is  $40 * 100 = 4000$  bytes, so  $4000 + 1,000,000 = 1,004,000$  bytes must be transferred.

3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3. In this case  $4000 + 3500 = 7500$  bytes must be transferred.

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

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

## Advance Database Management System

Provide the Query Information to recover from failure

- A Transfer the DEPARTMENT relation to site 1, execute the query at site 1, and send the result back to site 2. In this case  $400,000 + 3500 = 403,500$  bytes must be transferred

for Q and  $4000 + 3500 = 7500$  bytes for Q'.

g

.

Q.No.3.a. What is DTD? Give the DTD for an XML representation for the following nested relational schema

EMP=(ename,childrensetsetof(children),skillsetsetof(skill))

CHILDREN=(name,birthday)

BIRTHDAY=(day,month,year)

SKILLS=(type,examset,setoff(exams))

EXAMS=(year,city)

10

Ans:

XML DTD file, which specifies the elements (tag names) and their nested structures.

Any valid documents conforming

to this DTD should follow the specified structure. A special syntax exists for specifying DTD files, as illustrated in Figure First, a name is given to the root tag of the Document

- \* : the element name means that the element can be repeated zero or more times in the document. This kind of element is known as an *optional multivalued (repeating) element*.
- + : the element name means that the element can be repeated one or more times in the document. This kind of element is a *required multivalued(repeating) element*.
- ? : the element name means that the element can be repeated zero or one times. This kind is an *optional single-valued (nonrepeating) element*.
- An element appearing without any of the preceding three symbols must appear exactly once in the document. This kind is a *required single-valued (nonrepeating) element*.
- The type of the element is specified via parentheses following the element. If the parentheses include names of other elements, these latter elements are the *children* of the element in the tree structure. If the parentheses include the keyword #PCDATA or one of the other data types available in XML DTD, the element is a leaf node. PCDATA stands for *parsed character data*, which is roughly similar to a string data type.
- Parentheses can be nested when specifying elements.
- A bar symbol ( $e \mid e_z$ ) specifies that either  $e$  or  $e_z$  can appear in the document.

## Advance Database Management System

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE projects SYSTEM "emp.dtd">
<!DOCTYPE Employee [
<!ELEMENT Employee (Emp+)>
<!ELEMENT Emp (ename, Childrenset, Skillset)
<!ELEMENT ename (#PCDATA)
<!ELEMENT Childrenset (children*)
<!ELEMENT children (name, Birthday)
<!ELEMENT name (#PCDATA)
<!ELEMENT Birthday (Bday)
<!ELEMENT Bday (day, month, year)
<!ELEMENT day (#PCDATA)
<!ELEMENT month (#PCDATA)
<!ELEMENT year (#PCDATA)
<!ELEMENT Skillset (skill*)
<!ELEMENT skill (type, Examset)
<!ELEMENT type (#PCDATA)
<!ELEMENT Examset (Exam*)
<!ELEMENT Exam (year, city)
<!ELEMENT year (#PCDATA)
<!ELEMENT city (#PCDATA)

```

Q.No.4.a.

Explain with proper example nested relation in ORDBMS.

10

Ans

## Nested Relation

In *first normal form* (1NF), which requires that all attributes have *atomic domains*. That a domain is *atomic* if elements of the domain are considered to be indivisible units. The assumption of 1NF is a natural one in the bank examples we have considered. However, not all applications are best modeled by 1NF relations. For example, rather than view a

## Advance Database Management System

database as a set of records, users of certain applications view it as a set of objects (or entities). These objects may require several records for their representation. We shall see that a simple, easy-to-use interface requires a one-to-one correspondence between the user's intuitive notion of an object and the database system's notion of a data item.

<i>title</i>	<i>author-set</i>	<i>publisher</i> ( <i>name, branch</i> )	<i>keyword-set</i>
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

**Figure 9.1** Non-1NF books relation, *books*.

The **nested relational model** is an extension of the relational model in which domains may be either atomic or relation valued. Thus, the value of a tuple on an attribute may be a relation, and relations may be contained within relations. A complex object thus can be represented by a single tuple of a nested relation. If we view a tuple of a nested relation as a data item, we have a one-to-one correspondence between data items and objects in the user's view of the database. We illustrate nested relations by an example from a library. Suppose we store for

each book the following information:

- Book title
  - Set of authors
  - Publisher
  - Set of keywords
- **Authors.** A book may have a set of authors. Nevertheless, we may want to find all books of which Jones was one of the authors. Thus, we are interested in a subpart of the domain element "set of authors."
  - **Keywords.** If we store a set of keywords for a book, we expect to be able to retrieve all books whose keywords include one or more keywords. Thus, we view the domain of the set of keywords as nonatomic.
  - **Publisher.** Unlike *keywords* and *authors*, *publisher* does not have a set-valued domain. However, we may view *publisher* as consisting of the subfields *name* and *branch*. This view makes the domain of *publisher* nonatomic.

Figure shows an example relation, *books*. The *books* relation can be represented

<i>title</i>	<i>author</i>	<i>pub-name</i>	<i>pub-branch</i>	<i>keyword</i>
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

Figure 9.2 *flat-books*, a 1NF version of non-1NF relation *books*.

Q.No.5.a.

Explain data fragmentation, replication and allocation technique for distributed database design.

10

Ans:

### Data Fragmentation

If relation  $r$  is fragmented,  $r$  is divided into a number of fragments  $r_1, r_2, \dots, r_m$ . These fragments contain sufficient information to allow reconstruction of the original relation  $r$ . There are two different schemes for fragmenting a relation: *horizontal* fragmentation and *vertical* fragmentation. Horizontal fragmentation splits the relation by assigning each tuple of  $r$  to one or more fragments. Vertical fragmentation splits the relation by decomposing the scheme  $R$  of relation  $r$ . We shall illustrate these approaches by fragmenting the relation *account*, with the Schema *Account-schema* = (*account-number*, *branch-name*, *balance*)

In **horizontal fragmentation**, a relation  $r$  is partitioned into a number of subsets,

$r_1, r_2, \dots, r_m$ . Each tuple of relation  $r$  must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed. As an illustration, the *account* relation can be divided into several different fragments, each of which consists of tuples of accounts belonging to a particular branch. If the banking system has only two branches—Hillside and Valleyview—then there are two different fragments:

$$account_1 = \sigma_{branch-name = \text{“Hillside”}}(account)$$

$$account_2 = \sigma_{branch-name = \text{“Valleyview”}}(account)$$

Horizontal fragmentation is usually used to keep tuples at the sites where they are

used the most, to minimize data transfer. In general, a horizontal fragment can be defined as a *selection* on the global relation  $r$ . That is, we use a predicate  $P_i$  to construct fragment  $r_i$ :

$$r_i = \sigma_{P_i}(r)$$

## Advance Database Management System

---

We reconstruct the relation  $r$  by taking the union of all fragments; that is,

$$r = r_1 \cup r_2 \cup \dots \cup r_n$$

In our example, the fragments are disjoint. By changing the selection predicates used to construct the fragments, we can have a particular tuple of  $r$  appear in more than one of the  $r_i$ . In its simplest form, vertical fragmentation is the same as decomposition. **Vertical fragmentation** of  $r(R)$  involves the definition of several subsets of attributes  $R_1, R_2, \dots, R_n$  of the schema  $R$  so that

$$R = R_1 \cup R_2 \cup \dots \cup R_n$$

Each fragment  $r_i$  of  $r$  is defined by

$$r_i = \Pi_{R_i}(r)$$

The fragmentation should be done in such a way that we can reconstruct relation  $r$  from the fragments by taking the natural join

$$r = r_1 \bowtie r_2 \bowtie r_3 \bowtie \dots \bowtie r_n$$

One way of ensuring that the relation  $r$  can be reconstructed is to include the primary-key attributes of  $R$  in each of the  $R_i$ . More generally, any superkey can be used. It is often convenient to add a special attribute, called a *tuple-id*, to the schema  $R$ . The tuple-id value of a tuple is a unique value that distinguishes the tuple from all other tuples. The tuple-id attribute thus serves as a candidate key for the augmented schema, and is included in each of the  $R_i$ s. The physical or logical address for a tuple can be used as a tuple-id, since each tuple has a unique address.

To illustrate vertical fragmentation, consider a university database with a relation *employee-info* that stores, for each employee, *employee-id*, *name*, *designation*, and *salary*. For privacy reasons, this relation may be fragmented into a relation *employee-private-info* containing *employee-id* and *salary*, and another relation *employee-public-info* containing attributes *employee-id*, *name*, and *designation*. These may be stored at different sites, again for security reasons.

The two types of fragmentation can be applied to a single schema; for instance, the

fragments obtained by horizontally fragmenting a relation can be further partitioned vertically. Fragments can also be replicated. In general, a fragment can be replicated, replicas of fragments can be fragmented further, and so on.

### Data Replication

If relation  $r$  is replicated, a copy of relation  $r$  is stored in two or more sites. In the most extreme case, we have **full replication**, in which a copy is stored in every site in the system. There are a number of advantages and disadvantages to replication.

## Advance Database Management System

• **Availability.** If one of the sites containing relation  $r$  fails, then the relation  $r$  can be found in another site. Thus, the system can continue to process queries involving  $r$ , despite the failure of one site.

• **Increased parallelism.** In the case where the majority of accesses to the relation  $r$  result in only the reading of the relation, then several sites can process queries involving  $r$  in parallel. 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 replication minimizes movement of data between sites.

• **Increased overhead on update.** The system must ensure that all replicas of a relation  $r$  are consistent; otherwise, erroneous computations may result. Thus, whenever  $r$  is updated, the update must be propagated to all sites containing replicas. The result is increased overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary to ensure that the balance in a particular account agrees in all sites. In general, replication enhances the performance of read operations and increases the availability of data to read-only transactions. However, update transactions incur greater overhead. Controlling concurrent updates by several transactions to replicated data is more complex than in centralized systems. We can simplify the management of replicas of relation  $r$  by choosing one of them

as the **primary copy** of  $r$ . For example, in a banking system, an account can be associated with the site in which the account has been opened. Similarly, in an airline reservation system, a flight can be associated with the site at which the flight originates.

Q.No.6.a

Explain with suitable example Aggregation, Specialization and generalization in EER diagram..

10

Ans:

### Specialization

An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings. Consider an entity set *person*, with attributes *name*, *street*, and *city*. A person may be further classified as one of the following:

- *customer*
- *employee*

Each of these person types is described by a set of attributes that includes all the attributes of entity set *person* plus possibly additional attributes. For example, *customer* entities may be described further by the attribute *customer-id*,

## Advance Database Management System

---

whereas *employee* entities may be described further by the attributes *employee-id* and *salary*. The process of designating subgroupings within an entity set is called **specialization**. The specialization of *person* allows us to distinguish among persons according to whether they are employees or customers.

We can apply specialization repeatedly to refine a design scheme. For instance,

bank employees may be further classified as one of the following:

- *officer*
- *teller*
- *secretary*

Each of these employee types is described by a set of attributes that includes all the attributes of entity set *employee* plus additional attributes. For example, *officer* entities may be described further by the attribute *office-number*, *teller* entities by the attributes *station-number* and *hours-per-week*, and *secretary* entities by the attribute *hours-perweek*. Further, *secretary* entities may participate in a relationship *secretary-for*, which identifies which employees are assisted by a secretary.

An entity set may be specialized by more than one distinguishing feature. In our example, the distinguishing feature among employee entities is the job the employee performs. Another, coexistent, specialization could be based on whether the person is a temporary (limited-term) employee or a permanent employee, resulting in the entity sets *temporary-employee* and *permanent-employee*. When more than one specialization is formed on an entity set, a particular entity may belong to multiple specializations. For instance, a given employee may be a temporary employee who is a secretary.

In terms of an E-R diagram, specialization is depicted by a *triangle* component labeled **ISA**, as Figure shows. The label ISA stands for “is a” and represents, for example, that a customer “is a” person. The ISA relationship may also be referred to as a **superclass-subclass** relationship. Higher- and lower-level entity sets are depicted as regular entity sets—that is, as rectangles containing the name of the entity set.

### Generalization

The refinement from an initial entity set into successive levels of entity subgroupings represents a **top-down** design process in which distinctions are made explicit. The design process may also proceed in a **bottom-up** manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features. The database designer may have first identified a *customer* entity set with the attributes *name*, *street*, *city*, and *customer-id*, and an *employee* entity set with the attributes *name*, *street*, *city*, *employee-id*, and *salary*. There are similarities between the *customer* entity set

## Advance Database Management System

and the *employee* entity set in the sense that they have several attributes in common. This commonality can be expressed by **generalization**, which is a containment relationship that exists between a *higher-level* entity set and one or more *lower-level* entity sets. In our example, *person* is the higher-level entity set and *customer* and *employee* are lower-level entity sets. Higher- and lower-level entity sets also may be designated by the terms **superclass** and **subclass**, respectively. The *person* entity set is the superclass of the *customer* and *employee* subclasses.

For all practical purposes, generalization is a simple inversion of specialization.

We will apply both processes, in combination, in the course of designing the E-R

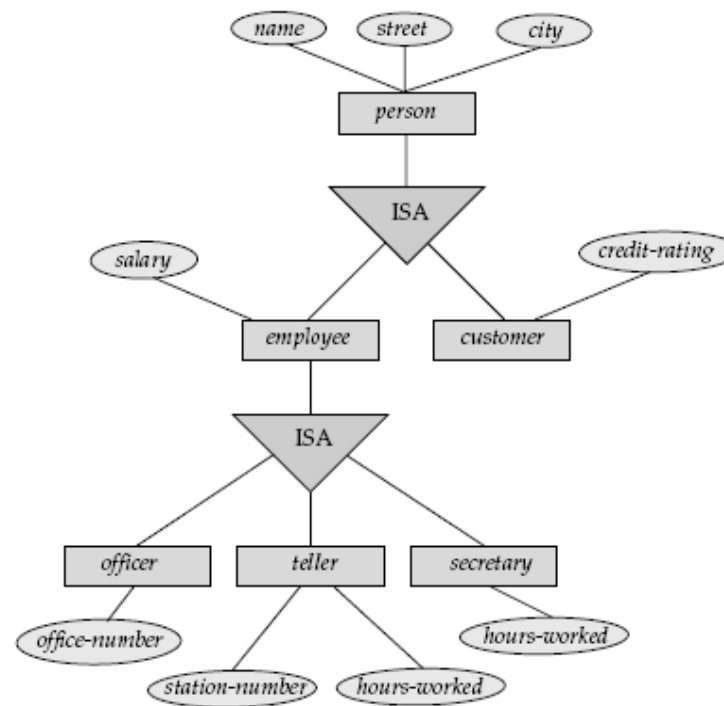


Figure 2.17 Specialization and generalization.

schema for an enterprise. In terms of the E-R diagram itself, we do not distinguish between specialization and generalization. New levels of entity representation will be distinguished (specialization) or synthesized (generalization) as the design schema comes to express fully the database application and the user requirements of the database. Differences in the two approaches may be characterized by their starting point and overall goal.

### Aggregation-

Example-

Advance Database Management System

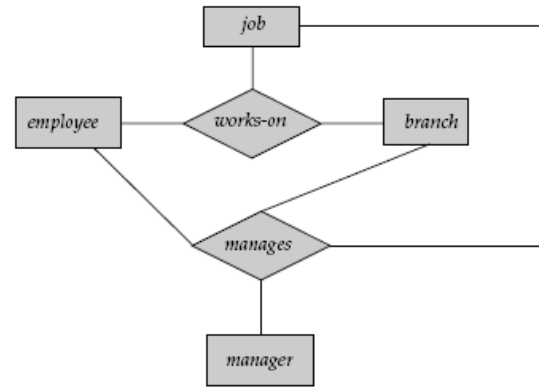


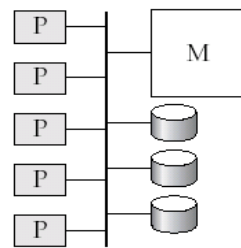
Figure 2.18 E-R diagram with redundant relationships.

Q.No.6.b.

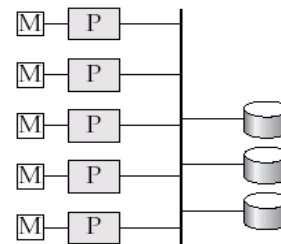
Explain different architecture of parallel database.

10

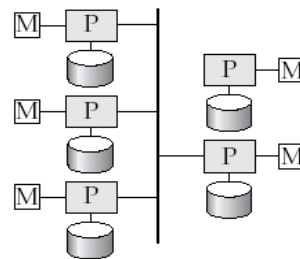
Ans



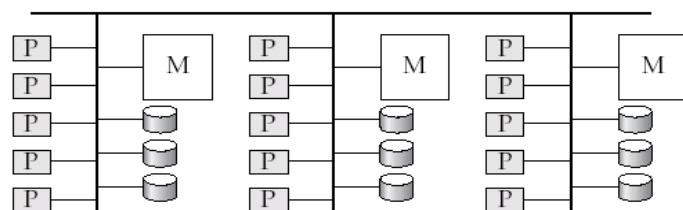
(a) shared memory



(b) shared disk



(c) shared nothing



(d) hierarchical

In a shared-memory architecture, the processors and disks have access to a common memory, typically via a bus or through an interconnection network. The benefit of shared memory is extremely efficient communication between processors—data in shared memory can be accessed by any processor without being moved with software. A processor can send messages to other processors much faster by using memory writes (which usually take less than a microsecond) than by sending a message through a communication mechanism. Shared-memory architectures usually have large memory caches at each processor, so that referencing of the shared memory is avoided whenever possible. However, at least some of the data will not be in the cache, and accesses will have to go to the shared memory.

## Advance Database Management System

---

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

of this architecture over a shared-memory architecture. First, since each processor has its own memory, the memory bus is not a bottleneck. Second, 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.

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. A node functions as the server for the data on the disk or disks that the node owns. 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. Moreover, 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

### References-

- 1.Database System Concept by Silbarchtz,Kort,Sudarshan
- 2.Fundamentals Of Database System S.B.Navathe

