

B. E.
Fourth Semester Examination, Dec, 2008
DATABASE MANAGEMENT SYSTEM

Note : Attempt any five questions. All questions carry equal marks.

Q. 1. (a) Differentiate between

- (i) Procedural and non-procedural DML.**
- (ii) Logical and physical data independence.**

Ans. (i) Procedural and Non-procedural DML

- * **Procedural DML.**
 - Allows user to tell system how to manipulate data.
- * **Non-Procedural DML.**
 - Allows user to state what data is needed rather than how it is to be retrieved.
- * **High Level or Non-procedural Language :**
 - For example, the SQL relational language.
 - May be used standalone or embedded in a PL.
 - Are "set"-oriented and specify what data to retrieve rather than how to retrieve it.
 - Also called declarative languages.
- * **Low Level or Procedural Language :**
 - Retrieve data one record-at-a-time;
 - Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

Ans. (ii) Logical and physical data independence :

- * **Logical Data Independence.**
 - Refers to immunity of external schemas to changes in conceptual schema.
 - Should not require changes to external schema or rewrites of application programs.
- * **Physical Data Independence.**
 - Refers to immunity of conceptual schema to changes in the internal schema.
 - Should not require change to conceptual or external schemas.
- * **Only mappings between schemas need to be changed.**

Q. 1. (b) What are the responsibilities of DBA? Discuss.

Ans. Responsibilities of DBA :

Responsibility of DBA: Some of the more advanced duties of the Oracle DBA might include the following

*** Data analysis :**

The DBA will frequently be called on to analyze the data stored in the database and to make recommendations relating to performance and efficiency of that data storage. This might relate to the more effective use of indexes or the use of some feature such as the Parallel Query option.

*** Database design (preliminary) :**

The DBA is often involved at the preliminary database-design stages. Through the involvement of the DBA, many problems that might occur can be eliminated. The DBA, knowing the DBMS and system, can point out potential problems, and can help the development team with special performance considerations.

*** Data modeling and optimization :**

By modeling the data, it is possible to optimize the system layout to take the most advantage of your I/O subsystem.

*** Assisting developers with SQL and stored procedure development :**

The DBA should be prepared to be a resource for developers and users. The DBA is often called on to help with SQL problems as well as to design and write stored procedures.

*** Enterprise standards and naming conventions :**

Because many different groups might perform different roles in developing and deploying applications, it is often the DBA who is called on to help define enterprise standards and naming conventions as well as to ensure that new applications are conforming to these standards.

*** Development of production migration procedures :**

Because the DBA is responsible for the availability and reliability of the DBMS and applications using that DBMS, it is up to the DBA to develop and maintain procedures for rolling out new applications and DBMS software. This involves evaluating new software or patches as well as testing them. It is up to the DBA to guarantee the stability and robustness of the system.

*** Environmental documentation :**

The DBA should document every aspect of the DBMS environment, including hardware configuration and maintenance records, software updates, changes to the applications and DBMS, and all other items related to changes made to the system. The DBA should be able to access these records and fully reproduce the current system as necessary.

*** Consult with development team and end users :**

The DBA is often called on to act as a consultant to the development team as well as to the user community. This might involve personally assisting a single user or developing training courses for the user community as a whole.

*** Evaluation of new software :**

The DBA might be called on to evaluate new software and make recommendations based on that evaluation. This might be related to a software purchase or a scheduled roll out of a new version of software. This evaluation must be done in the context of the stability of the system. It is your responsibility to maintain system stability and reliability.

*** Evaluation of new hardware and software purchases :**

There is much consideration involved in purchasing new hardware and software. Much of this consideration involves the functionality and compatibility of the software or hardware as well as the cost of these components. Although the cost of the item is not usually a concern of the DBA, the function away and compatibility is. The DBA might be asked to make recommendations based on whether these purchases make sense.

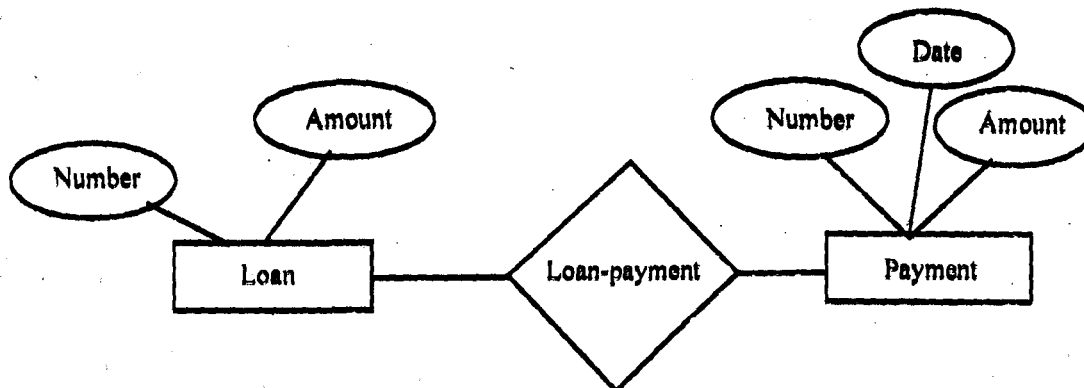
*** Capacity planning and sizing :**

Determining whether it is necessary to purchase new hardware or software to meet increased loads is often a job for the DBA. Capacity planning and sizing is important to provide the level of service your users require. By anticipating the future needs of your users, you can provide an excellent level of service with no interruptions.

Q. 2. (a) What is a weak entity? How can it be represented in ER model? Give examples.

Ans. Weak entity :

A weak entity is an entity that exists only if it is related to a set of uniquely determined entities, which are called the owners of the weak entity. For instance, we could extend our library with a weak entity type edition; each book has several editions, and certainly it is nonsense to speak about an edition if this does not happen in the context of a specific book. From a user interface viewpoint, weak entities are usually edited in the context of (one of) their owners. When an entity is deleted from a schema instance, all owned weak entities are deleted, too. We shall call the type of a weak entity a weak entity type.



For entities of type W to be owned by entities of type X, a requirement must be satisfied : there must be

an identifying function from W to X that specifies the owner of each entity of type W, that is, a relationship type going from W to X whose cardinality constraint impose its instances to be functions. Deletion of an entity of type X implies deletion of all related entities of W.

Weak entity set is non-existent is the corresponding strong entity set is removed.

Q. 2. (b) Define the following terms :

superkey, candidate key, foreign key, primary key.

Ans. Super key :

A superkey is defined in the relational model of database organisation as a set of attributes of a relation variable (reveler) for which it holds that in all relations assigned to that variable there are no two distinct tuples (rows) that have the same values for the attributes in this set. Equivalently a superkey can also be defined as a set of attributes of a relvar upon which all attributes of the relvar are functionally dependent.

Note that if attribute set K is a superkey of relvar R, then at all times it is the case that the projection of Rover K has the same cardinality as R itself.

Informally, a superkey is a set of columns within a table whose values can be used to uniquely identify a row. A candidate key is a minimal set of columns necessary to identify a row, this is also called a minimal superkey. For example, given an employee table, consisting of the columns employeeID, name, job, and departmentID, we could use the employeeID in combination with any or all other columns of this table to uniquely identify a row in the table. Examples of superkeys in this table would be {employeeID, Name}, {employeeID, Name, job}, and {employeeID, Name, job, departmentID}.

Candidate key: :

In the relational model, a candidate key of a relvar (relation variable) is a set of attributes of that relvar such that

1. At all times it holds in the relation assigned to that variable that there are no two distinct tuples with the same values for these attributes and
2. There is not a proper subset for which (1) holds.

Since a superkey is defined as a set of attributes for which (1) holds, we can also define a candidate key as a minimal superkey, i.e., a superkey of which no proper subset is also a superkey.

The importance of candidate keys is that they tell us how we can identify individual tuples in a relation. As such they are one of the most important types of database constraint that should be specified when designing a database schema. Since a relation is a set (no duplicate elements), it holds that every relation will have at least one candidate key (because the entire heading is always a superkey).

Foreign key :

In the context of relational databases, a foreign key is a referential constraint between two tables. The foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table. The columns in the referencing table must be the primary key or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that don't exist in the

referenced table (except potentially NULL). This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table. Most of the time, it reflects the one (master table, or referenced table) to many (child table, or referencing table) relationship.

Primary key: :

In relational database design, a unique key or primary key is a candidate key to uniquely identify each row in a table. A unique key or primary key comprises a single column or set of columns. No two distinct rows in a table can have the same value (or combination of values) in those columns. Depending on its design, a table may have arbitrarily many unique keys but at most one primary key. A unique key must uniquely identify all possible rows that exist in a table and not only the currently existing rows. Examples of unique keys are Social Security numbers (associated with a specific person) or ISBNs (associated with a specific book). Telephone books and dictionaries cannot use names or words or Dewey Decimal system numbers as candidate keys because they do not uniquely identify telephone numbers or words. A primary key is a special case of unique keys. The major difference is that for unique keys the implicit NOT NULL constraint is not automatically enforced, while for primary keys it is. Thus, the values in a unique key columns may or may not be NULL. Another difference is that primary keys must be defined using another syntax.

Q. 3. (a) Differentiate between sequential and index sequential files.

Ans. Sequential files :

Sequential file sorted in alphabetical order. Sequential files are usually sorted in ID sequence order to facilitate batch processing. Sequential files must be recopied from the point of any insertion or deletion to the end of the file. They are commonly used in batch processing where a new master file will be generated each time the file is updated.

Indexed sequential files :

Each record of a file has a key field which uniquely identifies that record.

An index consists of keys and addresses (physical disc locations).

An indexed sequential file is a sequential file (i.e. sorted into order of a key field) which has an index.

A full index to a file is one in which there is an entry for every record.

Indexed sequential files are important for applications where data needs to be accessed.....

- * Sequentially.

- * Randomly using the index.

An indexed sequential file allows fast access to a specific record.

Example :

A company may store details about its employees as an indexed sequential file. Sometimes the file is accessed...

- * Sequentially :

For example when the whole of the file is processed to produce pay slips at the end of the month.

*** Randomly :**

May be an employee changes address, or a female employee gets married and changes her surname.

An indexed sequential file can only be stored on a random access device e.g. magnetic disc, CD.

Q. 3. (b) What is hashing? What is it used for? How?

Ans. Hashing : Hash File Organization :

1. Hashing involves computing the address of a data item by computing a function on the search key value.
 2. A hash function h is a function from the set of all search key values K to the set of all bucket addresses B .
- * We choose a number of buckets to correspond to the number of search key values we will have stored in the database.
 - * To perform a lookup on a search key value K_i , we compute $h(K_i)$ and search the bucket with that address.
 - * If two search keys i and j map to the same address, because $h(K_i) = h(K_j)$, then the bucket at the address obtained will contain records with both search key values.
 - * In this case we will have to check the search key value of every record in the bucket to get the ones we want.
 - * Insertion and deletion are simple.

Applications :

Hash functions are mostly used in hash tables, to quickly locate a data record (for example, a dictionary definition) given its search key (the headword). Specifically, the hash function is used to map the search key to the index of a slot in the table where the corresponding record is supposedly stored.

In general, a hashing function may map several different keys to the same hash value. Therefore, each slot of a hash table contains (implicitly or explicitly) a set of records, rather than a single record. For this reason, each slot of a hash table is often called a bucket, and hash values are also called bucket indices.

Thus, the hash function only hints at the record's location - it only tells where one should start looking for it. Still, in a half-full table, a good hash function will typically narrow the search down to only one or two entries.

- Finding duplicate records.
- Finding similar records.
- Finding similar substrings .
- Geometric hashing.

Q. 4. (a) What is meant by union compatibility? Discuss various set-theoretic operators relational algebra.

Ans. Union compatibility :

For operators \cup , \cap , or $-$, the operand relations $R_1(A_1, A_2, \dots, A_n)$ and $R_2(B_1, B_2, \dots, B_n)$ must have the same number of attributes, and the domains of the corresponding attributes must be compatible; that is, $\text{dom}(A_i) = \text{dom}(B_i)$ for $i=1, 2, \dots, n$.

The resulting relation for \cup , \cap , or $-$ has the same attribute names as the first operand relation R_1 (by convention).

- Are $S(SNO, SNAME, AGE, STATE)$ and $C(CNO, CNAME, CREDIT, DEPT)$ union compatible?
- Are $S(S\#, SNAME, AGE, STATE)$ and $C(CNO, CNAME, CREDIT_HOURS, DEPT_NAME)$ union compatible?

The Relational Union

| | |
|----------|----------|
| r | |
| A | B |
| a | 1 |
| a | 2 |
| b | 1 |

| | |
|----------|----------|
| s | |
| A | B |
| a | 2 |
| b | 3 |

| | |
|------------------------------|----------|
| r \cup s | |
| A | B |
| a | 1 |
| a | 2 |
| b | 1 |
| b | 3 |

Q. 4. (b) What is tuple calculus? What are safe expressions? Give examples.

Ans. Tuple calculus :

The tuple calculus is a calculus that was introduced by Edgar F. Codd as part of the relational model in order to give a declarative database query language for this data model. It formed the inspiration for the database query languages QUEL and SQL of which the latter, although far less faithful to the original relational model and calculus, is now used in almost all relational database management systems as the ad-hoc query language. Lacroix and Pirotte proposed domain calculus, which is closer to first-order logic and showed that these two calculi (and the relational algebra) are equivalent in expressive power. Subsequently query languages for the relational model were called relationally complete if they could express at least all these queries.

1. The tuple relational calculus is a nonprocedural language. (The relational algebra was procedural.)

We must provide a formal description of the information desired.

2. A query in the tuple relational calculus is expressed as

$$\{t \mid P(t)\}$$

i.e. the set of tuples for which predicate P is true.

3. We also use the notation.

- $t[a]$ to indicate the value of tuple t on attribute a .

- $t \in r$ to show that tuple t is in relation r .

4. For example, to find the branch-name, loan number, customer name and amount for loans over \$1200:

$$\{t \mid t \in \text{borrow} \wedge t[\text{amount}] > 1200\}$$

This gives us all attributes, but suppose we only want the customer names. (We would use project in the algebra.)

We need to write an expression for a relation on scheme (cname).

$$\{t \mid \exists s \in \text{borrow} (t[\text{cname}] = s[\text{cname}] \wedge s[\text{amount}] > 1200)\}$$

In English, we may read this equation as "the set of all tuples such that there exists a tuple s in the relation borrow for which the values of t and s for the cname attribute are equal, and the value of s for the amount attribute is greater than 1200."

The notation $\exists t \in r(Q(t))$ means "there exists a tuple t in relation r such that predicate $Q(t)$ is true."

How did we get the above expression? We needed tuples on scheme cname such that there were tuples in borrow pertaining to that customer name with amount attribute > 1200.

The tuples t get the scheme cname implicitly as that is the only attribute t is mentioned with.

Let's look at a more complex example.

Find all customers having a loan from the SFU branch and the cities in which they live :

$$\left\{ t \mid \exists s \in \text{borrow} \left(\begin{array}{l} t[\text{cname}] = s[\text{cname}] \wedge s[\text{bname}] = \text{"SFU"} \wedge \exists u \in \text{customer} \\ (u[\text{cname}] = s[\text{cname}] \wedge t[\text{ccity}] = u[\text{ccity}]) \end{array} \right) \right\}$$

In English, we might read this as "the set of all (cname, ccity) tuples for which cname is a borrower at the SFU branch, and ccity is the city of cname".

Tuple variable s ensures that the customer is a borrower at the SFU branch.

Tuple variable u is restricted to pertain to the same customer as s , and also ensures that ccity is the city

of the customer.

The logical connectives \wedge (AND) and \vee (OR) are allowed, as well as \neg (negation).

We also use the existential quantifier \exists and the universal quantifier \forall .

Q. 5. (a) Differentiate between BCNF and 3NF.

Ans. BCNF :

- * We say a relation R is in BCNF if whenever $X \rightarrow A$ is a nontrivial FD that holds in R, X is a superkey.
- * Remember : nontrivial means A is not a member of set X.
- * Remember, a superkey is any superset of a key (not necessarily a proper superset)

Example : Drinkers(name, addr, beersLiked, manf, favBeer)

FD's : name \rightarrow addr favBeer, beersLiked \rightarrow manf

- * Only key is {name, beersLiked}.
- * In each FD, the left side is not a superkey.
- * Anyone of these FD's shows Drinkers is not in BCNF

Decomposition into BCNF :

- * Given : relation R with FD's F.
- * Look among the given FD's for a BCNF violation $X \rightarrow B$.
- If any FD following from F violates BCNF, then there will surely be an FD in F itself that violates BCNF.
- * Compute X +.
- Not all attributes, or else X is a superkey.

3NF : The third normal form (3NF) is a normal form used in database normalization. 3NF was originally defined by E.F. Codd in 1971. Codd's definition states that a table is in 3NF if and only if both of the following conditions hold :

- * The relation R (table) is in second normal form (2NF).
- * Every non-prime attribute of R is non-transitively dependent (i.e. directly dependent) on every key of R.

A non-prime attribute of R is an attribute that does not belong to any candidate key of R. A transitive dependency is a functional dependency in which $X \rightarrow Z$ (X determines Z) indirectly, by virtue of $X \rightarrow Y$ and $Y \rightarrow Z$ (where it is not the case that $Y \rightarrow X$).

A 3NF definition that is equivalent to Codd's, but expressed differently, was given by Carlo Zaniolo in 1982. This definition states that a table is in 3NF if and only if, for each of its functional dependencies $X \twoheadrightarrow A$, at least one of the following conditions holds :

* X contains A (that is, $X \rightarrow A$ is trivial functional dependency), or X is a superkey, or A is a prime attribute (i.e., A is contained within a candidate key)

Q. 5. (b) What are MVDS? What is 4NF? Discuss.

Ans. 4NF :

Fourth normal form (4NF) is a normal form used in database normalization. Introduced by Ronald Fagin in 1977, 4NF is the next level of normalization after Boyce-Codd normal form (BCNF). Whereas the second, third, and Boyce-Codd normal forms are concerned with functional dependencies, 4NF is concerned with a more general type of dependency known as a multivalued dependency. A table is in 4NF if and only if, for every one of its non-trivial multivalued dependencies $X \twoheadrightarrow Y$, X is a superkey—that is, X is either a candidate key or a superset thereof.

Multivalued dependencies :

If the column headings in a relational database table are divided into three disjoint groupings X, Y, and Z, then, in the context of a particular row, we can refer to the data beneath each group of headings as x, y, and z respectively. A multivalued dependency $X \twoheadrightarrow Y$ signifies that if we choose any x actually occurring in the table (call this choice xc), and compile a list of all the xcyz combinations that occur in the table, we will find that X_c is associated with the same y entries regardless of z.

A trivial multivalued dependency $X \twoheadrightarrow Y$ is one in which Y consists of all columns not belonging to X. That is, a subset of attributes in a table has a trivial multivalued dependency on the remaining subset of attributes.

A functional dependency is a special case of multivalued dependency. In a functional dependency $X \rightarrow Y$, every x determines exactly one y, never more than one.

Q. 6. (a) Explain the following commands of SQL insert, create, drop table, update.

Ans. Insert :

An INSERT statement is used to add one or more new records to a table. This is how you populated a table. Each record is subdivided into fields and each field contains a single piece of data.

Syntax :

```
INSERT INTO [ MEMORY ] ["database_file_name".] table_name [PASSWORD database_password]
[(column_name [, ...]) ] {VALUES ( expression [, ...]) | query}
```

```
INSERT INTO developers (code, name) VALUES (5, 'Bob');
```

Create :

This command is used to create a table. The syntax of this command is :

```
create table tablename
```

```
(column1name datatype [constraint], column2name datatype [constraint], column3name datatype [constraint]);
```

create table citylist (name varchar(20), state varchar(20), population number(8), zipcode number(5) unique);

Drop :

Used to remove an entire table from the database. Syntax :

drop table tablename

Example :

drop table citylist;

Update :

This command is used to make changes to records in tables. Syntax:

Update tablename set columnname = newvalue [,columnxname = newvaluex...]

where columnname OPERATOR value [and or columnnamex OPERATOR value x];

The OPERATOR is one of the conditions listed on the Select Command page.

OPERATORS include :

- * = - Equal
- * < - Less than
- * > - Greater than
- * <= - Less than or equal
- * >= - Greater than or equal
- * <> - Not equal
- * **LIKE** : Allows the wildcard operator, %, to be used to select items that are a partial match. An example is :

select city, state from towntable where state like 'north%'

This allows selection of all towns in states that begin with the word "north" allowing states like North Dakota and North Carolina to be selected.

Example :

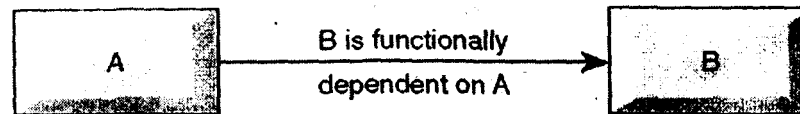
update citylist
set population = population+1
where name = 'Argos' and
state = 'Indiana';

Q. 6. (b) What are f.d.s? Explain inference rules for f.d.s.

Ans. F.D.S :

- * Describes the relationship between attributes in a relation.

- * For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted $A \rightarrow B$), if each value of A in R is associated with exactly one value of B in R.
- * Functional dependency is a property of the meaning or semantics of the attributes in a relation.
- * The determinant of a functional dependency refers to the attribute or group of attributes on the left-hand side of the arrow.
- * Diagrammatic representation.



- * From the semantics of the attributes, we know the following functional dependency should hold.

$SSN \rightarrow Ename$

- * $Pnumber \rightarrow \{Pname, Location\}$
- * $\{SSN, Pnumber\} \rightarrow Hours$

| | | | | | | |
|----------|-----|---------|-------|-------|-------|-----------|
| EMP_PROJ | SSN | Pnumber | Hours | Ename | Pname | Plocation |
|----------|-----|---------|-------|-------|-------|-----------|

- * A function dependency is a property of the relation schema (intension) R, not of a particular legal relation state (extension) of R.
- * The figure below show a particular state for the TEACH relation.

Inference rules.

Consider the relation schema EMP_PROJ in the figure below :

| | | | | | | |
|----------|-------|-----|-------|---------|---------|---------|
| EMP_PROJ | Fname | SSN | BDate | Address | Dnumber | DMGRSSN |
|----------|-------|-----|-------|---------|---------|---------|

FD1 FD2

- * We can infer the following addition FDs from F:
- * $FD1 = \{SSN \rightarrow \{Ename, BDate, Address, Dnumber\}\}$
- * $FD2 = \{Dnumber \rightarrow \{Dname, DMGRSSN\}\}$
- * $FD3 = \{SSN \rightarrow \{Dname, DMGRSSN\}\}$
- * $FD4 = \{SSN \rightarrow SSN\}$
- * $FD5 = \{Dnumber \rightarrow Dname\}$
- * An FD $X \rightarrow Y$ is inferred from a set of dependencies F specified on R if $X \rightarrow Y$ holds in every relation state r that is a legal extension of R; that is whenever r satisfies all the dependencies in F, $X \rightarrow Y$ also

holds in r.

- * The closure of F is the set of all functional dependencies that can be inferred from F.
- * A set of inference rules that can be used to infer new dependencies from a given set of dependencies.
- * The following six rules (IR1 through IR6) are well-known inference rules for FDs.
- * IR1 - Reflective rule: $Y \subseteq X$ then $X \rightarrow Y$.
- * IR2 - Augmentation rule: $X \rightarrow Y$, then $XZ \rightarrow YZ$.
- * IR3 - Transitive rule: $\{X \rightarrow Y, Y \rightarrow Z\}$ then $X \rightarrow Z$.
- * IR4 - Decomposition or projective, rule: $\{X \rightarrow YZ\}$ then $X \rightarrow Z$.
- * IR5 - Union or additive, rule: $\{X \rightarrow Y, X \rightarrow Z\}$ then $X \rightarrow YZ$.
- * IR6 - Pseudo transitive rule: $\{X \rightarrow Y, WY \rightarrow Z\}$ then $WX \rightarrow Z$.

Q. 7. (a) What are ACID properties of a transaction?

Ans. ACID properties :

ACID properties are an important concept for databases. The acronym stands for Atomicity, Consistency, Isolation, and Durability.

The ACID properties of a DBMS allow safe sharing of data. Without these ACID properties, everyday occurrences such as using computer systems to buy products would be difficult and the potential for inaccuracy would be huge. Imagine more than one person trying to buy the same size and color of a sweater at the same time -- a regular occurrence. The ACID properties make it possible for the merchant to keep these sweater purchasing transactions from overlapping each other -- saving the merchant from erroneous inventory and account balances.

Atomicity :

The phrase "all or nothing" succinctly describes the first ACID property of atomicity. When an update occurs to a database, either all or none of the update becomes available to anyone beyond the user or application performing the update. This update to the database is called a transaction and it either commits or aborts. This means that only a fragment of the update cannot be placed into the database, should a problem occur with either the hardware or the software involved. Features to consider for atomicity :

- a transaction is a unit of operation - either all the transaction's actions are completed or none are
- atomicity is maintained in the presence of deadlocks.
- atomicity is maintained in the presence of database software failures.
- atomicity is maintained in the presence of application software failures.
- atomicity is maintained in the presence of CPU failures.
- atomicity is maintained in the presence of disk failures.

- atomicity can be turned off at the system level.
- atomicity can be turned off at the session level.

Consistency :

Consistency is the ACID property that ensures that any changes to values in an instance are consistent with changes to other values in the same instance. A consistency constraint is a predicate on data which server as a precondition, post-condition, and transformation condition on any transaction.

Isolation :

The isolation portion of the ACID properties is needed when there are concurrent transactions. Concurrent transactions are transactions that occur at the same time, such as shared multiple users accessing shared objects.. This situation is illustrated at the top of the figure as activities occurring over time. The safeguards used by a DBMS to prevent conflicts between concurrent transactions are a concept referred to as isolation.

Durability :

Maintaining updates of committed transactions is critical. These updates must never be lost. The ACID property of durability addresses this need. Durability refers to the ability of the system to recover committed transaction updates if either the system or the storage media fails. Features to consider for durability :

- * Recovery to the most recent successful commit after a database software failure.
- * Recovery to the most recent successful commit after an application software failure.
- * Recovery to the most recent successful commit after a CPU failure.
- * Recovery to the most recent successful backup after a disk failure.
- * Recovery to the most recent successful commit after a data disk failure.

Q. 7. (b) What are parallel databases? What are they used for?

Ans. Parallel databases :

A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries. Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel. Centralized and client-server database systems are not powerful enough to handle such applications. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequential.

- * Parallel machines are becoming quite common and affordable.
 - Prices of microprocessors, memory and disks have dropped sharply.
 - Recent desktop computers feature multiple processors and this trend is projected to accelerate.
- * Databases are growing increasingly large.
 - Large volumes of transaction data are collected and stored for later analysis.

- Multimedia objects like images are increasingly stored in databases.
- * Large-scale parallel database systems increasingly used for :
 - Storing large volumes of data.
 - Processing time-consuming decision-support queries.
 - Providing high throughput (or transaction processing

Interquery parallelism :

- * Queries/transactions execute in parallel with one another.
 - Increases transaction throughput; used primarily to scale up a transaction processing system to support a larger number of transactions per second.
- * Easiest form of parallelism to support, particularly in a shared-memory parallel database, because even sequential database systems support concurrent processing.
- * More complicated to implement on shared-disk or shared-nothing architectures.
 - Locking and logging must be coordinated by passing messages between processors.
 - Data in a local buffer may have been updated at another processor.
 - Cache-coherency has to be maintained-reads and writes of data in buffer must find latest version of data.

Intra query parallelism :

- * Execution of a single query in parallel on multiple processors/disks; important for speeding up long-running queries.
- * Two complementary forms of intraquery parallelism :
 - Intraoperation Parallelism - parallelize the execution of each individual operation in the query.
 - Interoperation Parallelism - execute the different operations in a query expression in parallel.

The first form scales better with increasing parallelism because the number of tuples processed by each operation is typically more than the number of operations in a query.

Inter operator parallelism :

- * Pipelined parallelism.
 - Consider a join of four relations
- * $r1 \text{ } r2 \text{ } r3 \text{ } r4$
- * Set up a pipeline that computes the three joins in parallel
- * Let P1 be assigned the computation of
temp 1 = $r1 \text{ } r2$.
- * And P2 be assigned the computation of temp 2 = temp 1 r3

- * And P3 be assigned the computation of temp 2 r4.
- * Each of these operations can execute in parallel, sending result tuples it computes to the next operation even as it is computing further results.
- * Provided a pipelineable join evaluation algorithm (e.g., indexed nested loops join) is used.

Q. 8. Write short notes on the following :

- (a) **Data warehousing**
- (b) **Data Models**
- (c) **Distributed Databases.**

Ans. Dataware housing :

Data warehouse is a repository of an organization's electronically stored data. Data warehouses are designed to facilitate reporting and analysis. This Classic or general definition of the data warehouse focuses on data storage. However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform, and load data into the repository, and tools to manage and retrieve metadata. In contrast to data warehouses are operational systems which perform day-to-day transaction processing. The process of transforming data into information and making it available to the user in a timely enough manner to make a difference is known as data warehousing. Architecture, in the context of an organization's data warehousing efforts, is a conceptualization of how the data warehouse is built. There is no right or wrong architecture. The worthiness of the architecture can be judged in how the conceptualization aids in the building, maintenance, and usage of the data warehouse. One possible simple conceptualization of a data warehouse architecture consists of the following interconnected layers :

Operational database layer :

The source data for the data warehouse - An organization's EIS systems fall into this layer.

Informational access layer.

The data accessed for reporting and analyzing and the tools for reporting and analyzing data - Business intelligence tools fall into this layer. And the Inmon-Kimball differences about design methodology, discussed later in this article, have to do with this layer.

Data access layer :

The interface between the operational and informational access layer - Tools to extract, transform, load data into the warehouse fall into this layer.

Metadata layer :

The data directory : This is often usually more detailed than an operational system data directory. There are dictionaries for the entire warehouse and sometimes dictionaries for the data that can be accessed by a particular reporting and analysis tool.

Inmon states that the data warehouse is :

Subject-oriented.

The data in the data warehouse is organized so that all the data elements relating to the same real-world event or object are linked together.

Time-variant :

The changes to the data in the data warehouse are tracked and recorded so that reports can be produced showing changes over time.

Non-volatile :

Data in the data warehouse is never over-written or deleted - once committed, the data is static, read-only, and retained for future reporting.

Integrated :

The data warehouse contains data from most or all of an organization's operational systems and this data is made consistent.

The top-down design methodology generates highly consistent dimensional views of data across data marts since all data marts are loaded from the centralized repository. Top-down design has also proven to be robust against business changes. Generating new dimensional data marts against the data stored in the data warehouse is a relatively simple task. The main disadvantage to the top-down methodology is that it represents a very large project with a very broad scope. The up-front cost for implementing a data warehouse using the top-down methodology is significant, and the duration of time from the start of project to the point that end users experience initial benefits can be substantial. In addition, the top-down methodology can be inflexible and unresponsive to changing departmental needs during the implementation phases.

(b) Data Models :

Hierarchical Model: The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1 : N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths. For example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent segment.

Network Model :

The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to many relationships in data. The basic data modeling construct in the network model is the set construct. A set consists of an owner record type, a set name, and a member record type. A member

record type can have that role in more than one set, hence the multiparent concept is supported. An owner record type can also be a member or owner in another set. The data model is a simple network, and link and intersection record types (called junction records by IDMS) may exist, as well as sets between them. Thus, the complete network of relationships is represented by several pair wise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, a set defines a 1: M relationship, although 1: 1 is permitted.

Relational Model :

(RDBMS - relational database management system) A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables :

- Values Are Atomic.
- Each Row is Unique.
- Column Values Are of the Same Kind.
- The Sequence of Columns is Insignificant
- The Sequence of Rows is Insignificant.
- Each column has a unique name.

Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retrieval time, relational databases are classed as dynamic database management system. The RELATIONAL database model is based on the Relational Algebra.

Object/Relational Model :

Object/relational database management systems (ORDBMSs) add new object storage capabilities to the relational systems at the core of modern information systems. These new facilities integrate management of traditional fielded data, complex objects such as time-series and geospatial data and diverse binary media such as audio, video, images, and applets. By encapsulating methods with data structures, an ORDBMS server can execute complex analytical and data manipulation operations to search and transform multimedia and other complex objects.

As an evolutionary technology, the object/relational (OR) approach has inherited the robust transaction

and performance-management features of its relational ancestor and the flexibility of its object-oriented cousin. Database designers can work with familiar tabular structures and data definition languages (DDLs) while assimilating new object-management possibilities. Query and procedural languages and call interfaces in ORDBMSs are familiar: SQL3, vendor procedural languages, and ODBC, JDBC, and proprietary call interfaces are all extensions of RDBMS languages and interfaces. And the leading vendors are, of course, quite well known: IBM, Informix, and Oracle.

(c) Distributed database :

A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Collections of data (e.g. in a database) can be distributed across multiple physical locations. A distributed database is distributed into separate partitions/fragments. Each partition/fragment of a distributed database may be replicated (i.e., redundant fail-overs, RAID-like). Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does depend on the needs of the business and the sensitivity/confidentiality of the data to be stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

Advantages of distributed databases :

- * Reflects organizational structure—database fragments are located in the departments they relate to.
- * Local autonomy—a department can control the data about them (as they are the ones familiar with it.)
- * Improved availability—a fault in one database system will only affect one fragment, instead of the entire database.
- * Improved performance—data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the database to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- * Economics—it costs less to create a network of smaller computers with the power of a single large computer.
- * Modularity—systems can be modified, added and removed from the distributed database without affecting other modules (systems).

Disadvantages of distributed databases :

- * Complexity—extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database - for example, joins become prohibitively expensive when performed across multiple systems.
- * Economics—increased complexity and a more extensive infrastructure means extra labour costs.
- * Security—remote database fragments must be secured, and they are not centralized so the remote

sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).

- * Difficult to maintain integrity—in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- * Inexperience—distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.
- * Lack of standards—there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
- * Database design more complex—besides of the normal difficulties, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.