Q.2 a. List main responsibilities of Database Administrator?

Answer:

Database Administrator (DBA):

The main reason for using DBMS is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator. DBA coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.

Responsibilities of Database administrator:

1. Schema definition: DBA creates the original database schema by executing a set of data definition statements in the Data definition language(DDL).
2. Storage structure and access method definition
3. Schema and physical organization modification: DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization or to alter the physical organization to improve performance.
4. Granting user authority to access the database: By granting different types of authorization, the DBA can regulate which part of the database various user can access.
5. Specifying integrity constraints
6. Acting as liaison with users
7. Monitoring performance and responding to changes in requirements.
8. Understanding and employing the optimal flexible architecture to ease administration, allow flexibility in managing I/O, and to increase the capability to scale the system.

b. What are the advantages of DBMS? Explain the difference between physical and logical data independence.

Answer:

Data independence is the capacity to change the schema at one level of a database system without having to change the schema at the next level. The three-schema architecture allows the feature of data independence. Data independence occurs because when the

schema is changed at some level, the schema at the next level remains unchanged; only the *mapping* between the two levels is changed.

Types of data independence:

Physical Data Independence – It is capacity to change the internal schema without having to change conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files had to be reorganized to improve the performance of retrieval or update. If the same data as before remains in the database, the conceptual schema needs not be changed.

Logical Data Independence - It is the capacity to change the conceptual schema without having to change external schemas or application programs. The conceptual schema may be changed to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

Q.3 a. Define and explain "Mapping Cardinalities". List various types of Mapping Cardinalities.

Answer:

1. Express the number of entities to which another entity can be associated via a relationship set.
2. Most useful in describing binary relationship sets.

For a binary relationship set the mapping cardinality must be one of the following types:

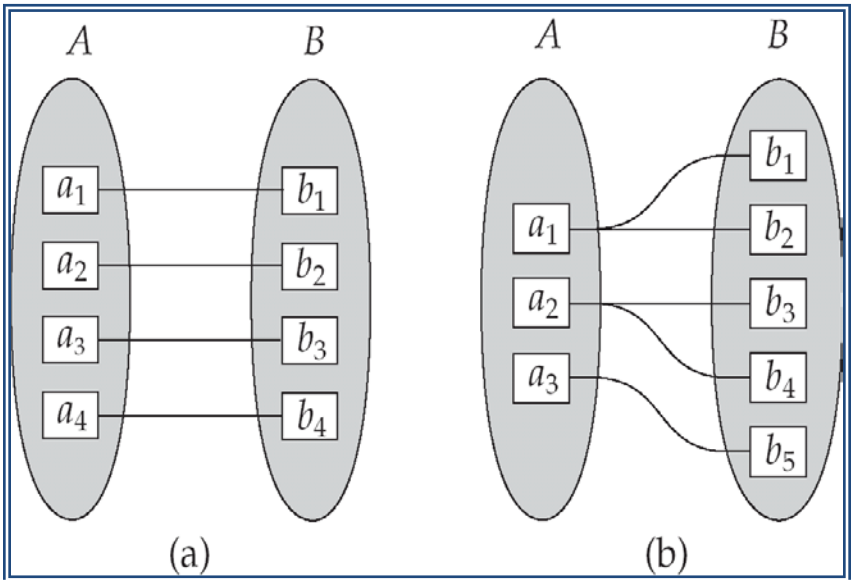
1. One to one
2. One to many
3. Many to one
4. Many to many

One to one: An entity in A is associated with at most one entity in B and an entity in B is associated with at most one entity in A.

On-to-many: An entity in A is associated with any number (zero or more) of entities in B .An entity in B ,however, can be associated with at most one entity in A.

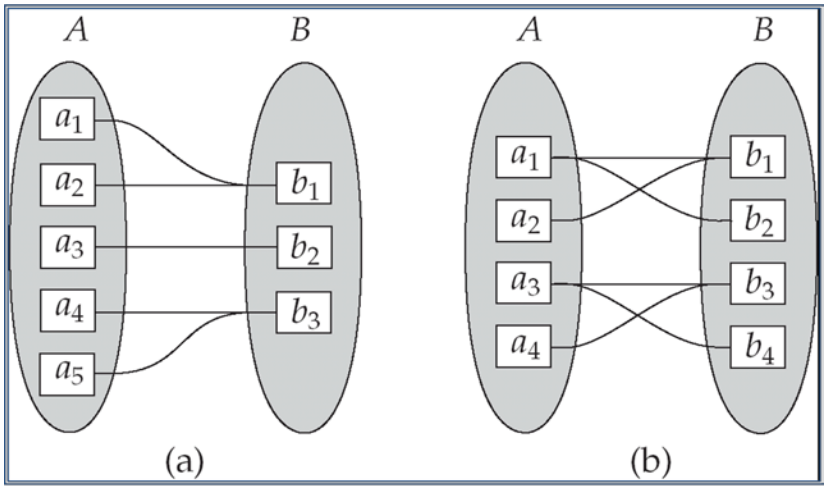
Many-to-one: An entity in A is associated with at most one entity in B and an entity in B, however, can be associated with any number (zero or more) of entities in A.

Many-to-many: An entity in A is associated with any number (zero or more) of entities in B and an entity in B, however, can be associated with any number (zero or more) of entities in A.



One to one One to many

(Some elements in *A* and *B* may not be mapped to any elements in the other set)



Many to one Many to many

(Some elements in A and B may not be mapped to any elements in the other set)

b. Define and explain the data constraints. What are various types of data constraints?

Answer:

3 b. **Data Constraints**: Rules which are enforced on data being entered and prevents the user from entering invalid data into tables are called constraints. Thus, constraints super control the data being entered in tables for permanent storage. Data constraints are attached to the table column through SQL syntax that checks the data for integrity. Once that data constraint is the part of a table column construction, the oracle engine checks the data being entered into a table column against the data constraints. If the data passes this check, it is stored in the table, else the data is rejected. Even if a single column of the record being entered into the table fails a constraint, the entire record is rejected and not stored in the table.

2M

Types of Data Constraints:

6M

Sno	Constraint	Description
1.	PRIMARY KEY	Determines which column(s) uniquely identifies each record. The primary key cannot be NULL, and the data value(s) must be unique. Ensures that the data entered in the table column is Unique across the entire column and none of the cells belonging to the table column are left empty.
2.	FOREIGN KEY	This constraint establishes a relationship between record across a Master and a Detail table and ensures: • That records cannot be inserted into a detail table if corresponding records in the master table do not exist. • That records of the master table cannot be deleted if corresponding records in the detail table exist.
3.	UNIQUE	Ensures that all data values stored in a specified column are unique. The UNIQUE constraint differs from the PRIMARY KEY constraint in that it allows NULL values.
4.	CHECK	Ensures that a specified condition is true before the data value is added to a table. For example, an order's ship date cannot be earlier than its order date.
5.	NOT NULL	Ensures that a specified column cannon contain a NULL value. The NOT NULL constraint can only be created with the column-level approach to table creation.

Q.4 a. Differentiate between Relational Algebra and Relational Calculus.

Answer:

Relational Algebra (RA) and Relational Calculus (RC) are formal languages for the database relational model while SQL is the practical language in the database relational model. In these formal languages a conceptual database model is expressed in mathematical terms and notations while in the practical language – SQL, the mathematical expressions of the functionality and transaction of the database operations are implemented physically. Formal languages provide a medium through which to optimize and implement queries in database transactions.

Of course the first notable differences between these languages in the syntax and notation used in the expressions. Each language, that RA, RC, and SQL have their own notations to express their notations.

Relation algebra is a procedural language where relation calculus is non-procedural language Relational algebra, an offshoot of first-order logic (and of algebra of sets), deals with a set of finitary relations which is closed under certain operators. These operators operate on one or more relations to yield a relation. Relational algebra is a part of computer science.

Relational calculus consists of two calculi, the tuple relational calculus and the domain relational calculus, that are part of the relational model for databases and provide a declarative way to specify database queries. This in contrast to the relational algebra which is also part of the relational model but provides a more procedural way for specifying queries.

Relational Algebra describes step-by-step procedure for computing the desired answer depend on the order in which operator are applies in query.
Relational Calculus describes the set of answer without being explicit about how they should be computed.

b. Explain basic operators of relational algebra.

Answer:

Five basic operators of relational algebra are:

1. Union (□) - Selects tuples that are in either P or Q or in both of them. *The duplicate tuples are eliminated.*

R = P □□Q

2. Minus (–) - Removes common tuples from the first relation.

R = P – Q

3. Cartesian Product or Cross Product (□) - The cartesian product of two relations is the concatenation of tuples belonging to the two relations and consisting of all possible combination of the tuples**.**

R = P □□Q

For Example:

P:

ID	Name
101	Jones
103	Smith
104	Lalonde

Q:

ID	Name
100	John
104	Lalonde

R = P □□Q

ID	Name
100	John
101	Jones
103	Smith
104	Lalonde

R = P – Q

ID	Name
101	Jones
103	Smith

R = P □□Q

P.ID	P.Name	Q.ID	Q.Name
101	Jones	100	John
101	Jones	104	Lalonde
103	Smith	100	John
103	Smith	104	Lalonde
104	Lalonde	100	John
104	Lalonde	104	Lalonde

4. Projection (□) - The projection of a relation is defined as a projection of all its tuples over some set of attributes, i.e., it yields a *vertical subset* of the relation. It is used to either *reduce* the number of attributes (degree) in the resultant relation or to *reorder* attributes. The projection of a relation T on the attribute A is denoted by □A(T).

Projection of relation EMPLOYEE over attribute Name

EMPLOYEE:

ID	Name
101	Jones
103	Smith
104	Lalonde
106	Byron

Name
Jones
Smith
Lalonde
Byron

5. Selection (□) - Selects only some of the tuples, those satisfy given criteria, from the relation. It yields a *horizontal subset* of a given relation, i.e., the action is defined over a complete set of attribute names but only a subset of the tuples are included in the result.

$R = \square B(P)$

For Example:

Result of Selection over **EMPLOYEE** for **ID > 103**

EMPLOYEE:

ID	Name
101	Jones
103	Smith
104	Lalonde
106	Byron

RESULT

ID	Name
104	Lalonde
106	Byron

Q.5 a. Discuss the following SQL commands with example:-
(i) Create Table (ii) Describe
(iii) Delete (iv) Select

Answer:
i) Create Table: This command is used to create a table or a relation in a database.
Syntax:
An SQL relation is defined using the **create table** command:

create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$))

where
r is the name of the relation
each A_i is an attribute name in the schema of relation r
D_i is the data type of values in the domain of attribute A_i

Example:

```
create table supplier( sno char(2),
                       sname varchar2(10),
                       status number(3),
                       city varchar2(10),
                       primary key(sno));
```

Describe: This command displays the column names, the data types and the special attributes connected to the table.

Syntax: **describe <tablename>**

Example: describe supplier;
Output:

Name	type	Null?
sno	NOT NULL	char(3)

sname	varchar2(10)
status	number(3)
city	varchar2(10)

The describe command displays the name of the columns,their data type and size along with the NOT NULL constraint.

i) Delete: The delete command in SQL is used to remove rows /records from the table like:

Removal of all rows:
Delete tablename;
e.g. delete s;
removal of specified rows:
delete from tablename where <search condition>
e.g. delete from s where sno='s1' will delete the row /record of supplier s1 from s table.

ii) Select: This command in SQL is used to view the data in the tables.

Syntax:

select A_1, A_2, ..., A_n
from r_1, r_2, ..., r_m
where P

A_i represents an attribute
R_i represents a relation
P is a predicate.

e.g. select sno,sname,city from s; will show the records of all suppliers existing in the s table
and select sno,sname from s where city='delhi' will show the records of all the suppliers who belong to delhi.

b. Consider the following Supplier-Part-Shipment(S-P-SP) database (keys are underlined)

S (sno,sname,status,city)
P(pno,pname,colour,weight)
SP(sno,pno,qty)

Write the SQL queries for the following statements:-

(i) Get the supplier names of the suppliers who are supplying red colour part.

(ii) Get the total quantity supplied by all suppliers.

Answer:

```
select sname
from s
 where sno in( select sno
                  from sp
                  where pno in ( select pno
                                 from  p
                                 where color =’red’)));
ii) select sum(qty)
    from sp
     group by (sno);
```

Q.6 a. Describe the guidelines for relation schemas.

Answer: Normalization is the process of designing a data model to efficiently store data in a database. The end result is that superfluous data is eliminating, and only data related to the quality is stored within the table.

For example, if we store City, State and ZipCode data for Customers in the same table as Other Customer data. With this approach, we keep repeating the City, State and ZipCode data for all Customers in the same area. Instead of storing the same data again and again, we could normalize the data and create a related table called City. The "City" table could then store City, State and ZipCode along with IDs that relate back to the Customer table, and we can eliminate those three columns from the Customer table and add the new ID column.

In the design of a relational database management system (RDBMS), the process of organize data to minimize being without a job is called normalization. The goal of database normalization is to decay relations with anomaly in order to produce smaller, logical relations. Normalization usually involves dividing large, badly-formed tables into smaller, well-formed tables and important relationships between them. The objective is to cut off data so that additions, deletions, and modification of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

Goals of Normalization

1. Let *R* be a relation scheme with a set *F* of functional dependencies.
2. Decide whether a relation scheme *R* is in "good" form.
3. In the case that a relation scheme *R* is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
   - each relation scheme is in good form
   - the decomposition is a lossless-join decomposition
   - Preferably, the decomposition should be dependency preserving.

Comparison of Normal forms:

Sno	Normal forms	Test	Remedy(Normalisation)
1.	First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
2.	Second (2 NF)	For relations where primary key contains multiple attributes,no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s).Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
3.	Third (3 NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

b. Differentiate 2nd and 3rd normal forms with example.

Answer:

Closure of a Set of Functional Dependencies

1. Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F.
2. For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
3. The set of all functional dependencies logically implied by F is the closure of F.
4. We denote the *closure* of F by F^+.
5. F^+ is a superset of F.

Computation of F^+ :

1. Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F.
   a. For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow$ C
2. The set of all functional dependencies logically implied by F is the *closure* of F.
3. We denote the *closure* of F by F^+.
4. We can find all of F^+ by applying Armstrong's Axioms:
   a. if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
   b. if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(augmentation)**
   c. if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**

These rules are

   d. sound (generate only functional dependencies that actually hold) and
   e. complete (generate all functional dependencies that hold).

Example

- $R = (A, B, C, G, H, I$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$

- some members of F^+
  - $A \rightarrow H$

- by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$

- $AG \rightarrow I$
  - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$

- $CG \rightarrow HI$
  - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$,

- and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$,
  - and then transitivity

Q.7 a. Define and explain Multivalve Dependencies.

Answer:

Multivalued Dependencies (MVDs)

Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The *multivalued dependency*

$$\alpha \rightarrow\rightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
$t_3[\beta] = t_1[\beta]$
$t_3[R - \beta] = t_2[R - \beta]$
$t_4[\beta] = t_2[\beta]$
$t_4[R - \beta] = t_1[R - \beta]$

Example:

Let R be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

$$Y, Z, W$$

then $Y \rightarrow\rightarrow Z$ (Y multidetermines Z) if and only if for all possible relations r (R)

$$< y_1, z_1, w_1 > \in r \text{ and } < y_1, z_2, w_2 > \in r$$

then

$$< y_1, z_1, w_2 > \in r \text{ and } < y_1, z_2, w_1 > \in r$$

since the behavior of Z and W are identical it follows that

$Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

From the definition of multivalued dependency, we can derive the following rule:

If $\alpha \rightarrow \beta$, *then* $\alpha \rightarrow\rightarrow \beta$

That is, every functional dependency is also a multivalued dependency

Use of Multivalued Dependencies:

1. To test relations to determine whether they are legal under a given set of functpecify constraints on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.

   If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations *r′* that does satisfy the multivalued dependency by adding tuples to *r*.

b. Find the canonical cover for the following relation with the given functional dependencies:

$$R = (A, B, C)$$
$$F = \{A \rightarrow BC$$
$$B \rightarrow C$$
$$A \rightarrow B$$
$$AB \rightarrow C\}$$

Answer: $R = (A, B, C)$

$F = \{A \rightarrow BC$

$B \rightarrow C$

$A \rightarrow B$

$AB \rightarrow C\}$

Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$

Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

A is extraneous in $AB \rightarrow C$

Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies

Yes: in fact, $B \rightarrow C$ is already present!

Set is now $\{A \rightarrow BC, B \rightarrow C\}$

C is extraneous in $A \rightarrow BC$

Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies

Yes: using transitivity on $A \rightarrow B$ and $B \rightarrow C$.

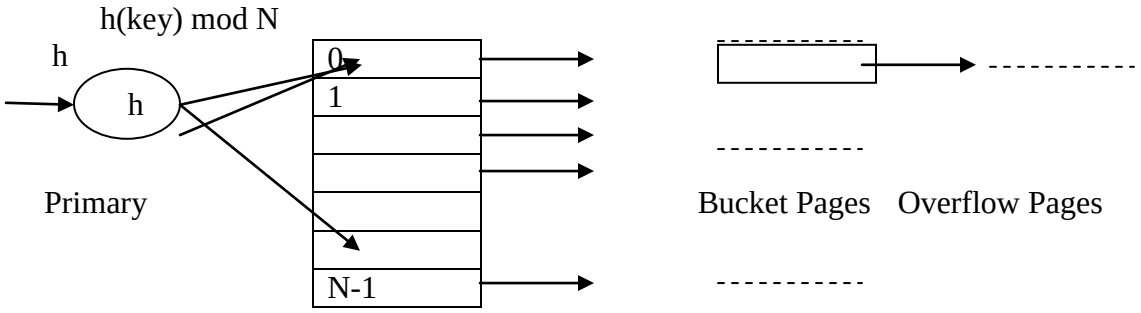
Can use attribute closure of A in more complex cases

The canonical cover is: $A \rightarrow B$

$B \rightarrow C$

Q.8 a. Describe the static hash file with buckets and chaining and show how insertion, deletion and modification of a record can be performed.

Answer:
In static hash file organization, the term bucket is used to denote a unit storage that can store one or more records. A file consists of buckets 0 through N-1, with one primary page per bucket initially and additional overflow pages chained with bucket, if required later. Buckets contain data entries (or data records). In hashing scheme, a hash function, h, is performed on the key of the record to identify the bucket to which data record belongs to. The hash function is an important component of the hashing approach. The main problem with static hash file is that the number of buckets is fixed.



Insertion of a record – To insert a data entry, the hash function is used to identify the correct bucket and then put the data entry there. If there is no space for this data entry, a new overflow page will be allocated, put the data entry on this page, and the page to the overflow chain of the bucket.

Deletion of a record – To delete a data entry, the hash function is used to identify the correct bucket, locate the data entry by searching the bucket, and then remove it. If the data entry is the last in an overflow page, the overflow page is removed from the overflow chain of the bucket and added to a list of free pages.

Modification of a record – To modify a data entry, the hash function is used to identify the correct bucket, locate the data entry by searching the bucket and get it, modify the data entry, and then rewrite the modified data entry on it.

b. Explain the types of multi-level ordered indexes.

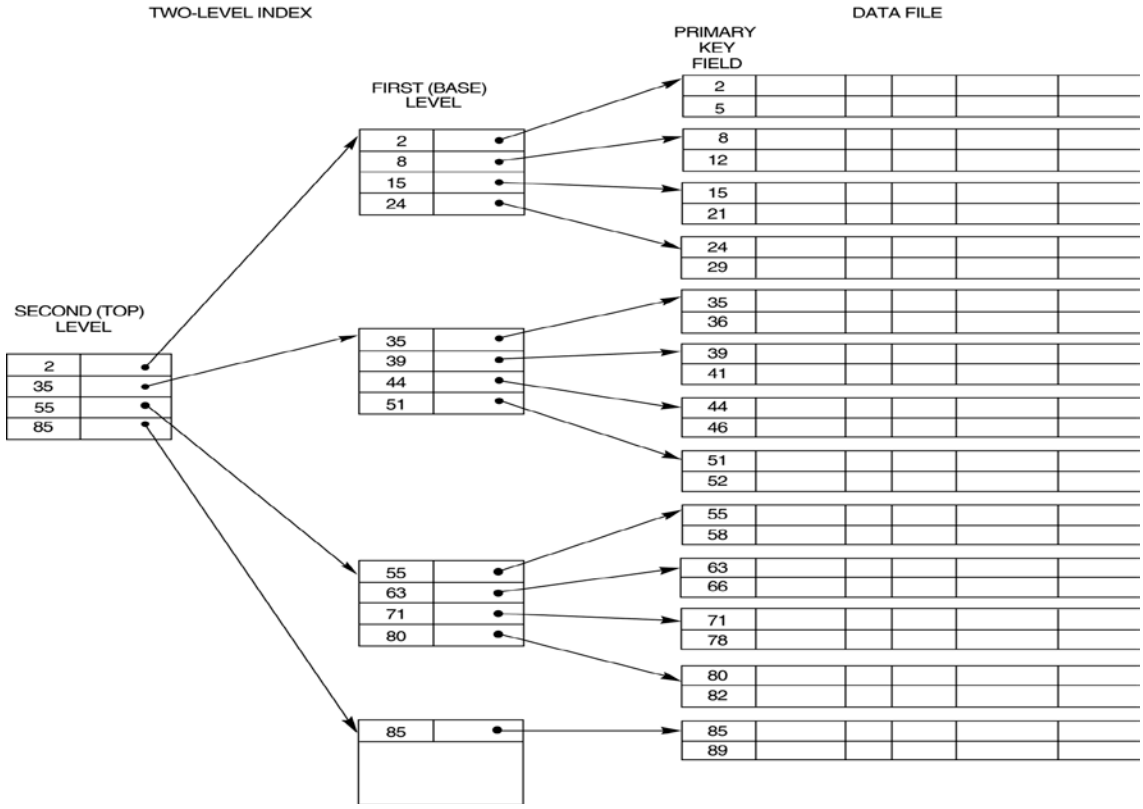
Answer:
Multi-Level Indexes:

- Because a single-level index is an ordered file, we can create a primary index *to the index itself* ; in this case, the original index file is called the *first-level index* and the index to the index is called the *second-level index.*

- We can repeat the process, creating a third, fourth, ..., top level until all entries of the *top level* fit in one disk block.
- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block

Dynamic Multi-Level Indexes

- To retain the benefits of using multilevel indexing while reducing index insertion and deletion problems
- Leaves some space in each of its blocks for inserting new entries and uses appropriate insertion/deletion algorithms for creating and deleting new index blocks when the data file grows and shrinks.
- Often implemented by using data structures called B-trees and B+-trees



(A two-level primary index ISAM (Indexed Sequential Access Method) organization)

Q.9 a. Explain sort-merge algorithm for external sorting.

Answer:

External sorting refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory, such as most database files. The typical external sorting algorithm uses a sort-merge strategy, which starts by sorting small subfiles—called runs—of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn. The sort-merge algorithm, like other database algorithms, requires *buffer space* in main memory, where the actual sorting and merging of the runs is performed. The basic algorithm, given below, consists of two phases: the sorting phase and the merging phase. The buffer space in main memory is part of the DBMS cache—an area in the computer's main memory that is controlled by the DBMS. The buffer space is divided into individual buffers, where each buffer is the same size in bytes as the size of one disk block. Thus, one buffer can hold the contents of exactly *one disk block.* In the sorting phase, runs (portions or pieces) of the file that can fit in the available buffer space are read into main memory, sorted using an *internal* sorting algorithm, and written back to disk as temporary sorted subfiles (or runs). The size of each run and the number of initial runs (*nR*) are dictated by the number of file blocks (*b*) and the available buffer space (*nB*). For example, if the number of available main memory buffers *nB* = 5 disk blocks and the size of the file *b* = 1024 disk blocks, then *nR*= (*b*/*nB*) □or 205 initial runs each of size 5 blocks (except the last run which will have only 4 blocks). Hence, after the sorting phase, 205 sorted runs (or 205 sorted subfiles of the original file) are stored as temporary subfiles on disk.In the merging phase, the sorted runs are merged during one or more merge passes. Each merge pass can have one or more merge steps. The degree of merging (*dM*) is the number of sorted subfiles that can be merged in each merge step.During each merge step, one buffer block is needed to hold one disk block from each of the sorted subfiles being merged, and one additional buffer is needed for containing one disk block of the merge result, which will produce a larger sorted file that is the result of merging several smaller sorted subfiles.Hence, *dM* is the smaller of (*nB* −□1) and *nR*, and the number of merge passes is (log$d_M(n_R)$) □In our example, where *nB* = 5, *dM* = 4 (four-way merging), so the 205 initial sorted runs would be merged 4 at a time in each step into 52 larger sorted subfiles at the end of the first merge pass. These 52 sorted files are then merged 4 at a time into

13 sorted files, which are then merged into 4 sorted files, and then finally into 1 fully sorted file, which means that *four passes* are needed.

Sort-merge algorithm for external sorting

Internal sorting

set *i* = 1;

j = *b*; {size of the file in blocks}

k = *nB*; {size of buffer in blocks}

m = ⌈*j*/*k*⌉;

{**Sorting Phase**}

while (*i* ≤ *m*)

do {

read next *k* blocks of the file into the buffer or if there are less than *k* blocks remaining, then read in the remaining blocks;

sort the records in the buffer and write as a temporary subfile;

i = *i* + 1;

}

{**Merging Phase:** merge subfiles until only 1 remains}

set *i* = 1;

p = ⌈log*k*–1*m*⌉; {*p* is the number of passes for the merging phase}

j = *m*;

while (*i* ≤*p*)

do {

n = 1;

q = ⌈*j*/(*k*–1)⌉; {number of subfiles to write in this pass}

while (*n* ≤ *q*)

do

{

read next *k*–1 subfiles or remaining subfiles (from previous pass) one block at a time;

merge and write as new subfile one block at a time;

n = *n* + 1;

}

j = *q*;

i = *i* + 1;

}

b. Explain various cost components that are the constituents of a Query Execution.

Answer:

Cost Components for Query Execution:

The cost of executing a query includes the following components:

1. **Access cost to secondary storage.** This is the cost of transferring (reading and writing) data blocks between secondary disk storage and main memory buffers. This is also known as *disk I/O (input/output) cost*. The cost of searching for records in a disk file depends on the type of access structures on that file, such as ordering, hashing, and primary or secondary indexes. In addition, factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.

2. **Disk storage cost.** This is the cost of storing on disk any intermediate files that are generated by an execution strategy for the query.

3. **Computation cost.** This is the cost of performing in-memory operations on the records within the data buffers during query execution. Such operations include searching for and sorting records, merging records for a join or a sort operation, and performing computations on field values. This is also known as *CPU (central processing unit) cost*.

4. **Memory usage cost.** This is the cost pertaining to the number of main memory buffers needed during query execution.

5. **Communication cost.** This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated. In distributed databases it would also include the cost of transferring tables and results among various computers during query evaluation.

Text book

Fundamentals of Database Systems, Elmasri, Navathe, Somayajulu, Gupta, Pearson Education, 2006