

Q.1 a) Explain the following terms-

i) Data Warehouse 2) Inheritance

5

Ans:

**i) Data Warehouse-**

- Provide strategic information
- Functional definition

The data warehouse is an informal environment that

- Provide an integral & total view of an enterprise
- Makes the enterprises current & historical information easily available for decision making.
- Present a flexible & interactive source of strategic information.

“A data warehouse is a subject orient, integrated nonvolatile & time variant collection of data in support of management decision.”

- Characteristics of data warehouse-

- Subject orient
- Integrated
- Time variant
- Nonvolatile
- Data Granularity

1) Subject Oriented

- In operational system, we store data by individual application
- We keep the data for particular application. ex-banking
- In DW, data is stored by subject not by application.
- In DW, there is no application flavor.

2) Integrated data

- Source data are in different db. files & data segment.
- There are disparate application , so the operational platforms & operating system could be different.

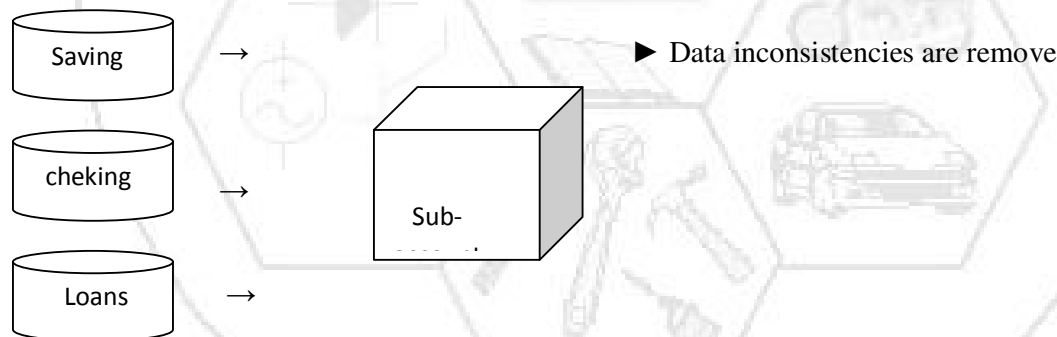


Fig: DW is integrated

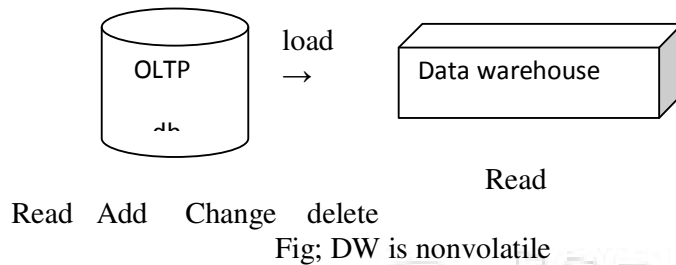
3) Time variant data

- For an operational system the stored data contains the current values.
- In accounts receivable system the balance is the current outstanding balance in the customer's accounts.

4) Nonvolatile data

- Data in DW is not intended to run the day to day business.

The operational order entry application is used for this purpose. You do not update the DW every time, when you want to process single order.



5) Data Granularity

Data granularity in a DW refers to level of details, the lower the level of detail, the finer the data granularity.

**ii) Inheritance**

-Child class inherits all the attributes & properties of parent.

-This is an imp. Concept associated with subclasses is that of type inheritance.

An entity that is a member of a subclass inherits all the attributes of an entity as a member of the superclass.

- The entity also inherits all the relationship in which superclass participate.

Example;

Consider, employer – superclass

Engineer, secretary – subclass

*(b) Explain 3NF with suitable ex. 5*

**Third normal form:**

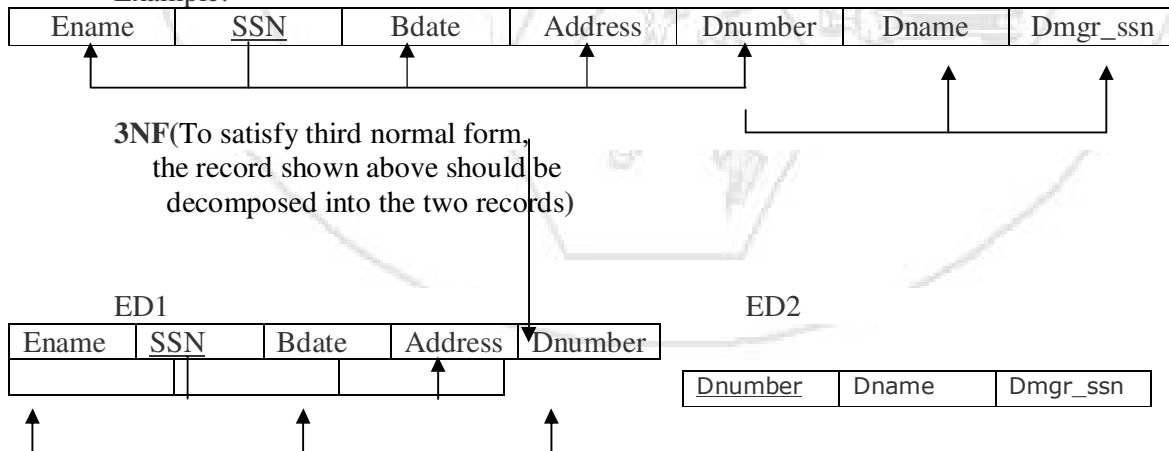
Normalization is the process of structuring relational database schema such that most ambiguity is removed. The stages of normalization are referred to as normal forms and progress from the least restrictive (First Normal Form) through the most restrictive (Fifth Normal Form).

Definition: A relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

There are two basic requirements for a database to be in third normal form:

- Already meet the requirements of both 1NF and 2NF
- Remove columns that are not fully dependent upon the primary key.

Example:



As shown above ED1 & ED2 represent independent entity facts about employees and department.

(c) Explain Data Fragmentation in distributed Database.

5

Ans:

Data Fragmentation:

- 1) Horizontal Fragmentation
- 2) Vertical Fragmentation
- 3) Mixed (Hybrid) Fragmentation

### 1) Horizontal fragmentation:

-Horizontal fragmentation splits the relation by assuming each tuple of r to one or more fragments.

-Each fragment consists of a subset of rows of the original relation.

-horizontal fragmentation divides a relation horizontally by grouping rows to create subset of tuples.

-Ex. in case of company schema, we assume there are three computer sites- one for each department in the company. if we want to store database information relating to each partition each relation by department.

- Horizontal fragmentation on relation R can be specified by a  $\sigma_{C_i}(R)$  operation in the relation algebra. A set of all horizontal fragments whose conditions  $C_1, C_2, \dots, C_n$  include all the tuples in R- is called a complete horizontal fragmentation.

-to reconstruct the relation R from a complete horizontal fragmentation, we need to apply the UNION operation to the fragments.

### Vertical fragmentation:

-Each fragment consists of a subset of columns of the original relation.

- vertical fragmentation divides a relation attributes of the relation.

-A vertical fragment on a relation R can be specified by a  $\Pi_{L_i}(R)$  operation in the relational algebra. A set of vertical fragments projection lists  $L_1, L_2, \dots, L_n$  include all the attributes in R but share only primary key attribute of R is called a complete vertical fragmentation.

- to reconstruct the relation R from a complete vertical fragmentation, we apply the OUTER UNION operation to the vertical fragment

Example:

Tid	Ename	Eid	city	Age	salary
T1	53666	Jones	madras	18	20000
T2	53668	smith	chicago	20	99990
T3	53650	jackson	chicago	24	34555
T4	53831	ramesh	mumbai	25	36465
T5	53832	jonson	mumbai	32	62000

Fig. vertical & horizontal fragmentation.

### 3) Mixed (Hybrid) Fragmentation

We can intermix the two types of fragmentation, yielding a **mixed fragmentation**. For example, we may combine the horizontal and vertical fragmentations of the EMPLOYEE relation given earlier into a mixed fragmentation that includes six fragments. In this case the original relation can be reconstructed by applying UNION and OUTER UNION (or OUTER JOIN) operations in the appropriate order

(d) Explain Transient & Persistent objects.

5

Ans:

**transient objects:** the objects which exist only during program execution and disappear once the program terminates.

**Persistent objects:** are stored permanently in the database and persist after program termination.

Typical mechanisms for making object persistent are: i) Naming  
ii) Reachability

**i) Naming Mechanism:**

- It involves giving an object a unique persistent name through which it can be retrieved by this & other programs.
- this persistent object name can be given via a specific statement or operation in the program as shown in fig below.
- All the names must be unique within database.
- it is practical to give names to all objects in a large database that includes thousands of objects so most objects are made persistent by using second mechanism i.e. reachability.

**ii) Reachability:**

-it works by making the object reachable from some persistent object.

$A \square B$

-An object B is said to be reachable from an object A

Example: creating persistent object by naming & reachability

**Define class Department\_Set:**

**Type set (DEPARTMENT);**

**Operations add\_dept (d: DEPARTMENT): boolean;**

**remove\_dept(d: DEPARTMENT): boolean;**

(\*removes a department to the Department\_Set object \*)

**create\_dept\_set: Department\_Set;**

**destroy\_dept\_set: boolean;**

**end Department\_Set;**

...

**persistent name ALL\_DEPARTMENTS:DEPARTMENT\_SET;**

(\*ALL\_DEPARTMENTS is a persistent named object of type DEPARTMENT\_SET \*)

...

**D:=create\_dept;**

(\*create a new DEPARTMENT object in the variable d\*)

...

**b:=ALL\_DEPARTMENTS.add\_dept(d);**

(\*make d persistent by adding it to the persistent set ALL\_DEPARTMENTS\*)

-As shown in above example we define a class Department\_Set whose objects are of type set(DEPARTMENT).

-Suppose that an object of type Department\_Set is created & suppose that it is named ALL\_DEPARTMENTS by using the add\_dept operation becomes persistent by virtue of its being reachable from ALL\_DEPARTMENT.

-the ALL\_DEPARTMENT object is often called the extent of the class DEPARTMENT, as it will hold all persistent object of type DEPARTMENT.

**Q.2 (a) What are triggers? Give an example. Illustrate the cases when triggers must not be used. 10**

Ans:

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database.

A trigger description contains three parts:

- Event: A change to the database that activates the trigger.
- Condition: A query or test that is run when the trigger is activated.
- Action: A procedure that is executed when the trigger is activated and its condition is true.

A trigger *action* can examine the answers to the query in the condition part of the trigger, refer to old and new values of tuples modified by the statement activating the trigger, execute new queries, and make changes to the database.

In fact, an action can even execute a series of data-definition commands (e.g., create new tables, change authorizations) and transaction-oriented commands (e.g., commit) or call host-language procedures.

### Examples of Triggers in SQL

The examples shown in Figure given below, written using Oracle Server syntax for defining triggers, illustrate the basic concepts behind triggers. (The SQL:1999 syntax for these triggers is similar; we will see an example using SQL:1999

syntax shortly.) The trigger called *iniLcount* initializes a counter variable before every execution of an INSERT statement that adds tuples to the Students relation. The trigger called *incr\_count* increments the counter for each inserted tuple that satisfies the condition *age < 18*.

One of the example triggers in Figure given below executes before the activating statement, and the other example executes after it.

A trigger can also be scheduled to execute *instead of* the activating statement; or in *deferred* fashion, at the end of the transaction containing the activating statement; or in *asynchronous* fashion, as part of a separate transaction.

```
CREATE TRIGGER iniLcount BEFORE INSERT ON Students I* Event *I
DECLARE
count INTEGER;
BEGIN I* Action *I
count := 0;
END
CREATE TRIGGER incLcount AFTER INSERT ON Students I* Event *I
WHEN (new.age < 18) I* Condition; 'new' is just-inserted tuple *I
FOR EACH ROW
BEGIN I* Action; a procedure in Oracle's PL/SQL syntax *I
count := count + 1;
END
```

Figure . Examples Illustrating Triggers

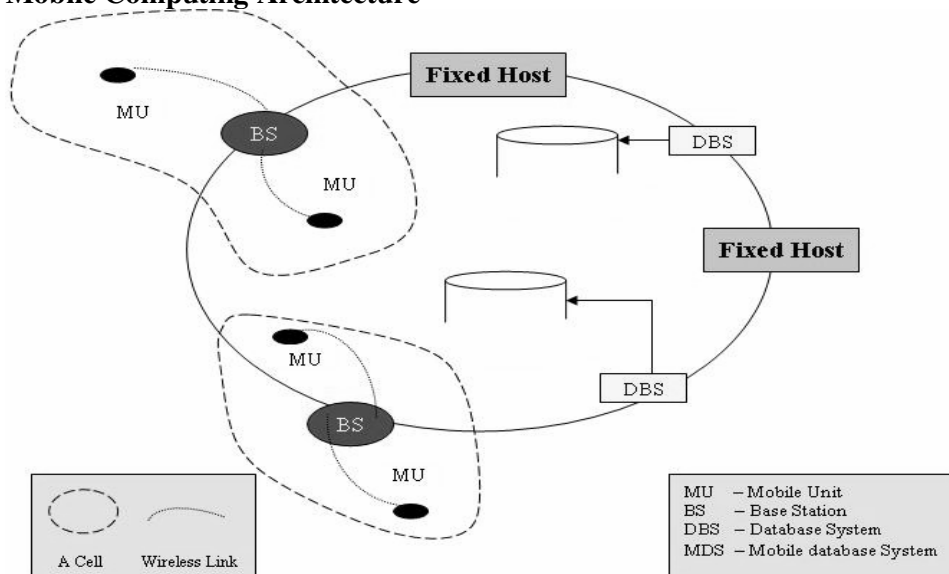
(b) Explain design & implementation issues in mobile databases. 10

Ans:

Recent advances in wireless technology have led to **mobile computing**, a new dimension in data communication and processing. The mobile computing environment will provide database applications with useful aspects of wireless technology. The mobile computing platform allows users to establish communication with other users and to manage their work while they are mobile. This feature is especially useful to geographically dispersed organizations. Typical examples might include traffic

police, taxi dispatchers, and weather reporting services, as well as financial market reporting and information brokering applications.

### Mobile Computing Architecture



### Characteristics of Mobile Environments

The general architecture of a mobile platform is illustrated in Figure 27.04. It is a distributed architecture where a number of computers, generally referred to as **Fixed Hosts (FS)** and **Base Stations (BS)**, are interconnected through a high-speed wired network. Fixed hosts are general purpose computers that are not equipped to manage mobile units but can be configured to do so. Base stations are equipped with wireless interfaces and can communicate with mobile units to support data access.

**Mobile Units (MU) (or hosts)** and base stations communicate through wireless channels having bandwidths significantly lower than those of a wired network. A **downlink channel** is used for sending data from a BS to an MU and an **uplink channel** is used for sending data from an MU to its BS. Recent products for portable wireless have an upper limit of 1 Mbps (megabits per second) for infrared communication, 2 Mbps for radio communication, and 9.14 Kbps (kilobits per second) for cellular telephony. Ethernet, by comparison, provides 10 Mbps fast Ethernet and FDDI provide 100 Mbps and ATM (asynchronous transfer mode) provides 155 Mbps.

Mobile units are battery-powered portable computers that move freely in a **geographic mobility domain**, an area that is restricted by the limited bandwidth of wireless communication channels. To manage the mobility of units, the entire geographic mobility domain is divided into smaller domains called **cells**. The mobile discipline requires that the movement of mobile units be unrestricted within the geographic mobility domain (intercell movement), while having information **access contiguity** during movement guarantees that the movement of a mobile unit across cell boundaries will have no effect on the data retrieval process.

The mobile computing platform can be effectively described under the client-server paradigm, which means we may sometimes refer to a mobile unit as a client or sometimes as a user, and the base stations as servers. Each cell is managed by a base station, which contains transmitters and receivers for responding to the information-processing needs of clients located in the cell. We assume that the average query response time is much shorter than the time required by the client to physically traverse a cell.

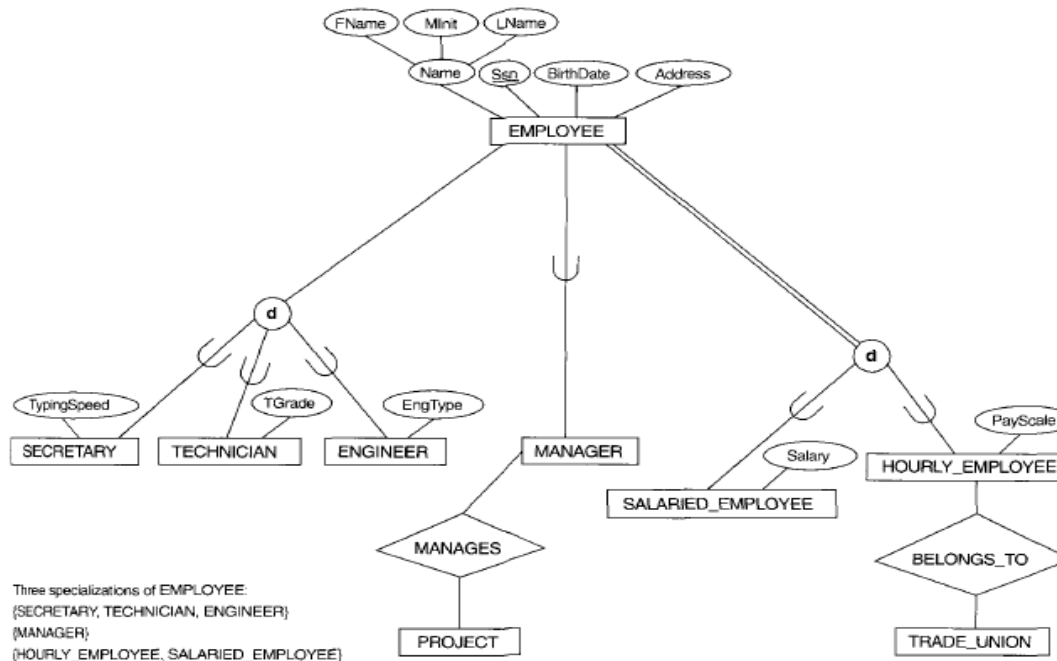
Clients and servers communicate through wireless channels. The communication link between a client and a server may be modeled as multiple data channels or as a single channel.

**Q.3 (a) Explain various extended features of ER diagram such as aggregation, specialization & generalization with suitable example. 10**

Ans:

### Extended features of ER diagram

The EER (Enhanced ER) model includes all the modeling concepts of the ER model. In addition, it includes the concepts of subclass and superclass and the related concepts of specialization and generalization.



**FIGURE 4.1** EER diagram notation to represent subclasses and specialization.

### Specialization

Specialization is the process of defining a *set of subclasses* of an entity type; this entity type is called the superclass of the specialization.

The set of subclasses that form a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.

For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the *job type* of each employee entity.

We may have several specializations of the same entity type based on different distinguishing characteristics.

For example, another specialization of the EMPLOYEE entity type may yield the set of subclasses {SALARIED\_EMPLOYEE, HOURLY\_EMPLOYEE}; this specialization distinguishes among employees based on the *method of pay*.

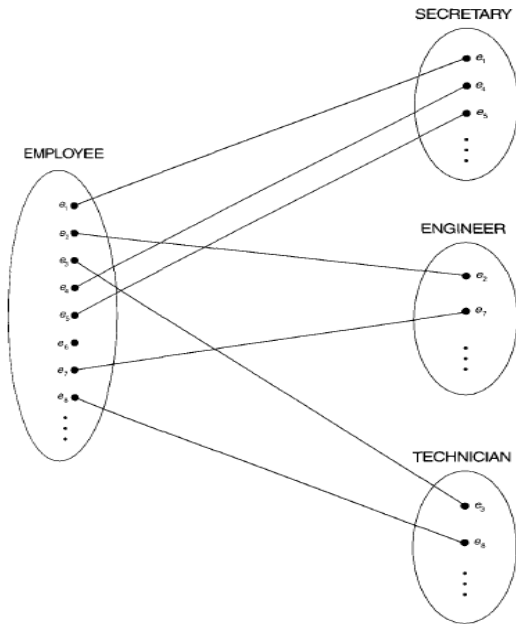


FIGURE 4.2 Instances of a specialization.

**Generalization**

We can think of a *reverse process* of abstraction in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses.

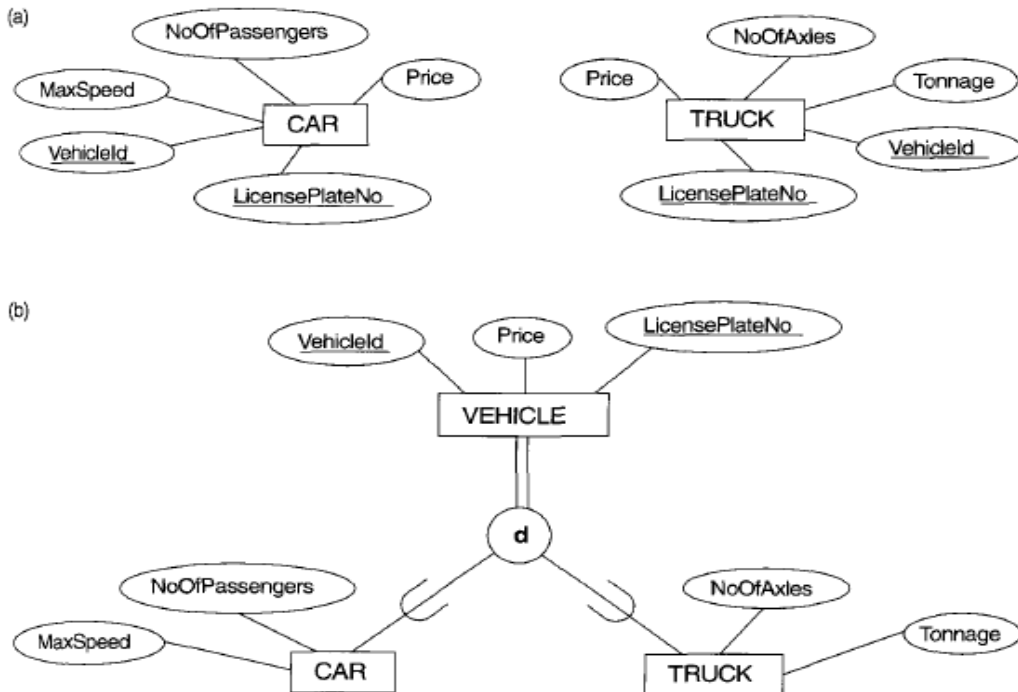


FIGURE 4.3 Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

For example, consider the entity types CAR and TRUCK shown in Figure 4.3a. Because they have several common attributes, they can be generalized into the entity type VEHICLE, as shown in Figure

4.3b. Both CAR and TRUCK are now subclasses of the generalized superclass VEHICLE. We use the term generalization to refer to the process of defining a generalized entity type from the given entity types

(b) Find out the data transfer cost of distributed query processing for following queries. "For each employee, retrieve the employee name & name of the department for which employee work."

10

Ans:

SITE 1:

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
-------	-------	-------	-----	-------	---------	-----	--------	----------	-----

10,000 records

each record is 100 bytes long

SSN field is 9 bytes long

DNO field is 4 bytes long

FNAME field is 15 bytes long

LNAME field is 15 bytes long

SITE 2:

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
-------	---------	--------	--------------

100 records

each record is 35 bytes long

DNUMBER field is 4 bytes long

MGRSSN field is 9 bytes long

DNAME field is 10 bytes long

**FIGURE 25.6** Example to illustrate volume of data transferred.

We will assume in this example that neither relation is fragmented. According to Figure shown above: The size of the EMPLOYEE relation is  $100 * 10,000 = 106$  bytes, And the size of the DEPARTMENT relation is  $35 * 100 = 3500$  bytes.

Consider the query Q: "For each employee, retrieve the employee name and the name of the department for which the employee works." This can be stated as follows in the relational algebra:

FNAME, 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.
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.
4. (a) Consider relation R (PQRSTU) with following dependencies,

Have sal  $P \rightarrow Q, ST \rightarrow PR, S \rightarrow U$

State R is in which normal form ? Decompose it to BCNF. Show step by procedure.

Ans:

(b) Explain object identifier and object structure with example.

10

Ans:

### object identifier

One goal of OO databases is to maintain a direct correspondence between real-world and database objects so that objects do not lose their integrity and identity and can easily be identified and operated upon. Hence, OO databases provide a unique system-generated *object identifier* (OID) for each object.

An OO database system provides a **unique identity** to each independent object stored in the database. This unique identity is typically implemented via a unique, system-generated **object identifier**, or **OID**. The value of an OID is not visible to the external user, but it is used internally by the system to identify each object uniquely and to create and manage inter-object references.

Properties:

The main property required of an OID is that it be **immutable**; that is, the OID value of a particular object should not change. This preserves the identity of the real-world object being represented. Hence, an OO database system must have some mechanism for generating OIDs and preserving the immutability property.

It is also desirable that each OID be used only once; that is, even if an object is removed from the database, its OID should not be assigned to another object.

These two properties imply that the OID should not depend on any attribute values of the object, since the value of an attribute may be changed or corrected. It is also generally considered inappropriate to base the OID on the physical address of the object in storage, since the physical address can change after a physical reorganization of the database.

However, some systems do use the physical address as OID to increase the efficiency of object retrieval. If the physical address of the object changes, an *indirect pointer* can be placed at the former address, which gives the new physical location of the object. It is more common to use long integers as OIDs and then to use some form of hash table to map the OID value to the physical address of the object.

### object structure:

In OO databases, the state (current value) of a complex object may be constructed from other objects (or other values) by using certain **type constructors**. One formal way of representing such objects is to view each object as a triple  $(i, c, v)$ , where  $i$  is a unique *object identifier* (the OID),  $c$  is a *type constructor* (Note 12) (that is, an indication of how the object state is constructed), and  $v$  is the object state (or *current value*).

The data model will typically include several type constructors. The three most basic constructors are **atom**, **tuple**, and **set**. Other commonly used constructors include **list**, **bag**, and **array**. The atom constructor is used to represent all basic atomic values, such as integers, real numbers, character strings, Booleans, and any other basic data types that the system supports directly.

The object state  $v$  of an object  $(i, c, v)$  is interpreted based on the constructor  $c$ . If  $c = \text{atom}$ , the state (value)  $v$  is an atomic value from the domain of basic values supported by the system.

If  $c = \text{set}$ , the state  $v$  is a *set of object identifiers*, which are the OIDs for a set of objects that are typically of the same type.

If  $c = \text{tuple}$ , the state  $v$  is a tuple of the form  $\langle \dots \rangle$ , where each is an attribute name (Note 13) and each is an OID.

If  $c = \text{list}$ , the value  $v$  is an *ordered list* of OIDs of objects of the same type. A list is similar to a set except that the OIDs in a list are *ordered*, and hence we can refer to the first, second, or object in a list. For  $c = \text{array}$ , the state of the object is a single-dimensional array of object identifiers.

#### EXAMPLE 1: A Complex Object

We now represent some objects from the relational database shown in Figure 5.6, using the preceding model, where an object is defined by a triple (OID, type constructor, state) and the available type constructors are atom, set, and tuple. We use  $i_i$  to stand for unique system-generated object identifiers. Consider the following objects:

$o_1 = (i_1, \text{atom}, \text{'Houston'})$

$o_2 = (i_2, \text{atom}, \text{'Bellaire'})$

$o_3 = (i_3, \text{atom}, \text{'Sugarland'})$

$o_4 = (i_4, \text{atom}, 5)$

$o_5 = (i_5, \text{atom}, \text{'Research'})$

$o_6 = (i_6, \text{atom}, \text{'1988-05-22'})$

$o_7 = (i_7, \text{set}, \{i_1, i_2, i_3\})$

$o_8 = (i_8, \text{tuple}, \langle \text{DNAME:}i_5, \text{DNUMBER:}i_4, \text{MGR:}i_9, \text{LOCATIONS:}i_7, \text{EMPLOYEES:}i_{10}, \text{PROJECTS:}i_{11} \rangle)$

$o_9 = (i_9, \text{tuple}, \langle \text{MANAGER:}i_{12}, \text{MANAGER\_START\_DATE:}i_6 \rangle)$

$o_{10} = (i_{10}, \text{set}, \{i_{12}, i_{13}, i_{14}\})$

$o_{11} = (i_{11}, \text{set}, \{i_{15}, i_{16}, i_{17}\})$

$o_{12} = (i_{12}, \text{tuple}, \langle \text{FNAME:}i_{18}, \text{MINIT:}i_{19}, \text{LNAME:}i_{20}, \text{SSN:}i_{21}, \dots, \text{SALARY:}i_{26}, \text{SUPERVISOR:}i_{27}, \text{DEPT:}i_8 \rangle)$

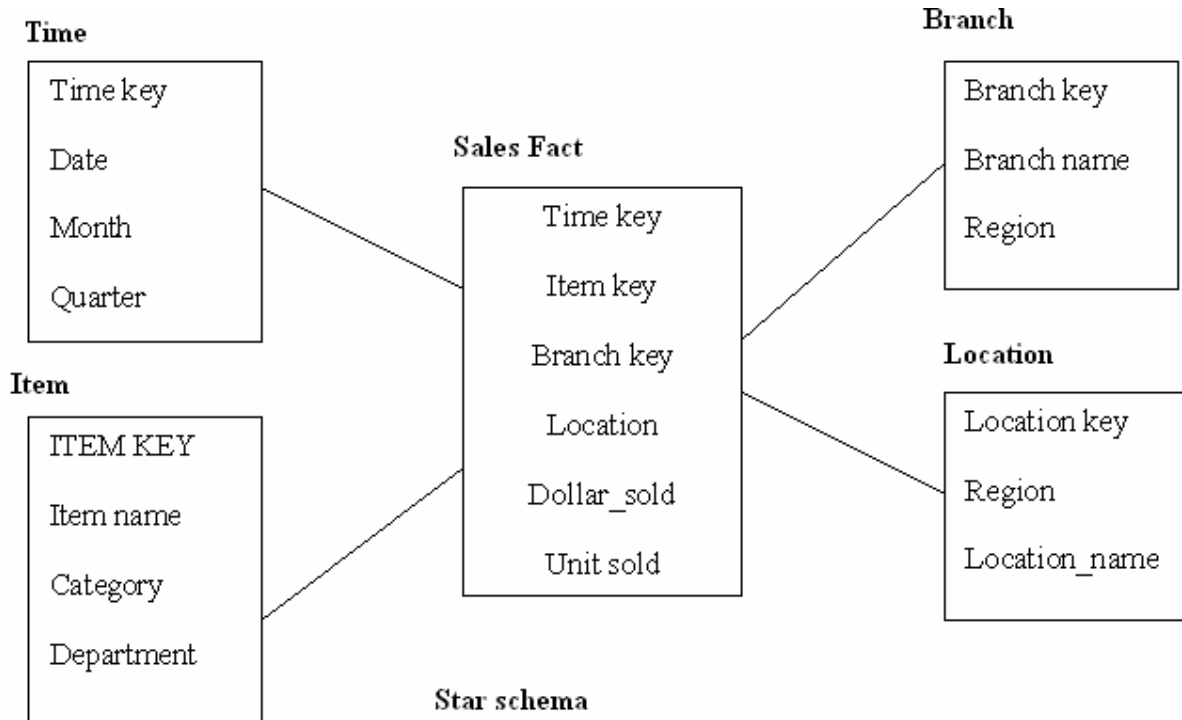
...

The first six objects ( $o_1$ - $o_6$ ) listed here represent atomic values. There will be many similar objects, one for each distinct constant atomic value in the database. Object  $o_7$  is a set-valued object that represents the set of locations for department 5; the set  $\{i_1, i_2, i_3\}$  refers to the atomic objects with values {'Houston', 'Bellaire', 'Sugarland'}. Object  $o_8$  is a tuple-valued object that represents department 5 itself, and has the attributes DNAME, DNUMBER, MGR, LOCATIONS, and so on.

5. (a) All electronics company have sales dept. Sales consider four dimensions namely time, item, branch & location. The schema contain a central fact table sales with two measures dollar\_sold & unit sold.

10  
Ans:

**Star Schema:**



The star schema consists of a fact tables with a single table for each dimensions.

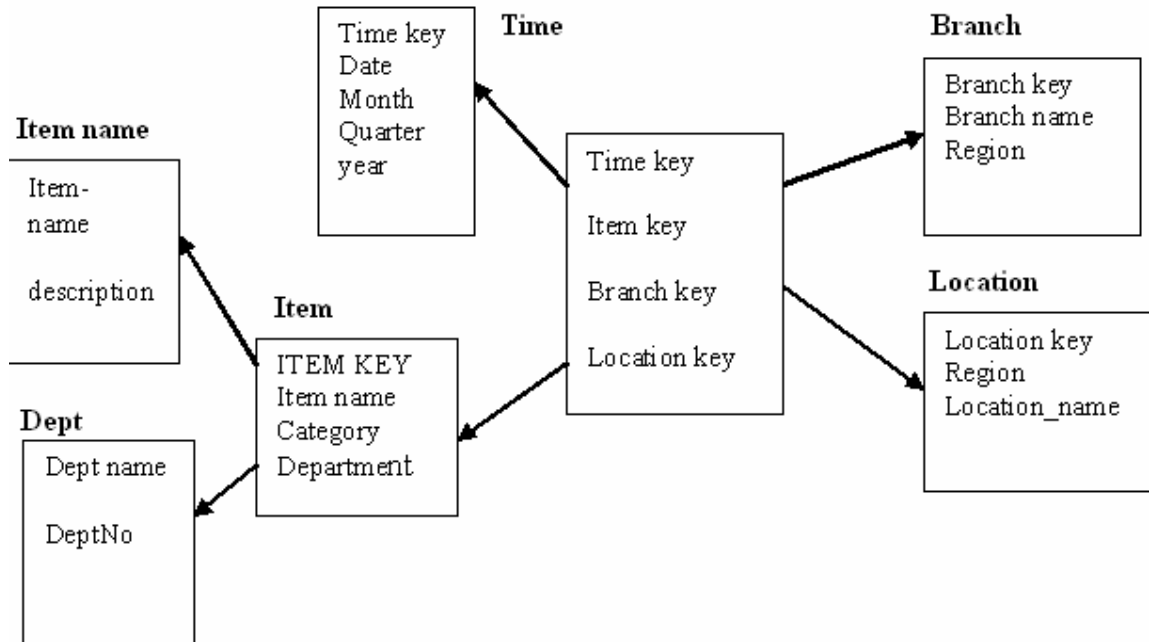
Four dimensions:

- Time X-axis
- Item Y-axis
- Branch Z-axis
- Locaton W-axis

- the fact table contains sales.

- Snowflake schema:

- It is a variations on the star schema with a dimensional tables from a star schema are organized into a hierarchy by normalizing them



(b) Consider a data warehouse for a hospital, where there are three dimensions. (1) Doctor (2) Patient & (3) Time & two measures: (1) Count & (2) Charge, where charge is the fee that the doctor charges a patient for a visit. Using the above example describe the following OLAP operations.

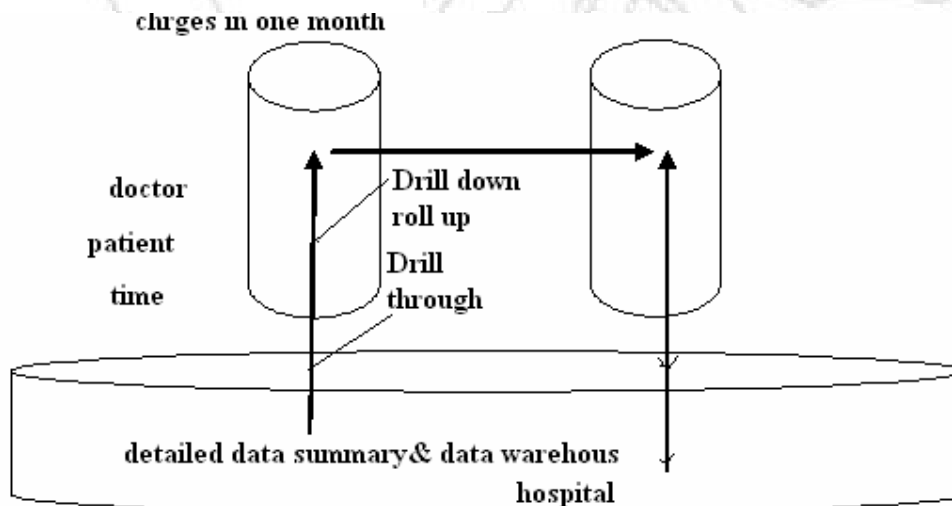
- (1) Slice (2) Dice (3) Rollup (4) Drilldown.

Ans:

With an OLAP system, a data analyst can look at different cross-tabs on the same data by interactively selecting the attributes in the cross-tab.

OLAP systems permit users to view data at any desired level of granularity. The operation of moving from finer-granularity data to a coarser granularity (by means of aggregation) is called a **rollup**.

In our example, starting from the data cube on the *sales* table, we got our example cross-tab by rolling up on the attribute *size*. The opposite operation—that of moving from coarser-granularity data to finer-granularity data—is called a **drill down**.

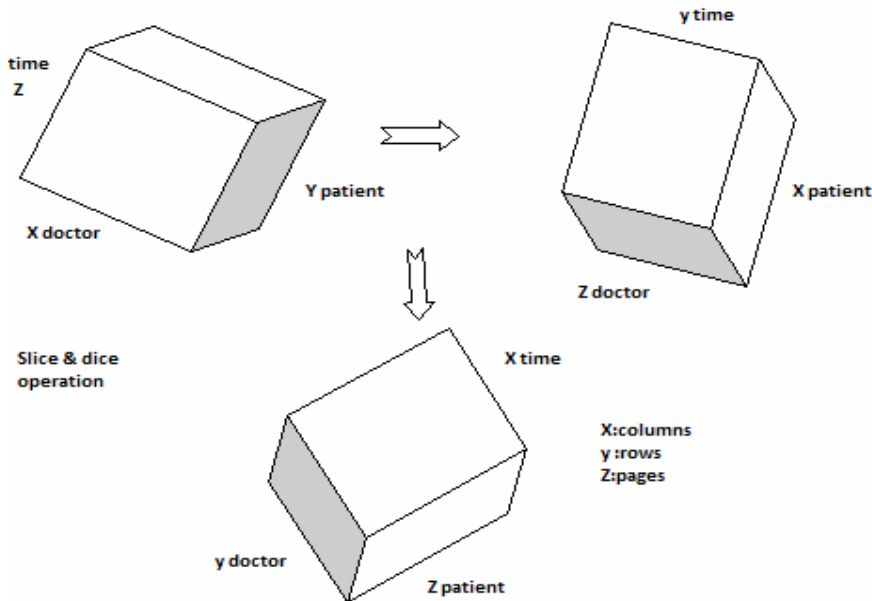


Three dimensions:

1. Patient
2. Doctor
3. Time

These attributes signifies as ascending hierarchical sequence from doctor to time.

As shown in fig. above, it shows the rolling up to higher hierarchical level of aggregation and drilling down to lower level details.



6. (a) Explain type of constraints with an example.

10

Ans:

**Integrity constraints** ensure that changes made to the database by authorized users do not result in a loss of data consistency.

**Domain constraints** specify the set of possible values that may be associated with an attribute. Such constraints may also prohibit the use of null values for particular attributes.

The **create domain** clause can be used to define new domains. For example, the statements:

```
create domain Dollars numeric(12,2)
```

```
create domain Pounds numeric(12,2)
```

define the domains *Dollars* and *Pounds* to be decimal numbers with a total of 12 digits, two of which are placed after the decimal point. An attempt to assign a value of type *Dollars* to a variable of type *Pounds* would result in a syntax error, although both are of the same numeric type

### Referential integrity

-A value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

**-Referential integrity** is a database constraint that ensures that references between data are indeed valid and intact.

-sometimes we wish to ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation, this condition is called referential integrity

-“Referential integrity in a relational database is consistency between coupled tables. Referential integrity is usually enforced by the combination of a primary key and a foreign key. For referential integrity to hold, any field in a table that is declared a foreign key can contain only values from a parent table's primary key field...”

- Foreign key can be specified as part of the SQL **Create table** statement by using the foreign key clause.

Example:

**Create table Employee (ename varchar (15), SSN int, address varchar (20),  
Primary key SSN);**

**Create table Department(Dname varchar(15),Dno int,Mgr\_ssn int,  
Primary key Dno,  
Foreign key (Mgr\_ssn) References Employee(SSN));**

-this foreign key declaration specifies that for each department, the Mgr\_ssn specified in the tuple must exist in the Employee relation.

There are many benefits of defining referential integrity in a database.

- **Improved data quality.** An obvious benefit is the boost to the quality of data that is stored in a database. There can still be errors, but at least data references are genuine and intact.
- **Faster development.** Referential integrity is declared. This is much more productive (one or two orders of magnitude) than writing custom programming code.
- **Consistency across applications.** Referential integrity ensures the quality of data references across the multiple application programs that may access a database.

*(b) Explain security & Authorization in SQL.*

10

Ans:

### **Security in SQL**

#### **Security Violations**

Among the forms of malicious access are:

- Unauthorized reading of data (theft of information)
- Unauthorized modification of data
- Unauthorized destruction of data

**Database security** refers to protection from malicious access. Absolute protection of the database from malicious abuse is not possible, but the cost to the perpetrator can be made high enough to deter most if not all attempts to access the database without proper authority.

To protect the database, we must take security measures at several levels:

- **Database system.** Some database-system users may be authorized to access only a limited portion of the database. Other users may be allowed to issue queries, but may be forbidden to modify the data. It is the responsibility of the database system to ensure that these authorization restrictions are not violated.
- **Operating system.** No matter how secure the database system is, weakness in operating-system security may serve as a means of unauthorized access to the database.
- **Network.** Since almost all database systems allow remote access through terminals or networks, software-level security within the network software is as important as physical security, both on the Internet and in private networks.
- **Physical.** Sites with computer systems must be physically secured against armed or surreptitious entry by intruders.
- **Human.** Users must be authorized carefully to reduce the chance of any user giving access to an intruder in exchange for a bribe or other favors.

### Authorization in SQL

We may assign a user several forms of authorization on parts of the database. For example,

- **Read authorization** allows reading, but not modification, of data.
- **Insert authorization** allows insertion of new data, but not modification of existing data.
- **Update authorization** allows modification, but not deletion, of data.
- **Delete authorization** allows deletion of data.

We may assign the user all, none, or a combination of these types of authorization.

In addition to these forms of authorization for access to data, we may grant a user authorization to modify the database schema:

- **Index authorization** allows the creation and deletion of indices.
- **Resource authorization** allows the creation of new relations.
- **Alteration authorization** allows the addition or deletion of attributes in a relation.
- **Drop authorization** allows the deletion of relations.

The **drop** and **delete** authorization differ in that **delete** authorization allows deletion of tuples only. If a user deletes all tuples of a relation, the relation still exists, but it is empty. If a relation is dropped, it no longer exists.

We regulate the ability to create new relations through **resource** authorization. A user with **resource** authorization who creates a new relation is given all privileges on that relation automatically.

**Index** authorization may appear unnecessary, since the creation or deletion of an index does not alter data in relations. Rather, indices are a structure for performance enhancements.

### Privileges in SQL

The SQL standard includes the privileges **delete**, **insert**, **select**, and **update**. The **select** privilege corresponds to the **read** privilege. SQL also includes a **references** privilege that permits a user/role to declare foreign keys when creating relations. If the relation to be created includes a foreign key that references attributes of another relation, the user/role must have been granted **references** privilege on those attributes.

The SQL data-definition language includes commands to grant and revoke privileges. The **grant** statement is used to confer authorization. The basic form of this statement is:

**grant** <privilege list> **on** <relation name or view name> **to** <user/role list>

The *privilege list* allows the granting of several privileges in one command.

The following **grant** statement grants users *U1*, *U2*, and *U3* **select** authorization on the *account* relation:

**grant select on account to U1, U2, U3**

The **update** authorization may be given either on all attributes of the relation or on only some. If **update** authorization is included in a **grant** statement, the list of attributes on which update authorization is to be granted optionally appears in parentheses immediately after the **update** keyword. If the list of attributes is omitted, the update privilege will be granted on all attributes of the relation.

This **grant** statement gives users *U1*, *U2*, and *U3* update authorization on the *amount*

attribute of the *loan* relation:

**grant update (amount) on loan to U1, U2, U3**

To revoke an authorization, we use the **revoke** statement. It takes a form almost identical to that of **grant**:

**revoke** <privilege list> **on** <relation name or view name>  
**from** <user/role list> [**restrict** | **cascade**]

7. Write short notes on (any four):-

- Comparison of RDBMS, OODBMS, ORDBMS
- SQL3 standard
- Comparison between OLTP and OLAP
- Two phase commit protocol
- Temporal databases.

Ans:

a) Comparison of RDBMS, OODBMS, ORDBMS

05

Type	RDBMS	OODBMS	ORDBMS
Maturity	Poor compared to OODBMS	Poor	Excellent
Usage	Known to internal structure of system	Easy & enable & users & to access databases	Easy if not used with ORDBMS extensions
Language	SQL	OQL	SQL with object oriented features
Object oriented support	No	Yes	Only for new data type
Advantage	Do not support complex application	Reusability of codeless coding, supports all complex application	supports complex & large application

b) SQL3 standard

05

The SQL standard

now includes the following parts: 1

- SQL/Framework, sQL/Foundation, SQL/Bindings, sQL/Object.
- New parts addressing temporal, transaction aspects of SQL.
- SQL/CLI (Call Level Interface).
- SQL/PSM (Persistent Stored Modules).

sQL/Foundation deals with new data types, new predicates, relational operations, cursors, rules and triggers, user-defined types, transaction capabilities, and stored routines.

SQL/CLI (Call Level Interface) provides rules that allow execution of

application code without providing source code and avoids the need for preprocessing. It contains about 50 routines for tasks such as connection to the SQL server, allocating and deallocating resources, obtaining diagnostic and implementation information, and

controlling termination of transactions. SQL/PSM (Persistent Stored Modules) specifies facilities for partitioning an application between a client and a server. The goal is to enhance performance by minimizing network traffic. SQL/Bindings includes Embedded SQL and Direct Invocation. Embedded SQL has been enhanced to include additional exception declarations.

The SQL/Object specification extends sQL-92 to include object-oriented capabilities. The following are some of the features that have been included in sQL-99:

- Some type constructors have been added to specify complex objects. These include the *row type*, An *array type* for specifying collections is also provided. Other collection type constructors, such as set, list, and bag constructors, are not yet part of the sQL-99 specifications, although some systems include them and they are expected to be in future versions of the standard.
- A mechanism for specifying object identity through the use of *reference type* is included.
- Encapsulation of operations is provided through the mechanism of user-defined types that may include operations as part of their declaration.
- Inheritance mechanisms are provided.

c) *Comparison between OLTP and OLAP* 05

Features	OLTP(online Transaction processing)	OLAP(online Analysis processing)
1)Characteristics	Operational processing	Information processing
2)Orientation	Transaction	Analysis
3)Users	Clerk, DBA, db professionals.	Knowledge worker(manager)
4)Function	Day to day operations.	Decision support, long term information requirement.
5)DB design	ER based ,Application oriented	Multidimensional models
6)Data	Current	historical
7)Summarization	Primitive, highly detailed	Summarized
8)View	Detailed, flat relational	Multidimensional
9)Unit of work	Simple transaction	Complex queries
10)Access	Read/write	Read(mostly)
11)Focus	Data in	Information out
12)No. of records accessed	Tens	Millions
13)No. of users	Thousands	Hundreds
14)Size	100 MB to GB	100 GB to TB

d) *Two phase commit protocol* 05

- 2 phase commit protocol is often used to ensure the correctness of distributed commit
- 2PC includes:
  - Voting phase
  - Termination phase
- Steps taken during normal execution -coordination  
-subordination
- When user decides to commit transaction, the commit command is sent to coordinator. This initiates the 2PC protocol.
  - 1) Coordinator sends a prepare message to each subordinator.
  - 2) When a subordinator receive a prepare message it decide whether to abort or commit, its subordinator sends a no. or yes message to Coordinator.
  - 3) When Coordinator receives yes message from subordinator it writes commit log record & sends commit message to all subordinates.  
→when it receive no msg.  
It writes an abort log record & sends abort msg. to all.

- 4) When a subordinate receives an abort msg. it writes abort log record & send acknowledgement msg. to coordinator, abort subtransactions.
- 5) When coordinator receives acknowledgement msg. from all subordinators, it writes end message record for the transactions.

e) *Temporal databases.*

05

- Databases that store information about states of the real world across time are called **temporal databases**.

**Temporal databases** permit the database system to store a history of changes, and allow users to query both current and past states of the database. Some temporal database models also allow users to store future expected information, such as planned schedules.

Temporal database store time related data .It usually stores relational data that includes time related attributes.

These attributes may involve several stamps, each having different semantics.

Example:

- Healthcare- where patient histories need to be maintained.
- Insurance- where claim & accident histories are required as well as information about time when insurance policies are in effect.
- Reservation system- where information on the data & times when reservation are in required.

### Time Specification in SQL

The SQL standard defines the types **date**, **time**, and **timestamp**. The type **date** contains four digits for the year (1–9999), two digits for the month (1–12), and two digits for the date (1–31).

The type **time** contains two digits for the hour, two digits for the minute, and two digits for the second, plus optional fractional digits.

The seconds field can go beyond 60, to allow for leap seconds that are added during some years to correct for small variations in the speed of rotation of Earth. The type **timestamp** contains the fields of **date** and **time**, with six fractional digits for the seconds field.

SQL supports a type called **interval**, which allows us to refer to a period of time such as “1 day” or “2 days and 5 hours,” without specifying a particular time when this period starts

### References:

- 1) **Ramakrishnan - Database Management Systems 3rd Edition**
- 2) **Fundamentals of Database Systems(Elmasri,Navathe)**