Tulsiramji Gaikwad-Patil College of Engineering and Technology

Wardha Road, Nagpur-441 108

NAAC Accredited

Department of Information Technology

Session 2018-19 (Even Semester)

Sixth Semester **Subject: Database Management System**

Notes on Unit-I: Introduction to Database Systems

Q.1. Explain the concept and meaning of Database Management System? 6M

Ans.

- A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data.
- The collection of data, usually referred to as the **database**, contains information relevant to an enterprise.
- The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.
- Database systems are designed to manage large bodies of information.
- Management of data involves both defining structures for storage of information and pro-viding mechanisms for the manipulation of information.
- In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.
- If data are to be shared among several users, the system must avoid possible anomalous results.
- Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data.
- Databases are widely used. Here are some representative applications:

1. Banking: For customer information, accounts, and loans, and banking transactions.
2. Airlines: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner — terminals situated around the world accessed the central database system through phone lines and other data networks.
3. Universities: For student information, course registrations, and grades.
4. Credit card transactions: For purchases on credit cards and generation of monthly statements.
5. Telecommunication: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
6. Finance: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
7. Sales: For customer, product, and purchase information.
8. Manufacturing: For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
9. Human resources: For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.
10. As the list illustrates, databases form an essential part of almost all enterprises today.

Although user interfaces hide details of access to a database, and most people are not

Q.2. Explain the conventional file processing system with example.
Or
Explain the purpose of database system.
Ans

- Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts.
- One way to keep the information on a computer is to store it in operating system files.
- To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including a program to debit or credit an account, a program to add a new account, a program to find the balance of an account, a program to generate monthly statements, etc.
- System programmers wrote these application programs to meet the needs of the bank.
- New application programs are added to the system as the need arises.
- For example, suppose that the savings bank decides to offer checking accounts. As a result, the bank creates new permanent files that contain information about all the checking accounts maintained in the bank, and it may have to write new application programs to deal with situations that do not arise in savings accounts, such as overdrafts. Thus, as time goes by, the system acquires more files and more application programs.
- This is an example of a typical **file-processing system** is supported by a conventional operating system.
- The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) came along, organizations usu-ally stored information in such systems.
- Keeping organizational information in a file-processing system has a number of major disadvantages due to which the concept of database was implemented. Some of the drawbacks are listed below:

1. Data redundancy and inconsistency.

2. Difficulty in accessing data.

3. Data isolation.

4. Integrity problems.

5. Atomicity problems.

6. Concurrent-access anomalies.

7. Security problems.

To solve the problems of traditional file processing system a database system is used. A database system is a collection of interrelated files and a set of programs that allow users to access and modify these files. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

Q.3. Discuss the drawbacks of a Conventional File Processing System over DBMS.

CT: W-06 7M

Or
Explain the drawbacks of file system over database system.

CT: S-07 (7M), S-12 (6M)

Ans:

- A file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them.
- File systems may use a storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files.
- A typical example of file processing system is a system used to store and manage data of each department or area within an organization having its own set of files, often creating data redundancy and data isolation.
- Before the advent of DBMS the data or records were stored in permanent system files using the conventional operating system.
- Application programs were then created independently to access the data stored in these files.

The following are the drawbacks of traditional File System:

1. **Difficulty in accessing data:**
   It is not easy to retrieve information using a conventional file processing system. Getting the exact result matching the query is difficult.
2. **Duplication of data:**
   - Often the same information is stored in more than one file. This uncontrolled duplication of data is not required for several reasons, such as:
   - Duplication is wasteful. It costs time and money to enter the data more than once
   - It takes up additional storage space, again with associated costs.
   - Duplication can lead to loss of data integrity; in other words the data is no longer consistent.
3. **Separated and Isolated Data:**
   - To make a decision, a user might need data from two separate files.
   - First, the files were evaluated by analysts and programmers to determine the specific data required from each file and then applications were written in a programming language to process and extract the needed data.
   - The amount of work involved increased because data from several files was needed.
   - Since data is scattered in various files, and files may be in different formats, it is difficult to write new application programs to retrieve the appropriate data.
4. **Data Security:**
   - The security of data is low in file based system because, the data maintained in the flat file(s) is easily accessible.

   Example: Consider the Banking System. The Customer Transaction file has details about the total available balance of all customers. A Customer wants information about his account balance. In a file system it is difficult to give the Customer access to only his data in the file. Thus enforcing security constraints for the entire file or for certain data items are difficult.. If the physical formats that were coded into the application program by programmers was changed, the code in each file containing that format must be updated. Furthermore, instructions for data storage and access were written into the application's code. Therefore, .changes in storage structure or access methods could greatly affect the processing or results of an application.

5. Data Dependence:

- In file processing systems, files and records were described by specific physical formats that were coded into the application program by programmers.
- If the format of a certain record was changed, the code in each file containing that format must be updated.
- Furthermore, instructions for data storage and access were written into the application's code.
- Therefore, changes in storage structure or access methods could greatly affect the processing or results of an application.

6. Data Inflexibility:

Program-data interdependency and data isolation, limited the flexibility of file processing systems in providing users with results of information requests.

7. Incompatible file formats:

- As the structure of files is embedded in the application programs, the structures are dependent on the application programming language.

Example, the structure of a file generated by a COBOL program may be different from the structure of a file generated by a 'C' program.

The direct incompatibility of such files makes them difficult to process jointly.

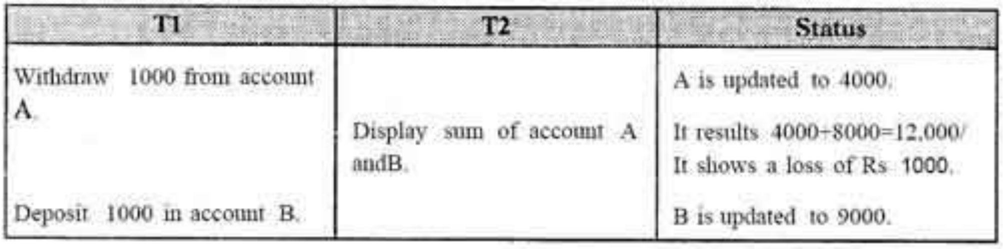
8. Concurrency problems.

When multiple users access the same piece of data at same interval of time then it is called as concurrency of the system.

When two or more users read the data simultaneously there is no problem, but when they like to update a file simultaneously, it may result in a big problem.

Example:

Let us consider a scenario where in transaction T 1 a user transfers an amout1t 1000 from Account A to B (initial value of A is 5000 and B is 8000). In mean while, another transaction T2, tries to display the sum of account A and B is also executed. If both the transaction runs in parallel it may results inconsistency as shown below:



T1	T2	Status
Withdraw 1000 from account A.		A is updated to 4000.
	Display sum of account A andB.	It results 4000+8000=12,000/ It shows a loss of Rs 1000.
Deposit 1000 in account B.		B is updated to 9000.

The above schedule results inconsistency of database and it shows Rs.12,000 as sum of accounts A and B instead of Rs .13,000. The problem occurs because second concurrently running transaction T2, reads A and B at intermediate point and computes its sum, which results inconsistent value.

9. Integrity Problems:

The data values may need to satisfy some integrity constraints.

For example, the balance field Value must be grater than 5000.

We have to handle this through program code in file processing systems.

But in database we can declare the integrity constraints along with definition itself.

10. Atomicity Problem:

It is difficult to ensure atomicity in file processing system.

Example: Transferring $100 from Account A to account B. If a failure occurs during execution there could be situation like $100 is deducted from Account A and not credited in Account B.

Q.4. What are the advantages of using Database Management approach over conventional file system approach?

CS: S-06 7M

Ans:

- A Database Management System (DBMS) is a set of computer programs that controls the creation, maintenance, and the use of the database of an organization and its end users.
- It allows organizations to place control of organization-wide database development in the hands of database administrators (DBAs) and other specialists.
- Following are the advantage of DBMS over conventional file system:

1. Program-Data Independence:

- The separation of data descriptions from the application programs that use the data is called Data independence.
- With the database approach, Data descriptions are stored in a central location called the repository.
- This property of database systems allows an organization's data to change without changing the application programs that process the data.

2. Data Redundancy and consistency:

- In File-processing System, files having different formats and application programs may be created by different programmers.
- Similarly different programs may be written in several programming languages.
- The same information placed at different files causes redundancy and inconsistency causing need of higher storage and access cost.
- Example: The address and telephone number of a person may exist in two files containing saving account records and checking account records. Now a change in person's address may reflect the saving account records but not any where in the whole system. This results in data inconsistency.
- One solution to avoid this data redundancy is keeping the multiple copies of same information, replace it by a system where the address and telephone number stored at just one place physically while it is accessible to all applications from this itself.

3. Data security:

- Since the data is stored centrally, enforcing security constraints is much easier.
- The DBMS ensures that the only means of access to the database is through an authorized channel.
- Hence, data security checks can be carried out whenever access is attempted to sensitive data.
- To ensure security, a DBMS provides security tools such as user codes and passwords.
- Different checks can be established for each type of access (addition, modification, deletion, etc.) to each piece of information in the database.

4. Data Integrity: -

- The data values stored in the database must satisfy certain types of consistency constraints.
- For example, the balance of a bank account may never fall below a prescribed amount.
- These constraints are enforced in the system by adding appropriate code in the various application programs.
- However, when new constraints are added, it is difficult to change the programs to enforce them. ---The problem arises when constraints involve several data items from different files.
- In database approach, enforcing data integrity is much easier.
- Various integrity constraints are identified by database designer during database design.
- Some of these data integrity constraints can be enforced automatically by the DBMS, and others may have to be checked by the application programs.

5. Improved Data Sharing: -

- A database is designed as a shared resource.
- Authorized internal and external users are granted permission to use the database, and each user is provided one or more user views to facilitate this use.
- A user view is a logical description of some portion of the database that is required by a user to perform some task.

6. Increased Productive of Application Development: -

- A major advantage of the database approach is that it greatly reduces the cost and time of developing new business applications.
- There are two important reasons that data base applications can often be developed much more rapidly than conventional file applications.

a) Assuming that the database and the related data maintenance applications have already been designed and implemented, the programmer can concentrate on the specific functions required for the new application, without having to worry about file design or low-level implementation details.

b) The data base management system provides a number of high-level productivity tools such as forms and reports generators and high-level languages that automate some of the activities of database design and implementation.

7. Backup and recovery:

- The DBMS provides backup and recovery subsystem that is responsible for recovery from hardware and software failures.
- For example, if the failure occurs in between the transaction, the DBMS recovery subsystem either rolls back the database to the state which existed prior to the start of the transaction.
- It resumes the transaction from the point it was interrupted so that its complete effect can be recorded in the database.

Q5. Differentiate between conventional file system and DBMS.

CS: W-12, CT: W-07, W-08, 6M

Or

Describe various pros and cons of DBMS over conventional file system.

CS: S-12 6M

Ans.

Sr. No	DBMS	File Processing System
01	It is a computerized record-keeping system	It is a collection of individual files accessed by applications programs.
02	A database management system coordinates both the physical and logical access to the data.	A file-processing system co-ordinates only the physical access
03	A database management system reduces	Data written by one program in a file processing

	the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it.	system may not be readable y another program.
04	A database management system is designed to allow flexible access to data (i.e, queries)	A file-processing system is designed to allow predetermined access to data (i.e. compiled programs)
05	A DBMS is designed to co-ordinate multiple users accessing the same data at the same time.	A file –processing system is usually designed to allow one or more programs to access different data files at the same time.
06	Unauthorized access is restricted in DBMS	There are no restrictions in File system
07	DBMS provides multiple user interfaces	Data is isolated in File system
08	It is a bit complicated	The file system is easy to use
09	It is expensive as compared to file system	The file system is cheaper
10	Data is stored at a centralized location	Data is separated (isolated)
11	It is suitable for large databases	Suitable for small databases
12	It is not embedded in OS, sold separately	It is embedded in OS
13	It has good processing power	It has low processing speed
14	The transactions are possible	The concept of transactions is not possible
15	Redundancy is controlled in DBMS	It is not controlled in this system

Q.6. Explain different levels of Abstraction of DBMS System?
Or

CS: W-12 6M

Explain different levels of data abstraction with neat diagram.
Or

CS: W-13 6M

Illustrate the difference between the three levels of data abstraction.
Or

CS: W-10 7M

What do you mean by level of abstraction? Explain detailed three level architecture of database management system.
Or

CT:S-08,S-10 7M

Explain three level architecture with neat diagram.
Ans:

- For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database.
- Since many database-systems users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system. The Figure below shows the various levels of abstraction.

1. Physical Level :

- The lowest level of abstraction describes how the data are actually stored.
- The physical level describes complex low-level data structures in detail.
- The **physical schema** describes details of how data is stored: files, indices, etc. on the random access disk system.
- It also typically describes the record layout of files and type of files (hash, b-tree, flat).
- Early applications worked at this level - explicitly dealt with details. E.g., minimizing physical distances between related data and organizing the data structures within the file (blocked records, linked lists of blocks, etc.)

Problems:

- Routines are hardcoded to deal with physical representation.
- Changes to data structures are difficult to make.
- Application code becomes complex since it must deal with details.
- Rapid implementation of new features very difficult.

2. Logical Level :

- The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data.
- The logical level thus describes the entire database in terms of a small number of relatively simple structures.
- Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity.

- Database administrators, who must decide what information to keep in the database, use the logical level of abstraction. Q2 3
- This level hides details of the physical level and also called as conceptual level.
- In the relational model, the conceptual schema presents data as a set of tables.
- The DBMS maps data access between the conceptual to physical schemas automatically.
- Physical schema can be changed without changing application
- DBMS must change mapping from conceptual to physical.
- Referred to as **physical data independence**.

3. View Level :

- The highest level of abstraction describes only part of the entire database.
- Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.
- Many users of the database system do not need all this information; instead, they need to access only a part of the database.
- The view level of abstraction exists to simplify their interaction with the system.
- The system may provide many views for the same database.
- In the relational model, the **external schema** also presents data as a set of relations.
- An external schema specifies a **view** of the data in terms of the conceptual level.
- It is designed to fulfil the needs of a particular category of users.
- Portions of stored data should not be seen by some users and begins to implement a level of security and simplifies the view for these users

Examples:

- Students should not see faculty salaries.
- Faculty should not see billing or payment data.

-Applications are written in terms of an external schema.
-The external view is computed when accessed. It is not stored.
-Different external schemas can be provided to different categories of users.
-Translation from external level to conceptual level is done automatically by DBMS at run time.
-The conceptual schema can be changed without changing application:

- Mapping from external to conceptual must be changed.
- Referred to as **conceptual data independence**.

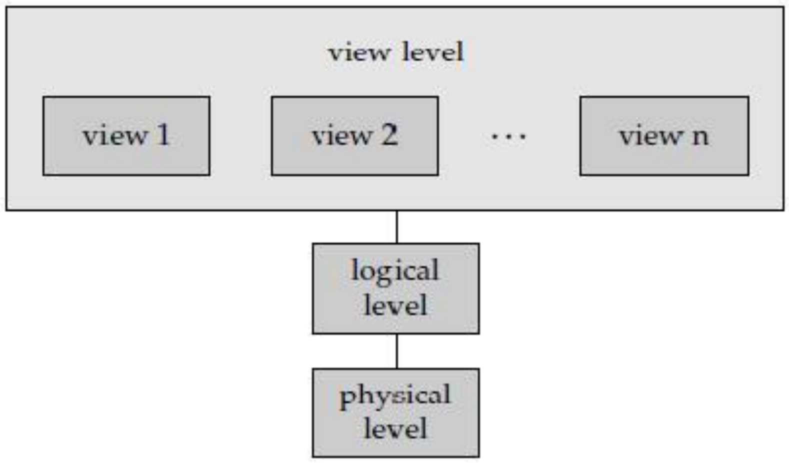


Fig. 5.2. DBMS Levels of Abstraction (Schemas)

Q.7. What are different types of database languages? Give suitable example of each.

CT: S-10, CS: W-11,S-13 6M

Ans.
Database Languages:

- A database system provides a **data definition language** to specify the database schema and a **data manipulation language** to express database queries and updates.
- In practice, the data definition and data manipulation languages are not two separate languages. Instead they simply form parts of a single database language, such as the widely used SQL language.

(1). Data-Definition Language:

- We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**).
- For instance, the following statement in the SQL language defines the *account* table:

> **create table** *account*
> (*account-number* **char**(10),
> *Balance* **integer**);

- Execution of the above DDL statement creates the *account* table. In addition, it updates a special set of tables called the **data dictionary** or **data directory**.
- A data dictionary contains **metadata**—that is, data about data. The schema of a table is an example of metadata. A database system consults the data dictionary before reading or modifying actual data.
- We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language.
- These statements define the implementation details of the database schemas, which are usually hidden from the users. The data values stored in the database must satisfy certain **consistency constraints**.

Example: Suppose the balance on an account should not fall below $100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated.

(2). Data-Manipulation Language:

- A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model.

Using data manipulation language the data can be accessed in following ways:
(i). The retrieval of information stored in the database
(ii). The insertion of new information into the database
(iii). The deletion of information from the database
(iv). The modification of information stored in the database

- There are basically two types:

(a). **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
(b). **Declarative DMLs** (also referred to as **nonprocedural** DMLs) require a user to specify *what* data are needed *without* specifying how to get those data.

  - Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural.

- A **query** is a statement requesting the retrieval of information.
- The portion of a DML that involves information retrieval is called a **query language**.
- Although technically incorrect, it is common practice to use the terms *query language* and *data manipulation language* synonymously.

Q. 8. What is the difference between procedural and non-procedural DML's.

CT: W-10, CS: W-13,S-13 4M

Ans.

Procedural DML	**Non- procedural DML**
It requires a user to specify what data are needed and how to get those data.	It requires a user to specify what data rae needed without specifying how to get those data.
It provides user a means to define precisely each step for solving problem.	It does not generate code as efficient as produced by procedural language.
It is difficult for normal users to learn and implement procedural DML's Example: Relational Algebra	It is easier to learn and use than procedural DML's Example: Tuple Relational Calculus

Q.9. Differentiate between DDL and DML. **6M**

Ans:

DDL	DML
Data Definition Language (DDL) statements are used to define the database structure or schema. Some examples:	**Data Manipulation Language (DML)** statements are used for managing data within schema objects. Some examples:
○ CREATE - to create objects in the database ○ ALTER - alters the structure of the database ○ DROP - delete objects from the database ○ TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed ○ COMMENT - add comments to the data dictionary ○ RENAME - rename an object	○ SELECT - retrieve data from the a database ○ INSERT - insert data into a table ○ UPDATE - updates existing data within a table ○ DELETE - deletes all records from a table, the space for the records remain
Data Definition Language (also known as DDL) is a computer language used to define data structures.	They are used by computer programs, and/or database users, to manipulate data in a database – that is, insert, delete and update this data in the database.
The DDL has no types .	It has two types: Procedural and nonprocedural

Q.10. What is data independence. Write short note on physical and logical data independence.

CS: S-11, CT: S-06 6M

Ans:

- Data independence is the type of data transparency that is important for a centralized DBMS.
- It refers to the power of user applications to make changes in the definition and organization of data.
- The ability to modify schema definition in a level without affecting schema definition in the next higher level is called data independence.
- There are two levels of data independence, they are Physical data independence and Logical data independence.
- Physical data independence deals with hiding the details of the storage structure from user applications.
- Each higher level of the data architecture is immune to changes of the next lower level of the architecture.
- The logical scheme stays unchanged even though the storage space or type of some data is changed for reasons of optimization or reorganization. In this external schema does not change.
- In this internal schema changes may be required due to some physical schema were reorganized here. The data independence and operation independence together gives the feature of data abstraction. 10. The two levels of data independence are explained below:

a). Logical Data Independance:

- the First level, the logical structure of the data is known as the 'schema definition'.
- In general, if a user application operates on a subset of the attributes of a relation, it should not be affected later when new attributes are added to the same relation.
- Logical data independence is the ability to modify the logical schema without causing application program to be rewritten.
- Modifications at the logical level are necessary whenever the logical structure of the database is altered (for example, when money-market accounts are added to banking system).
- This ability to change the logical (conceptual) schema without changing the External schema (User View) is useful in DBMS.
- For example, the addition or removal of new entities, attributes, or relationships to the conceptual schema should be possible without having to change existing external schemas or having to rewrite existing application programs.
- Logical data independence is more difficult to achieve than is physical data independence, since application programs are heavily dependent on the logical structure of the data that they access.

b). Physical Data Independence:

- At the Second level, the physical structure of the data is referred to as "physical data description".
- Physical data independence deals with hiding the details of the storage structure from user application.
- Physical data independence means we change the physical storage/level without affecting the conceptual or external view of the data. Mapping techniques absorbs the new changes.
- The ability to change the physical schema without changing the logical schema can be called physical data independence.
- Example, a change to the internal schema, such as using different file organization or storage structures, storage devices, or indexing strategy, should be possible without having to change the conceptual or external schemas.
- Physical data independence is present in most databases and file environment in which hardware storage of encoding, exact location of data on disk, merging of records, so on this are hidden from user.
- We can change the structure of a database without affecting the data required by users and program. This feature was not available in file oriented approach.
- Physical data independence is the ability to modify the physical schema without causing application programs to be rewritten. Modifications at the physical level are occasionally necessary to improve performance. The types of data independence is shown in figure below:

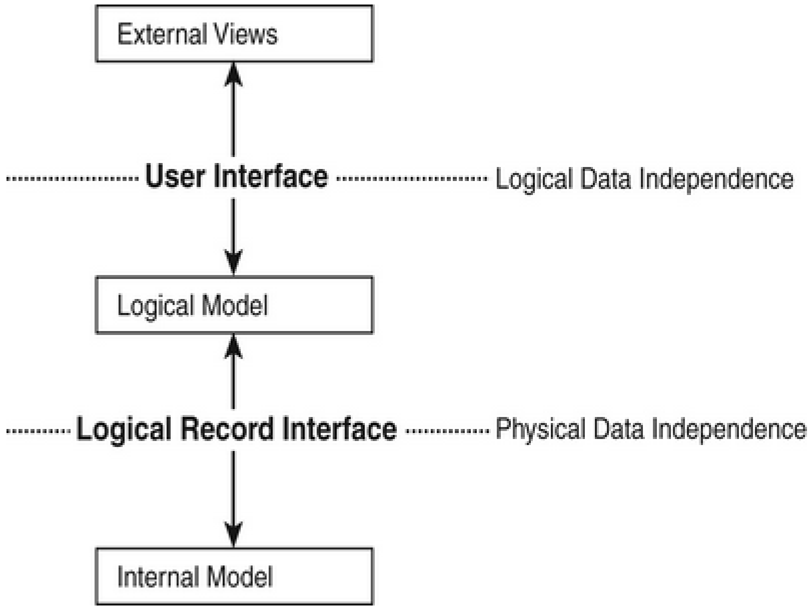
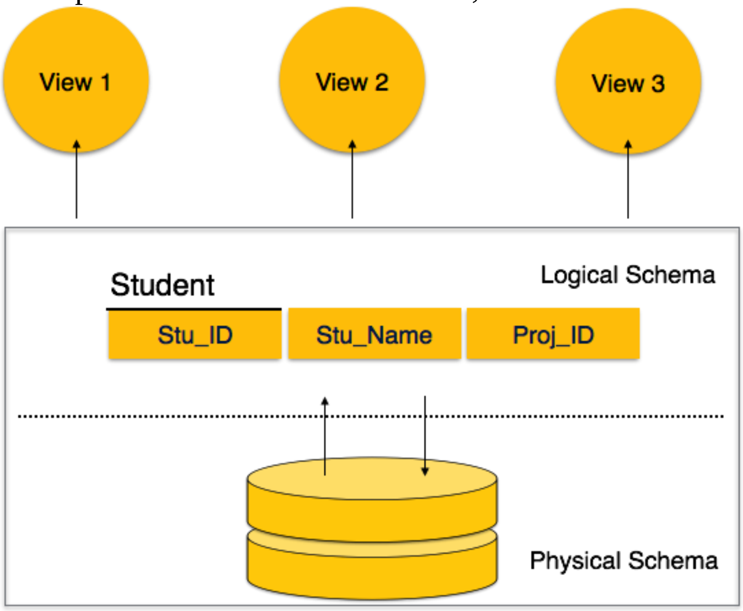


Fig.Types of Data Independance

Q. 11. Describe schema and instances.
Ans.

- Databases change over time as information is inserted and deleted.
- The collection of information stored in the database at a particular moment is called an **instance** of the database.
- The overall design of the database is called the database **schema**.
- Schemas are changed infrequently, if at all.
- The concept of database schemas and instances can be understood by analogy to a program written in a programming language.
- A database schema corresponds to the variable declarations (along with associated type definitions) in a program.
- Each variable has a particular value at a given instant.
- The values of the variables in a program at a point in time correspond to an instance of a database schema.
- Database systems have several schemas, partitioned according to the levels of abstraction.
- The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level.
- A database may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database.
- Of these, the logical schema is the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema.
- The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs.
- Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.
- Database schema structure of DBMS and it represents the logical view of entire database.
- It tells about how the data is organized and how relation among them is associated.
- A database schema defines its entities and the relationship among them. Database schema is a

descriptive detail of the database, which can be depicted by means of schema diagrams.



Database schema can be divided broadly in two categories:

- **Physical Database Schema:** This schema pertains to the actual storage of data and its form of storage like files, indices etc. It defines the how data will be stored in secondary storage etc.
- **Logical Database Schema:** This defines all logical constraints that need to be applied on data stored. It defines tables, views and integrity constraints etc.

Database Instance

It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when database doesn't exist at all and very hard to do any changes once the database is operational. Database schema does not contain any data or information.

Database instances, is a state of operational database with data at any given time. This is a snapshot of database. Database instances tend to change with time. DBMS ensures that its every instance (state) must be a valid state by keeping up to all validation, constraints and condition that database designers has imposed or it is expected from DBMS itself.

Q.12. Differentiate between instance and schema

Ans.

Sr.No.	**Schema**	**Instance**
01	Database schema skeleton structure of and it represents the logical view of entire database.	Database instances, is a state of operational database with data at any given time.
02	It tells about how the data is organized and how relation among them is associated.	This is a snapshot of database. Database instances tend to change with time.
03	Database schema does not contain any data or information.	It contains information which changes when updates are done

04	The schema is called intension	It is called extention
05	A schema diagram, as shown above, displays only names of record types (entities) and names of data items (attributes) and does not show the relationships among the various files.	An instance will be set of values of these attributes at a given time
06	The schema will be same but values in it will change	The instance of a database changes from time to time.
07	**Example:**	**Example:**

Figure shows the database schema for Publisher database.

An example of schema - PUBLISHER schema

P_ID	Pname	Address	State	Phone	Email_id

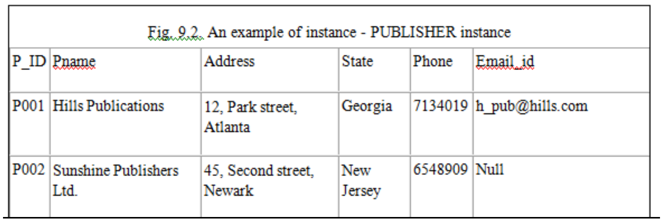


Fig. 9.2. An example of instance - PUBLISHER instance

P_ID	Pname	Address	State	Phone	Email_id
P001	Hills Publications	12, Park street, Atlanta	Georgia	7134019	h_pub@hills.com
P002	Sunshine Publishers Ltd.	45, Second street, Newark	New Jersey	6548909	Null

Q.13. Draw the Architecture of DBMS in detail with the help of neat diagram. Explain the function of each component.
Or

CS: W-12, CT: W-07, S-12,S-09 8M

Draw and explain the architecture of a database management system.
Or
Explain the functional components of Database management system.

CT: S-07, W-10, W-11, S-11 7M

Ans.
Architecture of DBMS:

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of a database system can be broadly divided into the storage manager and the query processor components.
- The storage manager is important because databases typically require a large amount of storage space.
- A typical structure of a DBMS with its components and relationships between them is shown below.
- Each module or component shown is assigned a specific operation to perform.
- Some of the functions of the DBMS are supported by operating systems (OS) to provide basic services and DBMS is built on top of it.
- The physical data and system catalog are stored on a physical disk.
- Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed.
- Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory.
- The query processor is important because it helps the database system simplify and facilitate access to data
- The various components are explained as follows:

1. Storage Manager

A *storage manager* is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

-The storage manager is responsible for the interaction with the file manager.

-The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

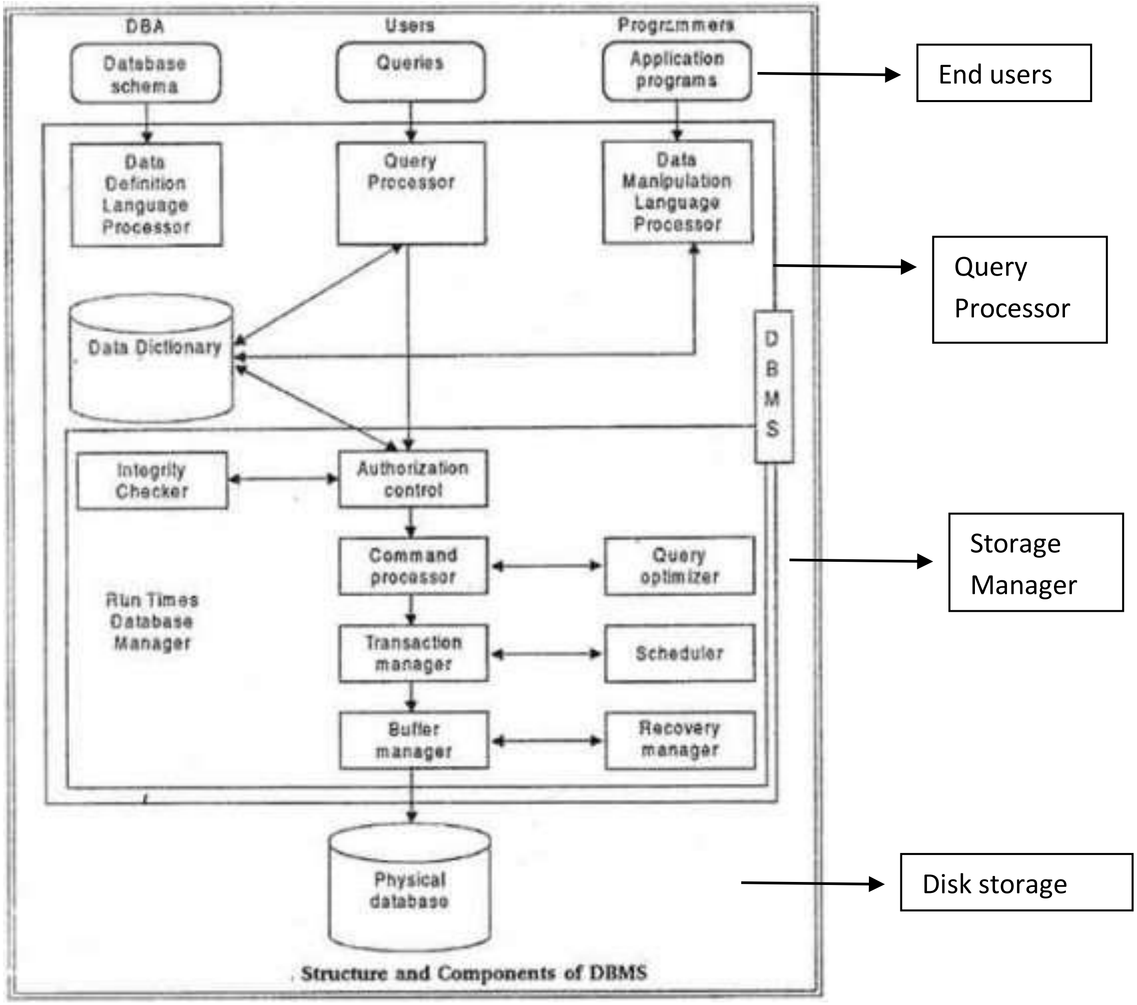
(a). **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
(b). **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
*(c)***File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
(d). **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory.
The storage manager implements several data structures as part of the physical system implementation:
• **Data files**, which store the database itself.
• **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
• **Indices**, which provide fast access to data items that hold particular values.

(2). The Query Processor
The query processor components include:
(a). **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
(b). **DML** compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs
(c). **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.
(d). **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

(3). Run time database manager:

- Run time database manager is the central software component of the DBMS, which interfaces with user-submitted application programs and queries. It handles database access at run time.
- It converts operations in user's queries coming directly via the query processor or indirectly via an application program from the user's logical view to a physical file system.
- It accepts queries and examines the external and conceptual schemas to determine what conceptual records are required to satisfy the user's request.
- It enforces constraints to maintain the consistency and integrity of the data, as well as its security. It also performs backing and recovery operations.

Run time database manager has the following components:
• **Authorization control**: The authorization control module checks the authorization of users in terms of various privileges to users.
• **Command processor**: The command processor processes the queries passed by authorization control module.
• **Integrity checker:** It .checks the integrity constraints so that only valid data can be entered into the database.
• **Transaction manager**: The transaction manager ensures that the transaction properties should be maintained by the system.
• **Scheduler:** It provides an environment in which multiple users can work on same piece of data at the same time in other words it supports concurrency.
• **Data Manager:** The data manager is responsible for the actual handling of data in the database. It provides recovery to the system so that system should be able to recover the data after some failure. It includes Recovery manager and Buffer manager. The buffer manager is responsible for the transfer of data between the main memory and secondary storage (such as disk or tape). It is also referred as the cache manager.



Structure and Components of DBMS

Fig.4.1. Architecture of DBMS

Q.14. Explain the responsibilities of DBA. Explain the problem that would arise if the responsibility were not discharged.

CS: S-12 7M

Or

What are the various responsibilities of DBA?

CS: S-11 6M

Or

What are different role played by DBA? Also give various types of users for the databases.

CS: W-11,W-13 7M

Or

What are the responsibilities of database manager ? For each responsibility, explain the problem that would aries if the responsibilities were not discharged.

CT: W-06 7M

Ans: Typically there are three types of users for a DBMS. They are :

1. The End User who uses the application. Ultimately, this is the user who actually puts the data in the system into use in business.
2. The Application Programmer who develops the application programs. He has more knowledge about the data and its structure.
3. The Database Administrator (DBA) who is like the super-user of the system. The role of the DBA is very important and is defined by the following functions.

- **Defining the Schema**

The DBA defines the schema which contains the structure of the data in the application. The DBA determines what data needs to be present in the system and how this data has to be represented and organized.

- **Coordinate with Users**

The DBA needs to interact continuously with the users to understand the data in the system and its use.

- **Defining Security & Integrity Checks**

The DBA finds about the access restrictions to be defined and defines security checks accordingly. Data Integrity checks are also defined by the DBA.

- **Defining Backup / Recovery Procedures**

The DBA also defines procedures for backup and recovery. Defining backup procedures includes specifying what data is to be backed up, the periodicity of taking backups and also the medium and storage place for the backup data.

- **Monitoring Performance**

The DBA has to continuously monitor the performance of the queries and take measures to optimize all the queries in the application.

Other tasks include the following:

- **Schema Definition**

The DBA creates the original database schema by executing a set of data definition statements in the DDL.

- **Schema and physical-organization modification**.

The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization. Also to alter the physical organization to improve performance.

- **Granting of authorization for data access** :

By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.
The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

- **Routine maintenance**.

Examples of the database administrator's routine maintenance activities are:
- Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
- Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Q.15. Explain the different types of users of DBMS

CT: S-09, W-11 6M

Ans:

There are three types of database system , differentiated by the way that they expect to interact with the system.

(1). Application Programmers or Ordinary users
(2). End users
(3). Database Administrator (DBA)
(4). System Analyst
(5). Sophisticated users
(6). Naïve users

(1). Application programmers or Ordinary users:

- These users write application programs to interact with the database.
- Application programs can be written in some Programming Languages such a COBOL, PL/I, C++, JAVA or some higher level fourth generation language. Such programs access the database by issuing the appropriate request, typically a SQL statement to DBMS.

(2). End Users:

- End users are the users, who use the applications developed.
- End users need not know about the working, database design, the access mechanism etc.
- They just use the system to get their task done. End users are of two types:

a) Direct users b) Indirect users

a) Direct users:

- Direct users are the users who see the computer, database system directly, by following instructions provided in the user interface.
- They interact using the application programs already developed, for getting the desired result.
- Example: People at railway reservation counters, who directly interact with database.

b) Indirect users:

- Indirect users are those users, who desire benefit from the work of DBMS indirectly.
- They use the outputs generated by the programs, for decision making or any other purpose.
- They are just concerned with the output and are not bothered about the programming part.

(3). Database Administrator (DBA):

- Database Administrator (DBA) is the person which makes the strategic and policy decisions regarding the data of the enterprise.
- He provides the necessary technical support for implementing these decisions.
- Therefore, DBA is responsible for overall control of the system at a technical level.
- In database environment, the primary resource is the database itself and the secondary resource is the DBMS and related software administering is the responsibility of the Database Administrator (DBA).

(4). System Analyst:

- System Analyst determines the requirement of end users, especially naïve and parametric end users.
- He develops specifications for transactions that meet these requirements.
- System Analyst plays a major role in database design, its properties.
- He prepares the system requirement statement, which involves the feasibility aspect, economic aspect, technical aspect etc. of the system.

(5). Sophisticated users:

- Sophisticated users interact with the system without writing programs.
- Instead, they form their requests in a database query language.
- They submit each such query to a query processor, whose function is to break down DML

statements into instructions that the storage manager understands.
Analysts who submit queries to explore data in the database fall in this category.

(6). Naive users:

- **Naive users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
- Example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*.
- This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

.

Q.16. Write short notes on the following:

CS: S-11, W-10 13M

(i). Transaction Management.
(ii). Storage Management
(iii). Data Dictionary.
(iv). Data Materialization

Ans.

(i). Transaction Management

- A transaction is a collection of operations that performs a single logical function in a database application.
- Transactions in a database environment have following purposes:

1. To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure.
2. When execution stops (completely or partially) and many operations upon a database remain uncompleted, transactions help to recover from these situatios.
3. To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the program's outcome contains error.
4. A database transaction, by definition, must be atomic, consistent, isolated and durable

- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g. power failures and operating system crashes).
- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.
- The transaction Manager ensures that the database remains in a consistent state despite system failure.
- It also concentrates on non conflicting execution of concurrent Transactions.

(ii). Storage Management

- Storage manager is a program module in DBMS that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for interaction with the file manager & efficient storing, retrieving and updating of data.
- In relational Database Management Systems, the database is placed on secondary storage (disk).
- When the data in the database are required, they need to be read into memory and may be written back to disk if updated.
- The data must be organized on the disk for efficient accessing.
- In order to increase the efficiency of storage management, part of memory is used to cache some of these data.

- The management of data on disk and in memory belongs to the storage management of DBMS.
- The disk access is the bottleneck of the database system in many workloads, because the access time of disk is several magnitudes longer than that of memory.
- The objective of storage management is to improve the efficiency of disk access, therefore to improve the performance of the whole system.
- The storage management can be divided into three levels:

(1). Logical level is the higher level. In this level, the logical information of the DBMS is used to optimize the data stored in memory

(2). Physical in-memory level is the middle level. In this level, the popular database pages are cached in the in-memory space buffer pool.

- The buffer pool can absorb most requests to the disk therefore reduce the number of disk accesses.
- This approach is used in almost all DBMSs and has important effects on their performance.

(3). Physical on-disk level is the lower level. The layout of database data and meta-data on the disk is studied to improve the disk I/O efficiency.

- It can be further divided into two sub-levels: software level and disk level.
- In the software level, the on-disk data structure and data placement strategy are designed based on the logical structure of data.

(iii). Data Dictionary:

- A **data dictionary**, or metadata repository, as defined as a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format".

- This term may have variety of definitions as given below:
  (1). a document describing a database or collection of databases
  (2). an integral component of a DBMS that is required to determine its structure
  (3). a piece of middleware that extends or supplants the native data dictionary of a DBMS
- The data dictionary (or data repository) is an important part of the DBMS which is also called as database catalog.
- It contains data about data (or metadata).
- It means that it contains the actual database descriptions used by the DBMS.
- In most DBMSs, the data dictionary is active and integrated.
- It means that the DBMS checks the data dictionary every time the database is accessed.
- The data dictionary contains the following information.

(a). Logical structure of database
(b). Schemas, mappings and constraints
(c). Description about application programs.
(d).Descriptions of record types, data item types, and data aggregates in the database.
(e). Description about storage structures, access paths etc.
(f). Descriptions about users of DBMS and their access rights.

- A data dictionary may be a separate part of DBMS (non-integrated data dictionary).
- It may be a commercial product or a simple file developed and maintained by a designer.
- This type of data dictionary is referred to as freestanding data dictionary.
- This type of data dictionary is useful in the initial stage of design for collecting and organizing information about data.
- Example: Each data item, its source, its name, its uses, its meaning, its relationship to other items etc. is stored in data dictionary.

(IV). Data Materialization:

- A **materialized view or data materialization** is a database object that contains the results of a query.
- Example: It may be a local copy of data located remotely, or may be a subset of the rows and/or columns of a table or join result, or may be a summary based on aggregations of a table's data.
- Materialized views, which store data based on remote tables, are also known as snapshots.
- A snapshot can be redefined as a materialized view.
- Materialized views were implemented first by the Oracle Database: the Query rewrite feature was added from version 8i.
- They are also supported in Sybase SQL Anywhere.
- In IBM DB2, they are called "materialized query tables"; -Microsoft SQL Server has a similar feature called "indexed views" and MySQL doesn't support materialized views natively, but workarounds can be implemented by using triggers or stored procedures

Q.17. Define the following: Data Model. CT: S-06, 2M

Ans.

- Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
- Data model tells how the logical structure of a database is modeled.
- Data Models are fundamental entities to introduce abstraction in DBMS.
- They define how data is connected to each other and how it will be processed and stored inside the system.
- The very first data model could be flat data-models where all the data used to be kept in same plane.
- Because earlier data models were not so scientific they were prone to introduce lots of duplication and update anomalies.
- Numerous data models have been in use and can be broadly categorized as under:

1. Object based data Models
2. Record-based Data Models
   - Relational Data Model
   - Network Model
   - Hierarchical Model
3. Physical Data model
4. Entity –Relationship Model

Q.18. What are the different data models?

CS: W-12, S-11 7M

Or

Define Data Model. State and explain Hierarchical, Network and E-R Data Model.

CS: S-12, W-07, W-13 7M

Or

Compare Hierarchical, Relational and Network data models with respect to the following operations:i) insertion ii) deletion iii)modification .

CT: W-06, S-06 6M

Ans:

Data Model:

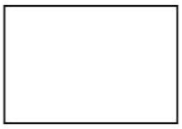
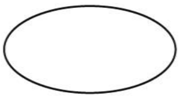
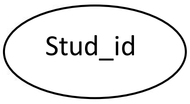
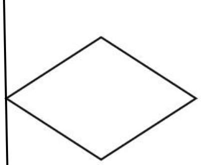
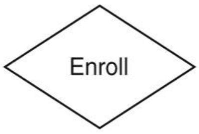
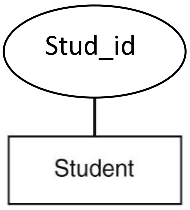
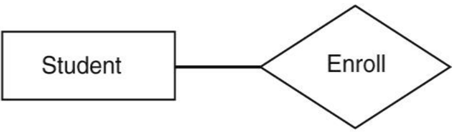
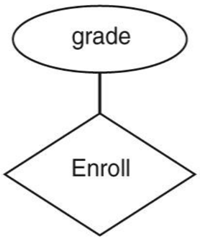
- A data model is a collection of conceptual tools used for describing data, data relationships, data semantics and data constraints.
- The model should enable the designer to incorporate major portion database in the schema
- The data models have been broadly classified into following types:

(1). Entity –Relationship Model
(2). Record-based Data Models
 - Relational Data Model
 - Network Model
 - Hierarchical Model
(3). Physical Data model
(4). Object based data Models

(1). Entity-Relationship Model

- The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities*, and of *relationships* among these objects.
- An entity is a “thing” or “object” in the real world that is distinguishable from other objects.
- Example: Each person is an entity, and bank accounts can be considered as entities.
- Entities are described in a database by a set of **attributes**.
- Example: The attributes *account-number* and *balance* may describe one particular account in a bank, and they form attributes of the *account* entity set.
- Similarly, attributes *customer-name*, *customer-street* address and *customer-city* may describe a *customer* entity.
- An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city). A unique customer identifier must be assigned to each customer.
- In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.
- A **relationship** is an association among several entities. For example, a *depositor* relationship associates a customer with each account that he/she has.
- The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.
- The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*, which is built up from the following components:
  (i). Entities are real-world objects about which we collect data
  (ii). Attributes describe the entities
  (iii). Relationships are associations among entities
  (iv). Entity set – set of entities of the same type
  (v). Relationship set – set of relationships of same type

Relationships sets may have descriptive attributes Represented by E-R diagrams as given below:

SYMBOL	NAME	MEANING	EXAMPLE
	Rectangle	Entity Set	
	Oval	Attribute	
	Diamond	Relationship	
	Line	Links: Attribute to Entity	
		Entity Set to Relationship	
		Attribute to Relationship	

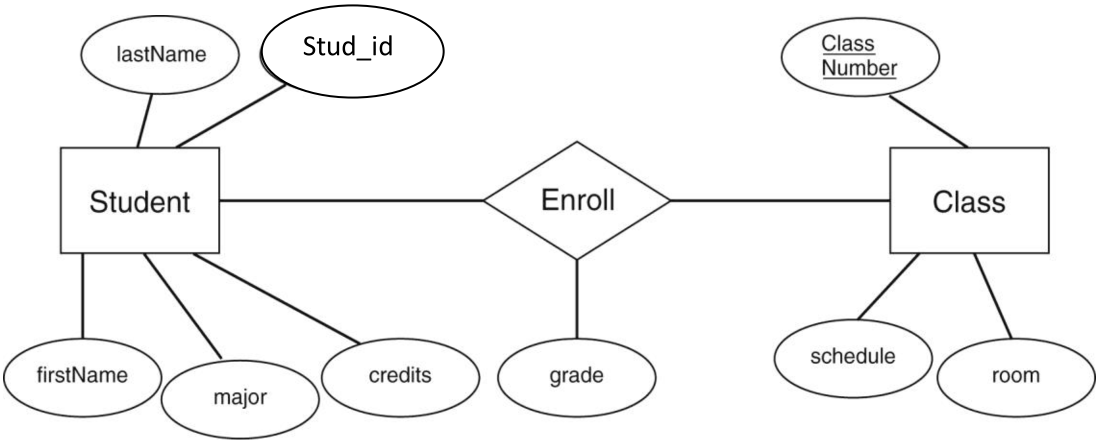


Fig. . Example of E-R Diagram

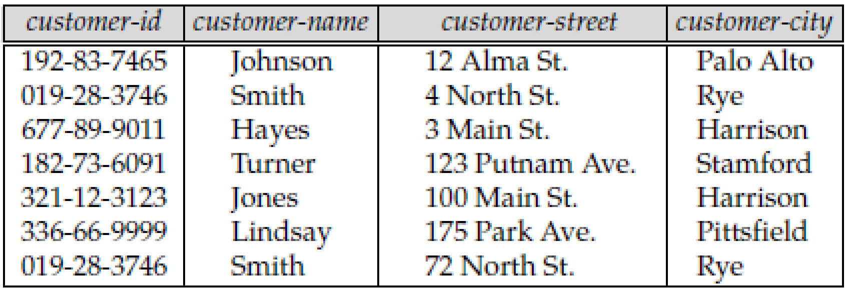
(2). Record- and table-based model

(A). Relational Data Model

- Relational database modeling is a ***logical-level*** model
- It was Proposed by E.F. Codd
  - Based on mathematical relations
  - Uses relations, represented as tables
  - Columns of tables represent attributes
  - Tables represent relationships as well as entities

Successor to earlier record-based models are network network and hierarchical

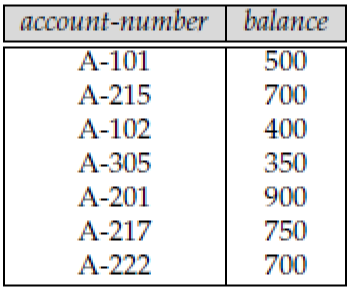
- The relational model uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns, and each column has a unique name. Figure 10.2 presents a sample relational database comprising three tables:
- Example: One shows details of bank customers, the second shows accounts, and the third shows which accounts belong to which customers.
- The first table, the customer table, shows, that the customer identified by customer-id 192-83-7465 is named Johnson and lives at 12 Alma St. in Palo Alto.
- The second table, account, shows, that account A-101 has a balance of $500, and A-201 has a balance of $900.
- The third table shows which accounts belong to which customers. Account number A-101 belongs to the customer whose customer-id is 192-83-7465, namely Johnson, and customers 192-83-7465 (Johnson) and 019-28-3746 (Smith) share account number A-201 (they may share a business venture).
- The relational model is an example of a record-based model.
- Record-based models are so named because the database is structured in fixed-format records of several types.
- Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes.
- The columns of the table correspond to the attributes of the record type
- The relational model hides such low-level implementation details from database developers and users.



customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

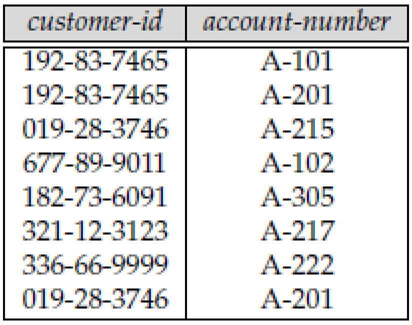
(a) The *customer* table

Fig. A Relational Data Model



account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table



customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Fig. 10.2. A Relational Data Model

(B). Network Model:

- The network model differs from the relational model in that data are represented by collections of records, and relationships among data are represented by links.
- A network database consists of a collection of records connected to one another through links.
- A record is in many respects similar to an entity in the E-R model.
- Each record is a collection of fields (attributes), each of which contains only one data value.\
- A link is an association between precisely two records.
- Thus, a link can be viewed as a restricted (binary) form of relationship in the sense of the E-R model. The Fig. shows an example:

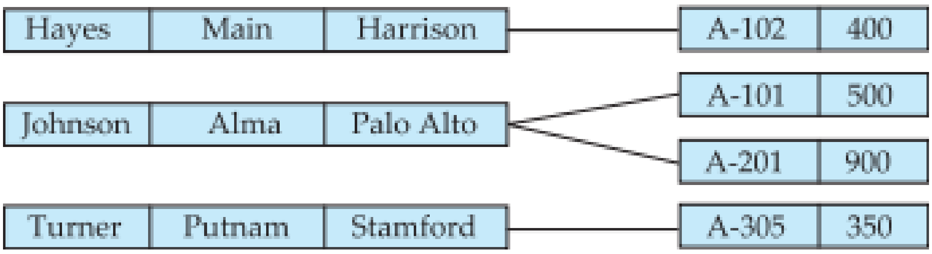


Fig. A Network data Model

(C). Hierarchical Model:

- A **hierarchical database model** is a data model in which the data is organized into a tree-like structure.
- The structure allows representing information using parent/child relationships: each parent can have many children, but each child has only one parent (also known as a **1-to-many relationship**).
- All attributes of a specific record are listed under an entity type.

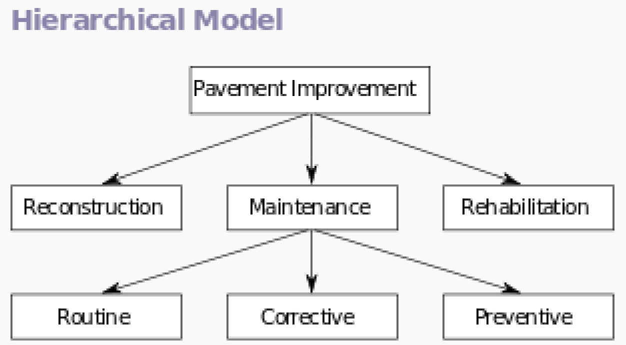


Fig. Example of a hierarchical model

- In a database an entity type is the equivalent of a table. Each individual record is represented as a row, and each attribute as a column.
- Entity types are related to each other using *1:N* mappings, also known as one-to-many relationships. This model is recognized as the first database model created by IBM in the 1960s.
- Currently the most widely used hierarchical databases are IMS developed by IBM and Windows Registry by Microsoft.

(3). Object Oriented Data Model:

- An **object database** (also **object-oriented database management system**) is a database management system in which information is represented in the form of objects as used in object-oriented programming.
- Object databases are different from relational databases which are table-oriented.
- Object-relational databases are a hybrid of both approaches.
- Uses the E-R modeling as a basis but extended to include **encapsulation**, **inheritance**
- Objects have both state and behavior
    - **State** is defined by attributes
    - **Behavior** is defined by methods (functions or procedures)
- Designer defines classes with attributes, methods, and relationships
- Class constructor method creates object instances. Each object has a unique object ID
    - Classes related by class hierarchies
    - Database objects have persistence

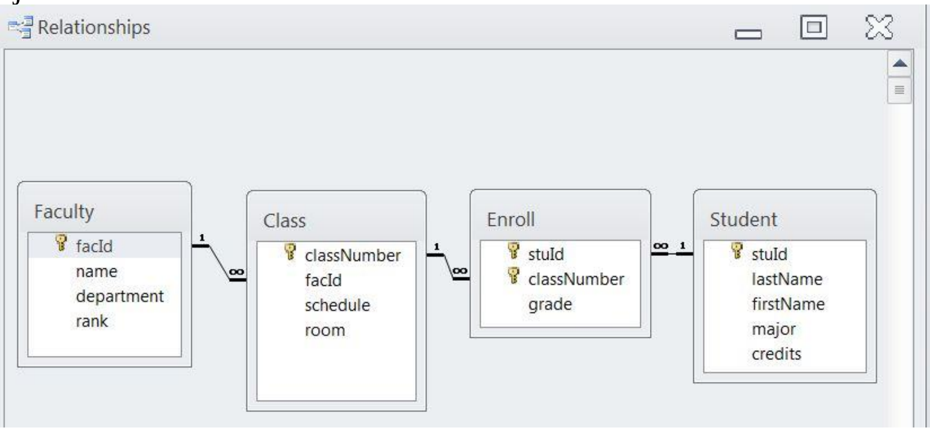


Fig. An Object Oriented data Model

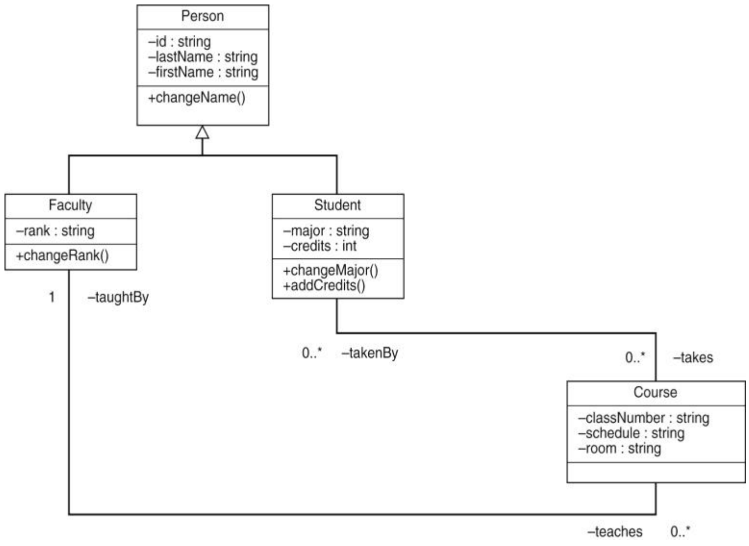


Fig. An Object Oriented data Model

(4). Physical Data Model:

- A **physical data model** (or database design) is a representation of a data design which takes into account the facilities and constraints of a given database management system.
- In the lifecycle of a project it typically derives from a logical data model, though it may be reverse-engineered from a given database implementation.
- A complete physical data model will include all the database artifacts required to create relationships between tables or to achieve performance goals, such as indexes, constraint definitions, linking tables, partitioned tables or clusters.
- Analysts can usually use a physical data model to calculate storage estimates; it may include specific storage allocation details for a given database system.

Q.19. Explain DBTG CODASYL model.

CT: S-07, W-13 7M

Ans:

- The acronym DBTG refers to the Data Base Task Group of the Conference on Data Systems Languages (CODASYL), the group responsible for standardization of the programming language COBOL.
- Many computer systems have been implemented using the network Database Management System (DBMS) specifications developed by the Conference on Data Systems Languages (CODASYL).
- There are two subgroups of CODASYL: the Database Task Group (DBTG), and the Data Description Language Committee (DDLC) .
- The DBTG is intended to meet the requirements of many distinct programming languages, not just COBOL.
- The user in a DBTG system is considered to be an ordinary application programmer and the language therefore is not biased toward any single specific programming language.
- The Data Base Task Group (DBTG) CODASYL Specifications included Schema definition, Device Media Control Language (DMCL) definition, Data Manipulation Language (DML) definition.
- It also included the concept of a database "area" which referred to the Physical Structure of the data files.

- The Logical Structure of the database was defined by a Data Definition Language (DDL), and a user view of the data was defined by a subschema.

- **The Data Manipulation Language (DML) commands** were used to navigate through the linked-list structures that comprised the database, much the same as object-oriented databases are navigated in C++.
- The CODASYL DML verbs included FIND, GET, STORE, MODIFY, and DELETE.
- **The Data Base Administrator (Remote DBA)** functions included: data structure or schema, data integrity, security, and authorization.
- Also a Data Base Manager (DBM) function was defined which included: operation, backup/recovery, performance, statistics, auditing.

Architecture of DBTG **Model**

The architecture of a DBTG system is illustrated in Figure.
The architecture of DBTG model can be divided in three different levels as the architecture of a' database system. These are:
• Storage Schema (corresponds to Internal View of database)
• Schema (corresponds to Conceptual View of database)
• Subschema (corresponds to External View of database)

Storage Schema

The storage structure (Internal· View) of the database is described by the storage schema, written in a Data Storage Description Language (DSDL).

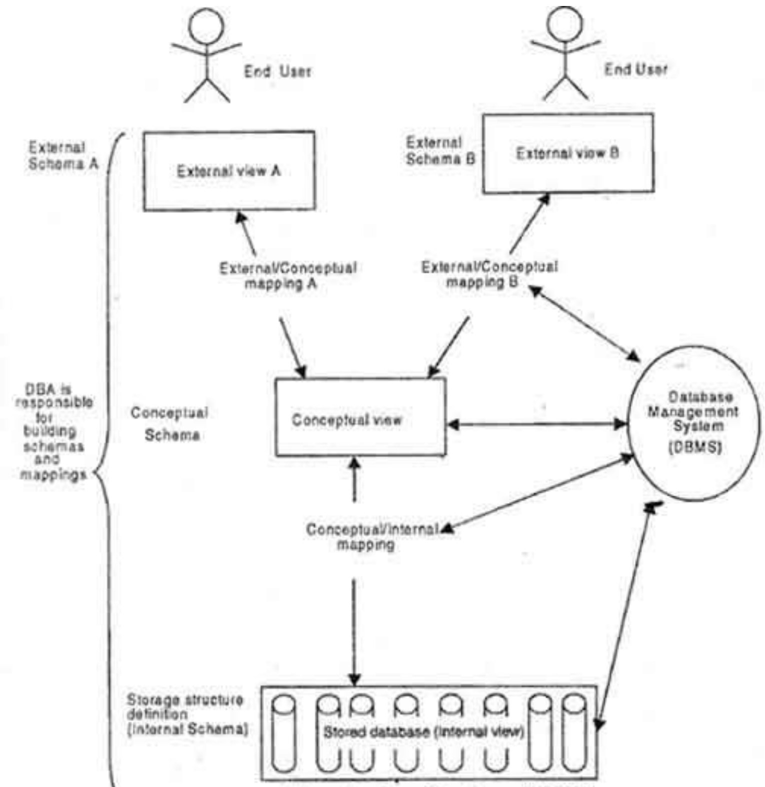
Schema

- In DBTG, the Conceptual View is defined by the schema.
- The schema consists essentially of definitions of the various type of record in the database, the data-items they contain, and the sets into which they are grouped. (Here, logical record types are· referred to as record types, the fields in a logical record format are called data items)

Subschema

- The External view (not a DBTG term) is defined by a *subschema*.
- A subschema consists essentially of a specification of which schema record types the user is interested in, which schema data-items he or she wishes to see in those records, and which schema relationships (sets) linking those records he or she wishes to consider.
- By default, all other types of record, data-item, and set are excluded.
- In DBTG model, the users are application programmers, writing in an ordinary programming language, such as COBOL that has been extended to include the DBTG data manipulation language.
- Each application program "invokes" the corresponding subschema; using the COBOL Data Base Facility, for example, the programmer simply specifies the name of the required subschema in the Data Division of the program.
- This invocation provides the definition of the "user work area" (UWA) for that program.

- The UWA contains a distinct location for each type of record (and hence for each type (data-item) defined in the subschema.
- The program may refer to these data-item and record locations by the names defined in the subschema.


End User
End User
External Schema A
External view A
External Schema B
External view B
External/Conceptual mapping A
External/Conceptual mapping B
DBA is responsible for building schemas and mappings
Conceptual Schema
Conceptual view
Database Management System (DBMS)
Conceptual/Internal mapping
Storage structure definition (Internal Schema)
Stored database (Internal view)

Three level architecture of DBMS

Q.20. What are the various DBTG data structure? Explain each brief?

CT: W-07, W-10, S-13, S-12 6M

Ans.

- The Network Data Model is also known as the "CODASYL Data Model" or sometimes as the "DBTG Data Model".

The DBTG report contained proposals for three distinct database languages:
(i). a schema data description language;
(ii). a subschema data description language;
(iii). a data manipulation language.

The DBTG data Structures are described as under:

Database Record

- A network data base consists of so-called records. A Record (i.e. Record Occurrence) is a collection of data items.
- Each data item has a name and a value. Every record describes some real person, object or event of the area being modelled.
- A network Database Managament System (DBMS) operates on records.

- A record (or, more precisely, record occurrence) is a collection of data items which can be retrieved from a data base, or which can be stored in a data base as an undivided object.
- Thus, a DBMS may: STORE, DELETE or MODIFY records within a data base.
- In this way, a number of records within a network database is dynamically changed.
- A CODASYL record may have its own internal structure.
- Two or more contiguous elementary items may be grouped together to form a group item. Network (CODASYL) Data Model

- A group item may consist not only of elementary items but also of other group items, hence allowing the user to build up a naming structure.
- To avoid confusion, the levels in this structure must be numbered downwards from the top.
- A CODASYL record may include so-called tables. A table is collection of values grouped under one name of a data item.
- The user references the elements in the table using subscripting similar to an array in

ordinary programming. For example:
Persons.Name.Monthly_Income[0]
A CODASYL record allows duplicate names of data items.

- Suppose the user needs to refer to the attribute Name which is an item of both records.
- In order to distinguish one from the other, the user must write:
  Supplier.Name and Product.Name
- This technique is known as a qualification. On complex case, the qualification carries through all levels of naming within the record.
- The qualification can be omitted if the user refers to the unique name of a data item.
- There may be two or more records with the same internal structure, or more precisely, which include different values of the same attributes.
- A collection of such record occurrences is called a Record Type. Each record type has a unique name.
- We can thus say that different record occurrences of a same record type describe different instances of a certain entity of the "real word". Network (CODASYL) Data Model
- In other words, a record type is a frame (template) for the real data representation.

Data Base Key

- Two or more different records within a network data base may have duplicate values of all data items.
- The Data Base Key (DBK) is conceptually a data item, whose value is associated with each stored record in the data base.
- We can think of it as a unique internal record identifier used inside a data base to distinguish one record from another.
- Each record is assigned a data base key value when it is stored in the data base for the first time.
- A record retains a value of the Data Base Key even if the record is modified until the record isfinally deleted from the data base.
- In some ways, the data base key for the CODASYL record is like a social security number or a personal identification number.

Data Set

- Normally, the information model consists of two main parts: Data Objects and Relationships. Network (CODASYL) Data Model

- In accordance with the Data Base Task Group (DBTG) proposals, each record directly corresponds to the concrete entity, but relationship between the records are implemented by means of a special logical construction.
- This logical construction is called a Data Set (or simply Set).
- In the simpliest case, each set (or more precisely, set occurrence) consists of records of two different types (e.g. Father and Child).
- The data set has the following properties:
  1. Each set includes exactly one record of the first type. This record is called an Owner of the set.
  2. Each set may include 0 (i.e. an Empty set occurrence), 1 or N records of the same type. These records are called members of the data set.
  3. All members within one set occurrence have a fixed order (are sorted).
  4. There may be two or more sets consisting of records of the same types and describing the same relationship between records.
  5. A collection of such set occurrences is called a set type. Each set type has a unique name. Network (CODASYL) Data Model

- We can thus say that different set occurrences of the same set type describe different instances of a certain relationship between entities of the area being modelled.
- Thus, the main data structure types used in the network data model are: record type and set type.
- In other words, according to the network data model the information within a database is arranged as a collection of record occurrences and a collection of set occurrences.
- All record types and all set types must be described in the data base schema.
- The data base consists of record occurrences and of set occurrences of such types as were previously defined in the data base schema.

Q.21. Explain Entity-Relationship model with proper example.. 7M

Ans : -

Entity-Relationship Model:

- The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities*, and of *relationships* among these objects.
- An entity is a “thing” or “object” in the real world that is distinguishable from other objects.
- Example, each person is an entity, and bank accounts can be considered as entities.
- Entities are described in a database by a set of **attributes**.
- Example, the attributes *account-number* and *balance* may describe one particular account in a bank, and they form attributes of the *account* entity set.
- Similarly, attributes *customer-name*, *customer-street* address and *customer-city* may describe a *customer* entity.
- An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city).
- A unique customer identifier must be assigned to each customer.
- In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.
- A **relationship** is an association among several entities. For example, a *depositor* relationship associates a customer with each account that he/she has.

- The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.
- The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*, which is built up from the following components:
  - **Entities** are real-world objects about which we collect data
  - **Attributes** describe the entities
  - **Relationships** are associations among entities
  - **Entity set** – set of entities of the same type
  - **Relationship set** – set of relationships of same type

- The figure shows the various components of E-R diagram.
- Relationships sets may have descriptive attributes Represented by E-R diagrams as given below:

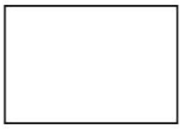
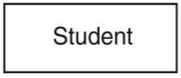
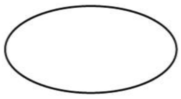
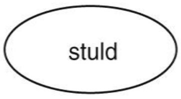
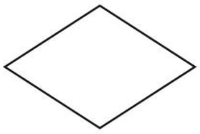
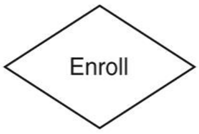
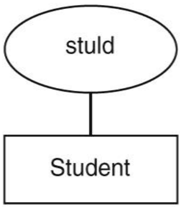
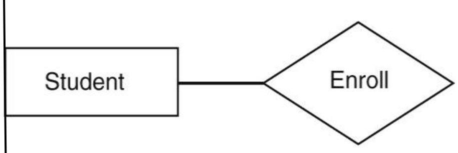
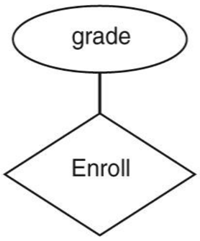
SYMBOL	NAME	MEANING	EXAMPLE
	Rectangle	Entity Set	
	Oval	Attribute	
	Diamond	Relationship	
	Line	Links: Attribute to Entity	
		Entity Set to Relationship	
		Attribute to Relationship	

figure. components of E-R Model

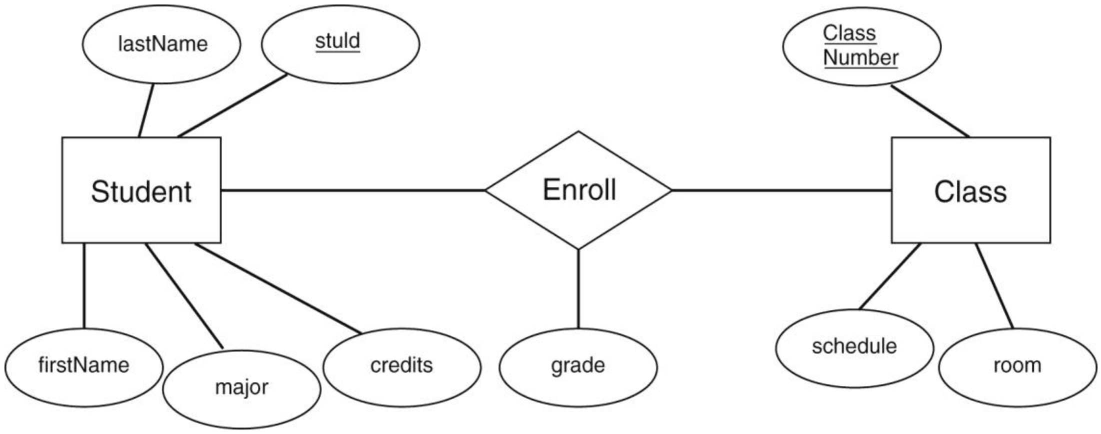


Fig. 10.2. Example of E-R Diagram

Session 2018-19 (Even Semester)

Sixth Semester **Subject: Database Management System**

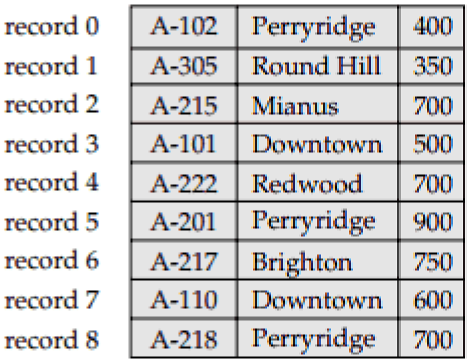
Notes on Unit-II: File Organization, Indexing and Hashing

Q.1. What are the various ways of organizing records in files? Explain sequential file organization.

CT: W-12 6M

Ans.

- A file is sequence of records stored in binary format.
- A disk drive is formatted into several blocks, which are capable for storing records.
- File records are mapped onto those disk blocks. The fig below shows the file structure.



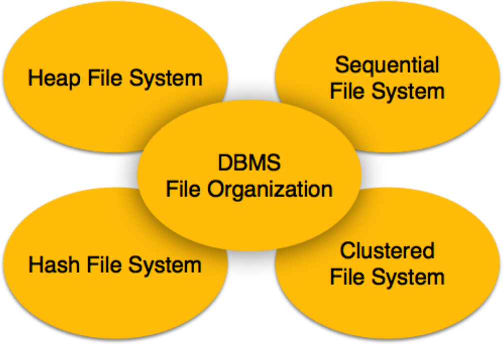
record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

File containing *account* records.

- The blocks are of a fixed size which is determined by the physical properties of the disk operating system, but the size may vary.
- The records which make up the block can be of fixed size or variable size.
- Files with fixed length records are easier to manage & implement.

a) File Organization

- The method of mapping file records to disk blocks defines file organization, i.e. how the file records are organized.
- The following are the types of file organization



(i) Heap File Organization:

- When a file is created using Heap File Organization mechanism, the Operating Systems allocates memory area to that file without any further accounting details.
- File records can be placed anywhere in that memory area.
- It is the responsibility of software to manage the records.
- Heap File does not support any ordering, sequencing or indexing on its own.

(ii) Sequential File Organization:

- Every file record contains a data field (attribute) to uniquely identify that record.
- In sequential file organization mechanism, records are placed in the file in the some sequential order based on the unique key field or search key.
- Practically, it is not possible to store all the records sequentially in physical form.

(iii) Hash File Organization:

- This mechanism uses a Hash function computation on some field of the records.
- As we know, that file is a collection of records, which has to be mapped on some block of the disk space allocated to it.
- This mapping is defined that the hash computation.
- The output of hash determines the location of disk block where the records may exist.

(iv) Clustered File Organization:

- Clustered file organization is not considered good for large databases.
- In this mechanism, related records from one or more relations are kept in a same disk block, that is, the ordering of records is not based on primary key or search key.
- This organization helps to retrieve data easily based on particular join condition.

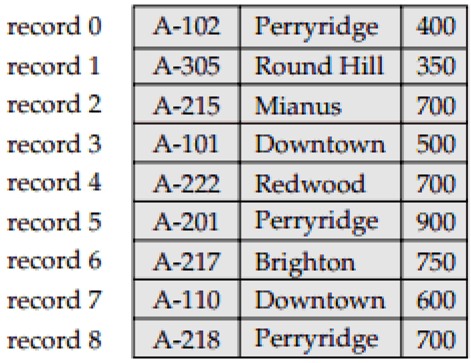
Q.2. Explain the fixed length records & their usage in DBMS file structure. 6M

Ans.

- Consider a file of *account* records for our bank database.
- Each record of this file is defined as:

  type *deposit* = **record**
  account-number : char(10);
  branch-name : char (22);
  balance : real;
  end

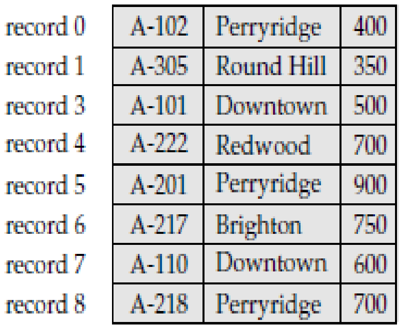
- If we assume that each character occupies 1 byte and that a real occupies 8 bytes, our *account* record is 40 bytes long.
- A simple approach is to use the first 40 bytes for the first record, the next 40 bytes for the second record, and so on (Figure shown below).



record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

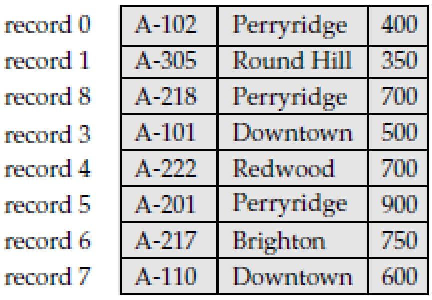
File containing *account* records.

- However, there are two problems with this simple approach:
  1. It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file.
  2. Unless the block size happens to be a multiple of 40 (which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.
  When a record is deleted, we could move the record that came after it into the space formerly occupied by the deleted record, and so on, until every record following
  the deleted record has been moved ahead (Figure shown below).



record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

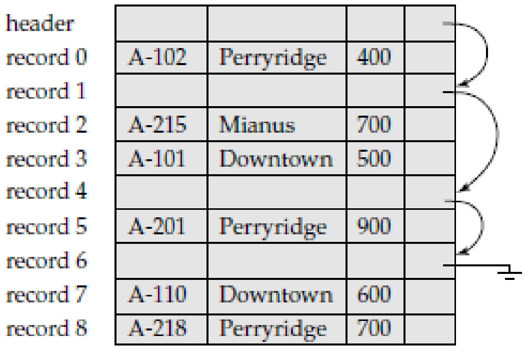
- Such an approach requires moving a large number of records.
- It might be easier simply to move the final record of the file into the space occupied by the deleted record (Figure shown below)



record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 8	A-218	Perryridge	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600

- It is undesirable to move records to occupy the space freed by a deleted record, since doing so requires additional block accesses.
- Since insertions are more frequent than deletions, it is acceptable to leave open the space occupied by the deleted record, and to wait for a subsequent insertion before reusing the space.

- Thus, we need to introduce an additional structure.
- At the beginning of the file, we allocate a certain number of bytes as a **file header**.
- The header will contain a variety of information about the file.
- We need to store there is the address of the first record whose contents are deleted.
- We use this first record to store the address of the second available record, and so on.
- We can think of these stored addresses as *pointers*, since they point to the location of a record.
- The deleted records thus form a linked list, which is often referred to as a **free list**. Figure below shows the file of Figure 1, with the free list, after records 1, 4, and 6 have been deleted.
- On insertion of a new record, we use the record pointed to by the header.
- We change the header pointer to point to the next available record.
- If no space is available, we add the new record to the end of the file.
- Insertion and deletion for files of fixed-length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record.
- If we allow records of variable length in a file, this match no longer holds. An inserted record may not fit in the space left free by a deleted record, or it may fill only part of that space.



header				
record 0	A-102	Perryridge	400	
record 1				
record 2	A-215	Mianus	700	
record 3	A-101	Downtown	500	
record 4				
record 5	A-201	Perryridge	900	
record 6				
record 7	A-110	Downtown	600	
record 8	A-218	Perryridge	700	

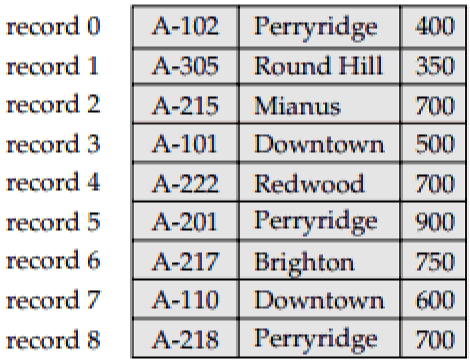
Q.3. Explain the variable length records in detail with example. **4M**

Ans.

- Variable-length records arise in database systems in several ways:

• Storage of multiple record types in a file
• Record types that allow variable lengths for one or more fields
• Record types that allow repeating fields

- Different techniques for implementing variable-length records exist.
- Consider a different representation of the *account* information stored in the file of Figure shown below, in which one variable-length record is used for each branch name and for all the account information for that branch.



record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

File containing *account* records.

- The format of the record is

```
`type account-list = record
branch-name : char (22);
account-info : array [1 ..∞] of
record;
account-number : char(10);
balance : real;
end
end
```

- We define *account-info* as an array with an arbitrary number of elements.
- That is, the type definition does not limit the number of elements in the array, although any actual record will have a specific number of elements in its array.
- There is no limit on how large a record can be (up to, of course, the size of the disk storage.

CONCEPT OF INDEXING

Q.4. What is index and state the types of indexes. 7M

Ans.

- Many queries reference only a small proportion of the records in the file.
- Example, To find all accounts at the perriridge branch "references only a fraction of the account records.
- It is efficient for the system to read every record and to check the branchname field for the name "perryridge".
- For quick output, the system should be able to locate these records directly.
- To perform such type of accessing, the index access methods are used.
- An index for a file in DBMS as same as the index in the textbook.
- If we want to learn about a particular topic, we can search the topic in the index at the back of the book, find the page no and read the content on that page no.
- The words in this index are in sorted order, making it easy to find the word and locate the content directly.

- The index is much smaller than the book.
- Similar is the case for the index structure for the file.
- Example: to find the account record given any account no, DBMS will look into the index to find on which disk block the record needed is located and then fetch the disk block to get the account number.
- This technique is used for data blocks where database is of smaller sizes.
- For the database system for larger sizes the sophisticated techniques are used.

a). Types of Indexes

- Indexing is a data structure technique to efficiently retrieve records from database files based on some attributes on which the indexing has been done.
- Basic Indexing types can be one of the following types:

1. **Ordered index:** It is based on a sorted ordering of the values.
   (i). Primary Index: If index is built on ordering 'key-field' of file it is called Primary Index. Generally it is the primary key of the relation.
   (ii). Secondary Index: If index is built on non-ordering field of file it is called Secondary Index.
   (iii). Clustering Index: If index is built on ordering non-key field of file it is called Clustering Index Ordering field is the field on which the records of file are ordered. It can be different from primary or candidate key of a file.

2. **Hash Index:** It is based on a uniform distribution of values across a range of buckets. The bucket to which a value is assigned is determined by a function, called a *hash function*.

Q.5. On what factors the various types of indexing must be evaluated? 6M

Ans:

- The ordered and hash indexing techniques must be evaluated based on the following factors.
- Each technique must be evaluated on the basis of these factors:

1. **Access types**: The types of access that are supported efficiently. Access types can include finding records with a specified attribute value and finding records whose attribute values fall in a specified range.
2. **Access time**: The time it takes to find a particular data item, or set of items, using the technique in question.
3. **Insertion time**: The time it takes to insert a new data item. This value includes the time it takes to find the correct place to insert the new data item, as well as the time it takes to update the index structure.
4. **Deletion time**: The time it takes to delete a data item. This value includes the time it takes to find the item to be deleted, as well as the time it takes to update the index structure.
5. **Space overhead**: The additional space occupied by an index structure. Provided that the amount of additional space is moderate, it is usually worthwhile to sacrifice the space to achieve improved performance.

Q.6. Give detailed classification of indexing. Explain each type with neat sketch. CT: S-06 7M

Or

Give detailed classification of indexing techniques. Explain each type with neat sketch. CT: W-11 6M

Or

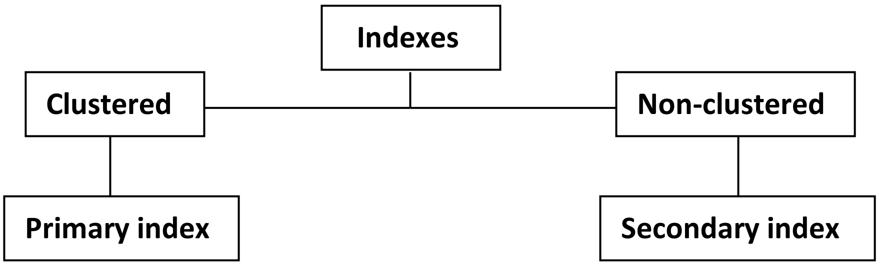
Explain Hash Index, function index and Bitmap index. CSE: W-13 6M

Or

Explain Hash Index, function index and Bitmap index. CSE: W-13 6M

Ans.

- A database index is a data structure that improves the speed of data retrieval operations on a database table .
- Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.
- Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.
- The Classification of Indexes is shown below:



1. Non-clustered index:

- The data is present in arbitrary order, but the logical ordering is specified by the index.
- The data rows may be spread throughout the table regardless of the value of the indexed column or expression.
- The non-clustered index tree contains the index keys in sorted order, with the leaf level of the index containing the pointer to the record.
- In non-clustered index, the physical order of the rows is not the same as the index order.
- The indexed columns are typically non-primary key columns used in JOIN, WHERE, and ORDER BY clauses.
- There can be more than one non-clustered index on a database table.

2. Clustered index:

- Clustering alters the data block into a certain distinct order to match the index, resulting in the row data being stored in order.
- Therefore, only one clustered index can be created on a given database table.

- Clustered indices can greatly increase overall speed of retrieval, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items is selected.
- Since the physical records are in this sort order on disk, the next row item in the sequence is immediately before or after the last one, and so fewer data block reads are required.
- The primary feature of a clustered index is therefore the ordering of the physical data rows in accordance with the index blocks that point to them.
- The types of indexes are explained as under:

(i). Primary Index:
If index is built on ordering 'key-field' of file it is called Primary Index. Generally it is the primary key of the relation.

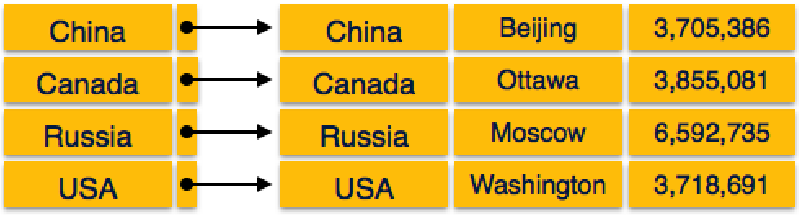
(ii). Secondary Index: If index is built on non-ordering field of file it is called Secondary Index.

(iii). Clustering Index: If index is built on ordering non-key field of file it is called Clustering Index Ordering field is the field on which the records of file are ordered. It can be different from primary or candidate key of a file.

- Ordering field is the field on which the records of file are ordered.
- It can be different from primary or candidate key of a file.
- Ordered Indexing is of two types:
  (i). Dense Index
  (ii).Sparse Index

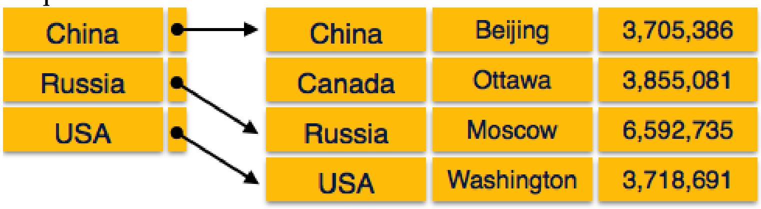
(i). Dense Index:

- In dense index, there is an index record for every search key value in the database.
- This makes searching faster but requires more space to store index records itself.
- Index record contains search key value and a pointer to the actual record on the disk.



(ii)Sparse Index

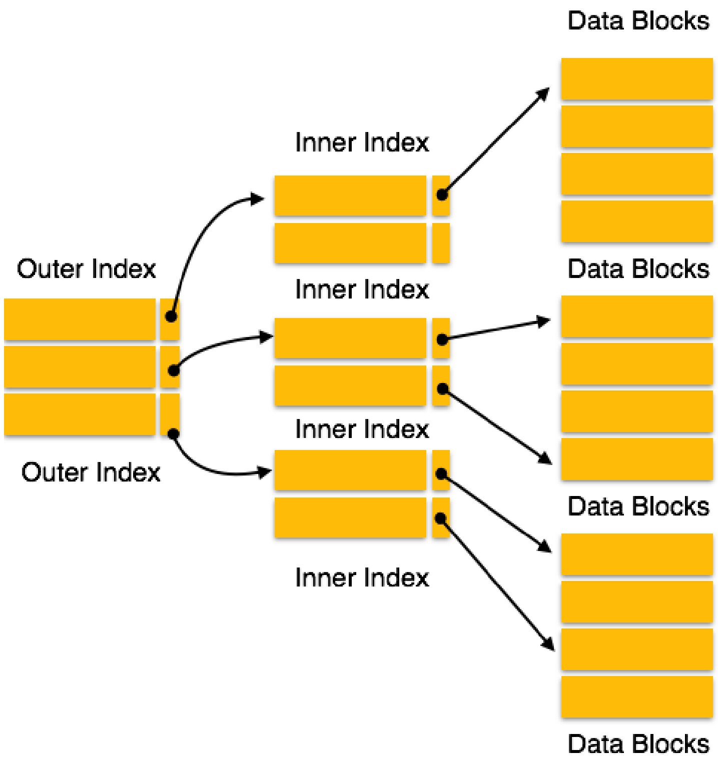
- In sparse index, index records are not created for every search key.
- An index record here contains search key and actual pointer to the data on the disk.
- To search a record we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following index, the system starts sequential search until the desired data is found.



- Other types of indexing structures used are as under:

(iii). Multilevel Index

- Index records are comprised of search-key value and data pointers.
- This index itself is stored on the disk along with the actual database files.
- As the size of database grows so does the size of indices.
- There is an immense need to keep the index records in the main memory so that the search can speed up.
- If single level index is used then a large size index cannot be kept in memory as whole and this leads to multiple disk accesses.

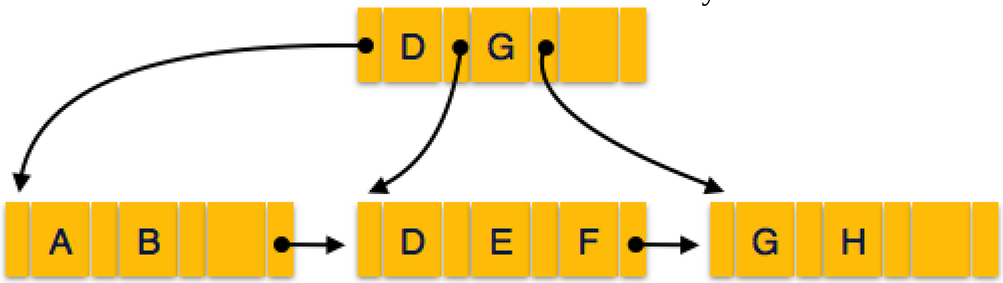


- Multi-level Index helps breaking down the index into several smaller indices in order to make the outer most level so small that it can be saved in single disk block.
- This block can then be easily accommodated anywhere in the main memory.

(iv). B$^+$ Tree:

- B+ tree is multi-level index format, which is balanced binary search trees.
- As mentioned earlier single level index records becomes large as the database size grows, which also degrades performance.
- All leaf nodes of B$^+$ tree denote actual data pointers.
- B$^+$ tree ensures that all leaf nodes remain at the same height, thus balanced.

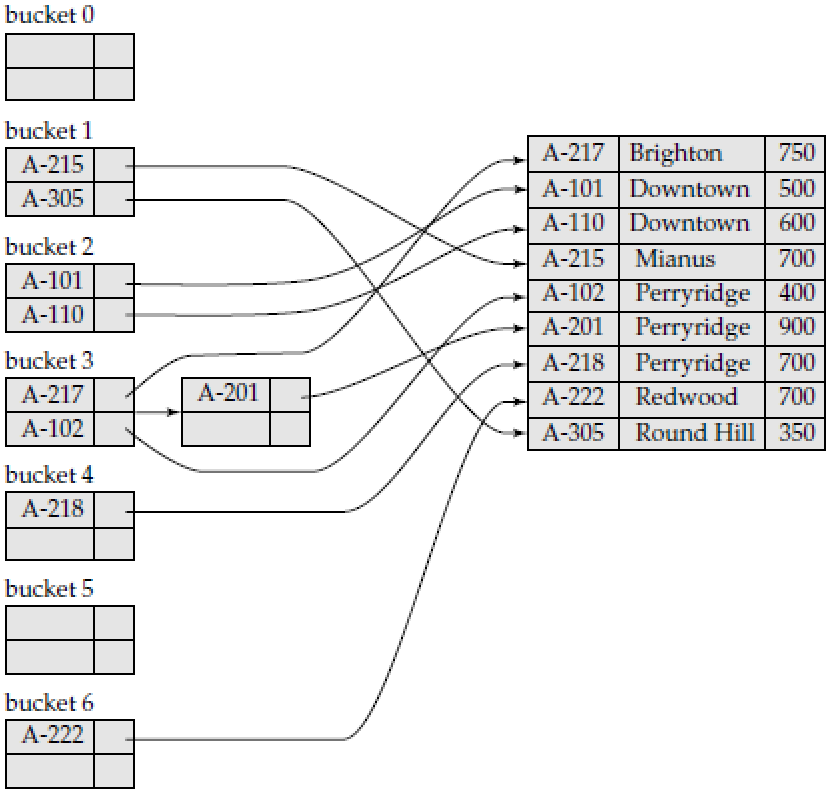
- Additionally, all leaf nodes are linked using link list, which makes B^+ tree to support random access as well as sequential access.
- The structure of B+ tree is shown below.
- Every leaf node is at equal distance from the root node.
- A B^+ tree is of order n where n is fixed for every B^+ tree.



(v). Hash Index:

- It is based on a uniform distribution of values across a range of buckets.
- The bucket to which a value is assigned is determined by a function, called a *hash function*.
- One disadvantage of sequential file organization is that we must access an index structure to locate data, or must use binary search, and that results in more I/O operations.
- File organizations based on the technique of **hashing** allow us to avoid accessing an index structure.
- Hashing also provides a way of constructing indices.
- A **hash index** organizes the search keys, with their associated pointers, into a hash file structure.
- The hash index can be constructed as follows:
  1. We apply a hash function on a search key to identify a bucket, and store the key and its associated pointers in the bucket.
  2. Figure below shows a secondary hash index on the *account* file, for the search key *account-number*.
  3. The hash function in the figure computes the sum of the digits of the account number modulo

4. The hash index has seven buckets, each of size



A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

2. Hash index on search key *account-number* of *account* file.

5. One of the buckets has three keys mapped to it, so it has an overflow bucket.
6. In this example, *account-number* is a primary key for *account*, so each searchkey has only one associated pointer.
7. In general, multiple pointers can be associated with each key.
8. We use the term *hash index* to denote hash file structures as well as secondary hash indices.

(vi). Bitmap index

- A bitmap index is a special kind of index that stores the bulk of its data as bit arrays (bitmaps).
- It answers most of the queries by performing bitwise logical operations on these bitmaps.
- The most commonly used indexes, such as B+trees, are most efficient if the values they index do not repeat or repeat a smaller number of times.
- In contrast, the bitmap index is designed for cases where the values of a variable repeat very frequently.
- For example, the gender field in a customer database usually contains at most three distinct values: male, female or other.
- For such variables, the bitmap index can have a significant performance advantage over the commonly used trees.

Example: A bitmap index may be logically viewed as follows:

identifier	**Has internet**	**Bitmaps**	
		Y	**N**

1	Yes	1	0
2	No	0	1
3	No	0	1
4	unspecified	0	0
5	yes	1	0

- On the left, Identifier refers to the unique number assigned to each resident, HasInternet is the data to be indexed, the content of the bitmap index is shown as two columns under the heading *bitmaps*.
- Each column in the left illustration is a *bitmap* in the bitmap index.
- In this case, there are two such bitmaps, one for "has internet" *Yes* and one for "has internet" *No*.
- It is easy to see that each bit in bitmap *Y* shows whether a particular row refers to a person who has internet access. This is the simplest form of bitmap index.

(vii). Function index:

- In most cases, an index is used to quickly locate the data record(s) from which the required data is read.
- In other words, the index is only used to locate data records in the table and not to return data.
- A function index is a special case where the index itself contains the required data field(s) and can return the data.
- Consider the following table (other fields omitted):

ID	Name	Other Fields
12	Plug	...
13	Lamp	...
14	Fuse	...

- To find the Name for ID 13, an index on (ID) is useful, but the record must still be read to get the Name. However, an index on (ID, Name) contains the required data field and eliminates the need to look up the record.
- A function index can dramatically speed up data retrieval but may itself be large due to the additional keys, which slow down data insertion & update.
- To reduce such index size, some systems allow non-key fields to be included in the index.
- Non-key fields are not themselves part of the index ordering but only included at the leaf level, allowing for a covering index with less overall index size.

Q.7. Explain the following:

(i). Ordered indexing
(ii). Hashing
(iii). Multiple Key access

CT: S-06 6M

Ans.

(i). Ordered Indexing:

- To gain fast random access to records in a file, we can use an index structure.
- Each index structure is associated with a particular search key.

- Just like the index of a book or a library catalog, an ordered index stores the values of the search keys in sorted order, and associates with each search key the records that contain it.
- The records in the indexed file may themselves be stored in some sorted order, just as books in a library are stored according to some attribute.
- A file may have several indices, on different search keys.
- If the file containing the records is sequentially ordered, a **primary index** is an index whose search key also defines the sequential order of the file.
- Primary indices are also called **clustering indices**.
- The search key of a primary index is usually the primary key, although that is not necessarily so.
- Indices whose search key specifies an order different from the sequential order of the file are called **secondary indices**, or **nonclustering** indices.

(ii). Hashing:

- For a huge database structure it is not sometime feasible to search index through all its level and then reach the destination data block to retrieve the desired data.
- Hashing is an effective technique to calculate direct location of data record on the disk without using index structure.
- It uses a function, called hash function and generates address when called with search key as parameters.
- Hash function computes the location of desired data on the disk.

Hash Organization

- **Bucket:** Hash file stores data in bucket format.
- Bucket is considered a unit of storage.
- Bucket typically stores one complete disk block, which in turn can store one or more records.
- **Hash Function:** A hash function h, is a mapping function that maps all set of search-keys K to the address where actual records are placed. It is a function from search keys to bucket addresses.

(iii). Multiple key access:

- For certain type of queries, it is advantageous to use multiple indices if they exist.
- Assume that the account file has two indices: one for branchname and other for balance.
- Consider the following query:Find all account numbers at the perriridge branch with balance = 1000.

■ Example:

```
select account_number
from account
where branch_name = “Perryridge” and  balance = 1000
```

■ The Possible strategies for processing query using indices on single attributes are :

1. Use index on branch_name to find accounts with branch name Perryridge; test balance = 1000
2. Use index on balance to find accounts with balances of $1000; test branch_name = “Perryridge”.
3. Use branch_name index to find pointers to all records pertaining to the Perryridge branch. Similarly use index on balance. Take intersection of both sets of pointers obtained.

- Those pointers that are in the intersection point to records pertaining to both perriridge and accounts with balance of 1000.

- The third strategy is the only one that is of advantage for the existence of multiple indices.

Q. 8. Write short note on the following(any two):
(i). B+tree (ii). Index files (iii). Static Hash Function

CT: W-07 8M

Or

Write short notes on the following (any one):

(i). B+ tree. (ii). Index files

CT: S-012 3M

Ans.

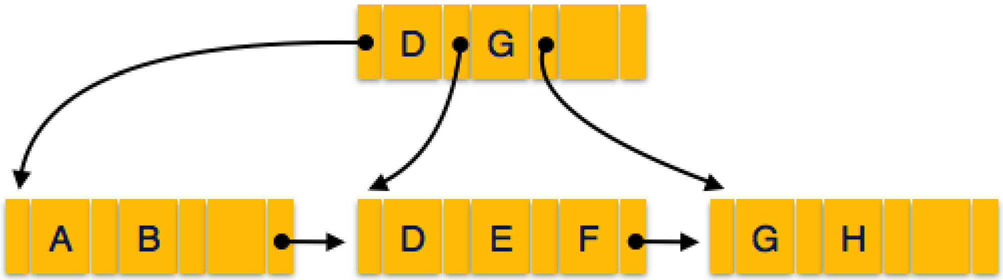
(i). B$^+$ Tree:

- B+ -tree indices are an alternative to indexed-sequential files.
- The disadvantage of indexed-sequential files are as follows:
  1. Performance degrades as file grows, since many overflow blocks get created.
  2. Periodic reorganization of entire file is required.

- **Advantage of B+ -tree index files:**

1. Automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
2. Reorganization of entire file is not required to maintain performance.

- Advantages of B+ -trees outweigh disadvantages of extra insertion and deletion overhead, space overhead
- A B+-tree is a rooted tree satisfying the following properties:
  1. All paths from root to leaf are of the same length
  2. Each node that is not a root or a leaf has between (n/2) and n children.
  3. A leaf node has between □ (n–1)/2□ and n–1 values
  4. Special cases:
- If the root is not a leaf, it has at least 2 children.
- If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n–1) values.
- B+ tree is multi-level index format, which is balanced binary search trees.
- Single level index records becomes large as the database size grows, which also degrades performance.
- All leaf nodes of B$^+$ tree denote actual data pointers.
- B$^+$ tree ensures that all leaf nodes remain at the same height, thus balanced.
- Additionally, all leaf nodes are linked using link list, which makes B$^+$ tree to support random access as well as sequential access.
- Structure OF B$^+$ Tree
  1. Every leaf node is at equal distance from the root node.
  2. A B$^+$ tree is of order n where n is fixed for every B$^+$ tree.



[*Image: B*$^+$ *tree*]

3. Internal nodes:

- Internal (non-leaf) nodes contains at least ⌈n/2⌉ pointers, except the root node.
- At most, internal nodes contain n pointers.

Leaf nodes:

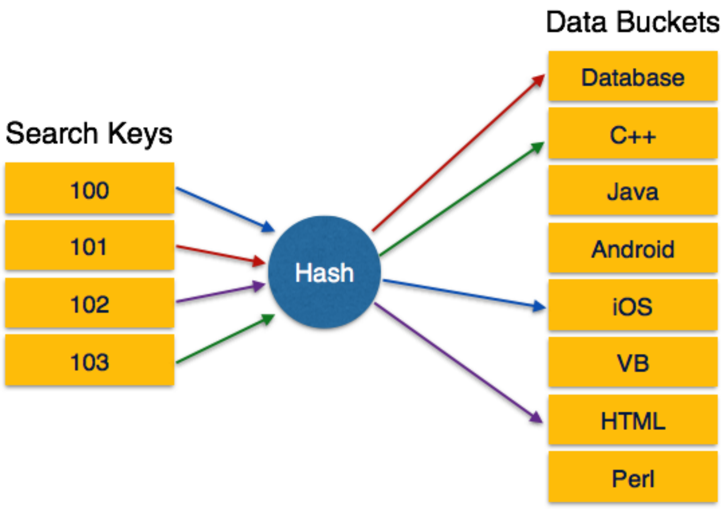
- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values
- At most, leaf nodes contain n record pointers and n key values
- Every leaf node contains one block pointer P to point to next leaf node and forms a linked list.

(ii). Index files:

- Many queries reference only a small proportion of the records in the file.
- Example, To find all accounts at the perriridge branch "references only a fraction of the account records.
- It is efficient for the system to read every record and to check the branchname field for the name "perriridge".
- For quick output, the system should be able to locate these records directly.
- To perform such type of accessing, the index access methods are used.
- An index for a file in DBMS as same as the index in the textbook.
- If we want to learn about a particular topic, we can search the topic in the index at the back of the book, find the page no and read the content on that page no.
- The words in this index are in sorted order, making it easy to find the word and locate the content directly.
- The index is much smaller than the book.
- Similar is the case for the index structure for the file.
- Example: to find the account record given any account no, DBMS will look into the index to find on which disk block the record needed is located and then fetch the disk block to get the account number.
- This technique is used for data blocks where database is of smaller sizes.
- For the database system for larger sizes the sophisticated techniques are used.

(iii). Static Hash function:

- In static hashing, when a search-key value is provided the hash function always computes the same address.
- For example, if mod-4 hash function is used then it shall generate only 5 values.
- The output address shall always be same for that function.
- The numbers of buckets provided remain same at all times.



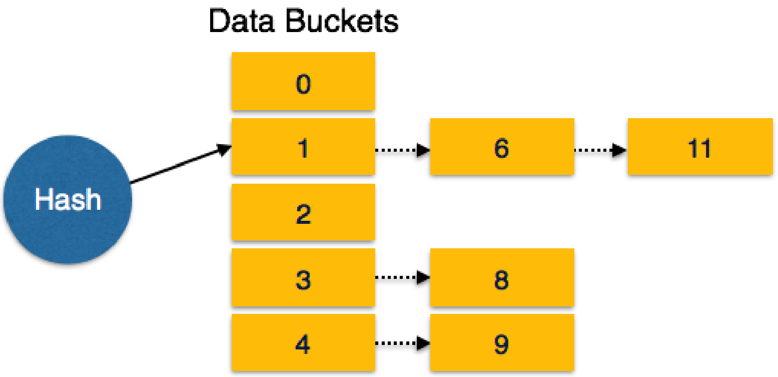
(a). Operation:

- **Insertion:** When a record is required to be entered using static hash, the hash function h, computes the bucket address for search key K, where the record will be stored.
  Bucket address = h(K)
- **Search:** When a record needs to be retrieved the same hash function can be used to retrieve the address of bucket where the data is stored.
- **Delete:** This is simply search followed by deletion operation.

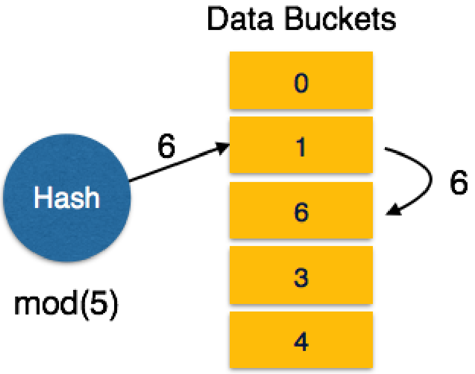
(b). Bucket overflow:

The condition of bucket-overflow is known as collision. This is a fatal state for any static hash function. In this case overflow chaining can be used.

- **Overflow Chaining:** When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called Closed Hashing.



- **Linear Probing:** When hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called Open Hashing.



For a hash function to work efficiently and effectively the following must match:

- Distribution of records should be uniform
- Distribution should be random instead of any ordering

B$^+$ TREE INSERTION

- B$^+$ tree are filled from bottom. And each node is inserted at leaf node.
- **If leaf node overflows:**
  - Split node into two parts
  - Partition at i = $\lfloor (m+1)_{/2} \rfloor$
  - First i entries are stored in one node
  - Rest of the entries (i+1 onwards) are moved to a new node
  - i^{th} key is duplicated in the parent of the leaf
- **If non-leaf node overflows:**
  - Split node into two parts
  - Partition the node at i = $\lceil (m+1)_{/2} \rceil$
  - Entries upto i are kept in one node
  - Rest of the entries are moved to a new node

B$^+$ TREE DELETION

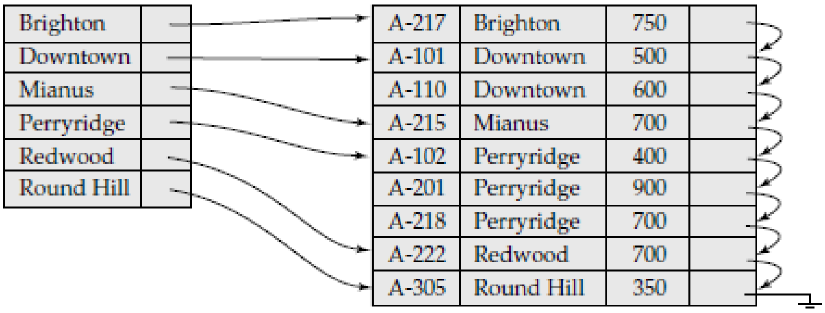
- B$^+$ tree entries are deleted leaf nodes.
- The target entry is searched and deleted.
  - If it is in internal node, delete and replace with the entry from the left position.
- After deletion underflow is tested
  - If underflow occurs
    - Distribute entries from nodes left to it.
  - If distribution from left is not possible
    - Distribute from nodes right to it
  - If distribution from left and right is not possible
    - Merge the node with left and right to it.

Q.9. When is it preferable to use a dense index rather than a sparse index. Explain.

CT: S-10 4M

Ans.

- **Dense index**: An index record appears for every search-key value in the file.
- In a dense primary index, the index record contains the search-key value and a pointer to the first data record with that search-key value.
- The rest of the records with the same search key-value would be stored sequentially after the first record, since, because the index is a primary one, records are sorted on the same search key. Dense index implementations may store a list of pointers to all records with the same search-key value; doing so is not essential for primary indices.

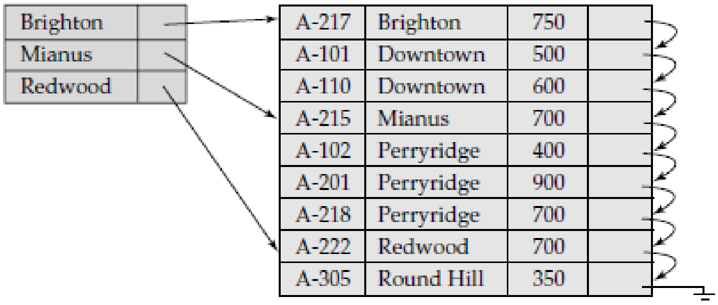


Brighton
Downtown
Mianus
Perryridge
Redwood
Round Hill

A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Dense index.

- **Sparse index**: An index record appears for only some of the search-key values.
- As is true in dense indices, each index record contains a search-key value and a pointer to the first data record with that search-key value.
- To locate a record, we find the index entry with the largest search-key value that is less than or equal to the search-key value for which we are looking.
- We start at the record pointed to by that index entry, and follow the pointers in the file until we find the desired record.



Brighton
Mianus
Redwood

A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Sparse index.

Compared to dense indices, sparse indices require
● Less space and less maintenance overhead for insertions and deletions, But are slower than dense index for locating records.

Q.10. Write short notes on:-

CSE: S-10 6M

(i). index organized table

(ii). Bitmap index

Or

Write a short note on:-

CSE: S-11 6M

(i). Bitmap index

(ii). Dense index

Ans.

(i). Bitmap index

- A bitmap index is a special kind of index that stores the bulk of its data as bit arrays (bitmaps).
- It answers most of the queries by performing bitwise logical operations on these bitmaps.
- The most commonly used indexes, such as B+trees, are most efficient if the values they index do not repeat or repeat a smaller number of times.
- In contrast, the bitmap index is designed for cases where the values of a variable repeat very frequently.
- For example, the gender field in a customer database usually contains at most three distinct values: male, female or other.
- For such variables, the bitmap index can have a significant performance advantage over the commonly used trees.

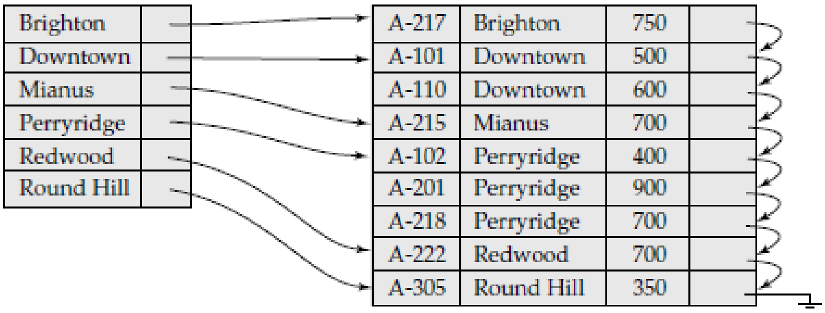
Example: A bitmap index may be logically viewed as follows:

identifier	Has internet	Bitmaps	
		Y	N
1	Yes	1	0
2	No	0	1
3	No	0	1
4	unspecified	0	0
5	yes	1	0

- On the left, Identifier refers to the unique number assigned to each resident, HasInternet is the data to be indexed, the content of the bitmap index is shown as two columns under the heading *bitmaps*.
- Each column in the left illustration is a *bitmap* in the bitmap index.
- In this case, there are two such bitmaps, one for "has internet" *Yes* and one for "has internet" *No*.
- It is easy to see that each bit in bitmap *Y* shows whether a particular row refers to a person who has internet access. This is the simplest form of bitmap index.

(ii) Dense Index

- **Dense index**: An index record appears for every search-key value in the file.
- In a dense primary index, the index record contains the search-key value and a pointer to the first data record with that search-key value.
- The rest of the records with the same search key-value would be stored sequentially after the first record, since, because the index is a primary one, records are sorted on the same search key. Dense index implementations may store a list of pointers to all records with the same search-key value; doing so is not essential for primary indices.



Brighton	
Downtown	
Mianus	
Perryridge	
Redwood	
Round Hill	

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Dense index.

(iii). Index organized table

- Normal relational tables, called heap-organized tables, store rows in any order (unsorted).
- In contrast to this, index-organized tables store rows in a B-tree index structure that is logically sorted in primary key order.
- Unlike normal primary key indexes, which store only the columns included in it definition, IOT indexes store all the columns of the table.
- Properties of the IOT are as under:

- An IOT must contain a primary key.
- Rows are accessed via a logical rowid and not a physical rowid like in heap-organized tables.
- An IOT cannot be in a cluster.
- An IOT cannot contain a column of LONG data type.
- You cannot modify an IOT index property using ALTER INDEX (error ORA-25176), you must use an ALTER TABLE instead.

Advantages:

- As an IOT has the structure of an index and stores all the columns of the row, accesses via primary key conditions are faster .
- As an IOT has the structure of an index and is thus sorted in the order of the primary key, accesses of a range of primary key values are also faster.
- As the index and the table are in the same segment, less storage space is needed.
- In addition, as rows are stored in the primary key order, you can further reduce space with key compression.

- As all indexes on an IOT uses logical rowids, they will not become unusable if the table is reorganized.

Q.11. Differentiate the following :-

(i). Primary & Secondary Index

CSE: W-10 6M

(ii). Indexing & hashing

(iii). B-tree & B+ tree

Ans.

(i). Primary & Secondary index:

Sr.no.	Primary index	Secondary index
1	The primary index contains the key fields of the table and every database must have one primary index.	The Secondary index does not usually contains the key fields of the table and a database can have multiple secondary index.
2	The primary index is automatically created in the database when the table is activated.	Secondary indexes are automatically created when data changes in the table.
3	If a large table is frequently accessed such that it is not possible to apply primary index sorting, you should create secondary indexes for the table.	Secondary indexes uses simple constraints to apply for searching and it can be created whenever primary index cannot be created.
4	It also improves performance but speed is good	Improves performance, but searching speed is low.
5	The primary index contains the key fields of the table and a pointer to the non-key fields of the table.	Additional indexes could be created considering the most frequently accessed dimensions of the table.
6	A primary index is the main index in a table	Secondary index is supplemental to the primary index in a table.
7	It requires unique values to create the index.	It does not require unique values to create the index.
8	It requires less memory space for storing the index structure.	This index requires more memory space as compared to primary index

(ii). B- tree & B+tree

Sr.no	B-tree	B+ tree
1.	In a B tree you can store both keys and data in the internal/leaf nodes	They can store data at the leaf nodes only
2	Here, since data is stored in internal nodes less keys can fit in the page of memory	Because B+ trees don't have data associated with interior nodes, more keys can fit on a page of memory.
3	They store data and key in interior nodes	They cannot store data with the interior nodes
4	It will require more cache misses in order to access data.	It will require fewer cache misses in order to access data that is on a leaf node
5	It would require a traversal of every level in the tree.	The leaf nodes of B+ trees are linked, so doing a full scan of all objects in a tree requires just one linear pass through all the leaf nodes.
6	They does not allow to pack more pointers to other nodes.	The principal advantage of B+ trees over B trees is they allow to pack more pointers to other nodes by removing pointers to data.
7	It increases the depth of the tree.	It increases the fanout and potentially decreases the depth of the tree.
8	B-tree is less efficient & difficult	B+ tree is more efficient, easier & give high performance

(iii). Indexing & Hashing:

Sr.no.	Indexing	Hashing
1	Indexing is a way of sorting a number of records on multiple fields.	Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string
2	Creating an index on a field in a table creates another data structure which holds the field value, and pointer to the record it relates to. This index structure allows Binary Searches to be performed on it.	Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.
3	Indexing guarantee that the distinct values will output the required record.	Hashing do not guarantee that distinct values will hash to distinct address.
4	It does not result in overflow	Hashing results in Overflow.
5	Indexing does not generate collision	Collision is there in Hashing.
6	There is command to define Indexing	There is no command to define hashing.
7	Indexing is linear Searching method	It is an advanced searching method.
8	It is easier to implement	It is difficult to implement.s

Q.12. Give differences between dense index and sparse index with neat diagram.

CT: S-11 3M

Or

Differentiate between Dense Index and Sparse Index.

CT: S-12 3M

Sr. No.	Dense index	Sparse index
1.	An index record appears for every search key value in file.	Index records are created only for some of the records.
2.	Dense indices are indices into a dense matrix	Sparse indices are indices into a sparse matrix
3	A dense matrix is not populated by a default value.	A sparse matrix is a matrix that is populated mostly by a default value, usually zero
4	It works faster than the Sparse Index	It is a bit slower than the dense index
5	Requires more memory space	Requires less memory space
6	Requires maintenance for insertion & deletion.	Require less maintenance for insertion & deletions
7	Index size is large.	Index size is small.

Q.13. Define the following:-

CS: S-12 10M

(i). Data Transfer rate

(ii). Mean Time to failure (MTTF)

(iii). Pinned Block

(iv). Forced output of block

(v). Dirty read

Ans.

(i). Data Transfer rate:

- The speed with which data can be transmitted from one device to another.
- Data rates are often measured in megabits (million bits) or megabytes (million bytes) per second.
- These are usually abbreviated as Mbps and MBps,respectively.
- Another term for data transfer rate is throughput.

(ii). Mean Time to failure (MTTF):

- Mean time to failure (MTTF) is the length of time a device or other product is expected to last in operation.
- MTTF is one of many ways to evaluate the reliability of pieces of hardware or other technology.
- Mean time to failure is extremely similar to another related term, mean time between failures (MTBF).
- The difference between these terms is that while MTBF is used for products than that can be repaired and returned to use, MTTF is used for non-repairable products.
- When MTTF is used as a measure, repair is not an option.
- As a metric, MTTF represents how long a product can reasonably be expected to perform in the field based on specific testing.

(iii). Pinned Block

- For the database to be able to recover from crashes, we need to restrict times when a block maybe written back to disk.
- A block not allowed to be written is said to be **pinned**. Many operating systems do not provide support for pinned blocks, and such a feature is essential if a database is to be ``crash resistant''.
- The buffers in the cache in memory are organized in two lists: the write list and the least recently used (LRU) list.
- The write list holds dirty buffers, which contain data that has been modified but has not yet been written to disk.
- The LRU list holds free buffers, pinned buffers, and dirty buffers that have not yet been moved to the write list.
- Free buffers do not contain any useful data and are available for use.

- Pinned buffers are currently being accessed.

(iv). Forced output of block

- Sometimes it is necessary to write a block back to disk even though its buffer space is not needed. (Called the **forced output** of a block.)
- This is due to the fact that MM contents (and thus the buffer) are lost in a crash, while disk data usually survives.

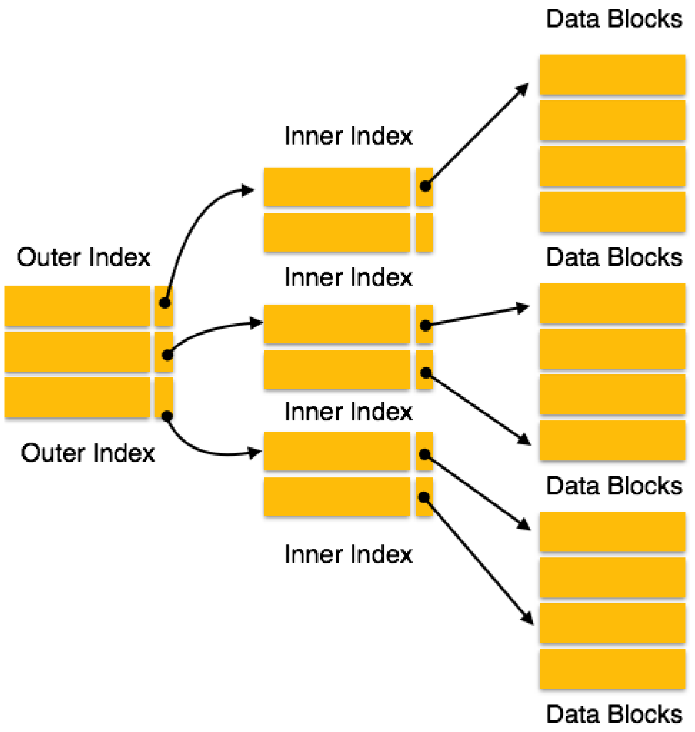
(v). Dirty read :

- There may be situation in database processing, wherein one transaction can change a value, and a second transaction can read this value before the original change has been committed or rolled back.
- This is known as a *dirty read* scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid value.
- While you can easily command a database to disallow dirty reads, this usually degrades the performance of your application due to the increased locking overhead.
- Disallowing dirty reads also leads to decreased system concurrency.

Q.**14. What is multilevel Indices. Explain with proper figure. 5M**

Ans. **Multilevel Index**

- Index records are comprised of search-key value and data pointers.
- This index itself is stored on the disk along with the actual database files.
- As the size of database grows so does the size of indices.
- There is an immense need to keep the index records in the main memory so that the search can speed up.
- If single level index is used then a large size index cannot be kept in memory as whole and this leads to multiple disk accesses.

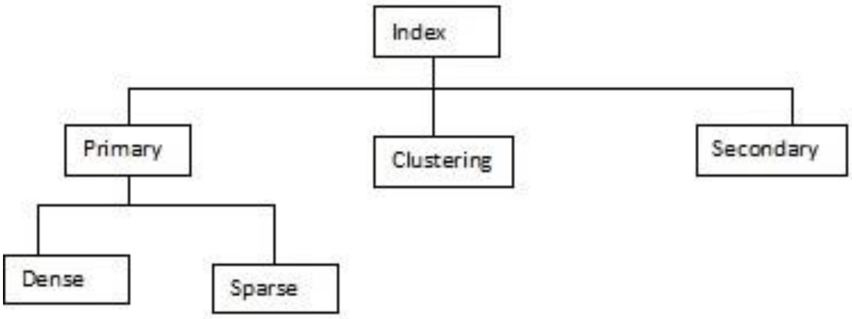


- Multi-level Index helps breaking down the index into several smaller indices in order to make the outer most level so small that it can be saved in single disk block.
- This makes it easily accommodable anywhere in the main memory.
- The idea behind multi-level indexes is to reduce the part of the index to search.
- 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 blocks.
- Such a multi-level index is a form of search tree ; however, insertion and deletion of new index entries is a severe problem because every level of the index is an ordered file.
- Hence most multi-level indexes use B-tree or B+ tree data structures, which leave space in each tree node disk block) to allow for new index entries.
- A multi-level index considers the index file as an ordered file with a distinct entry for each K(i).
- Using multi-level indexes, we can reduce the number of block accesses required to search for a record given its indexing field value.
- Insertions and deletions are still a problem because all the index levels are physically ordered files.
- The multilevel indexes can be used on any type of index, primary, clustering, or secondary, as long as the first level index has distinct values for K(i), and fixed length entries.

Q.15. Explain primary & secondary index in detail. **6M**

Ans.

- An index is a small table having only two columns.
- The first column contains a copy of the primary or candidate key of a table and the second column contains a set of pointers holding the address of the disk block where that particular key value can be found.
- The advantage of using index lies in the fact is that index makes search operation perform very fast.
- Suppose a table has a several rows of data, each row is 20 bytes wide.
- If you want to search for the record number 100, the management system must thoroughly read each and every row and after reading 99x20 = 1980 bytes it will find record number 100.
- If we have an index, the management system starts to search for record number 100 not from the table, but from the index.
- The index, containing only two columns, may be just 4 bytes wide in each of its rows.
- After reading only 99x4 = 396 bytes of data from the index the management system finds an entry for record number 100, reads the address of the disk block where record number 100 is stored and directly points at the record in the physical storage device.
- The result is a much quicker access to the record (a speed advantage of 1980:396).
- The only minor disadvantage of using index is that it takes up a little more space than the main table.
- Additionally, index needs to be updated periodically for insertion or deletion of records in the main table.
- **Types of Index**

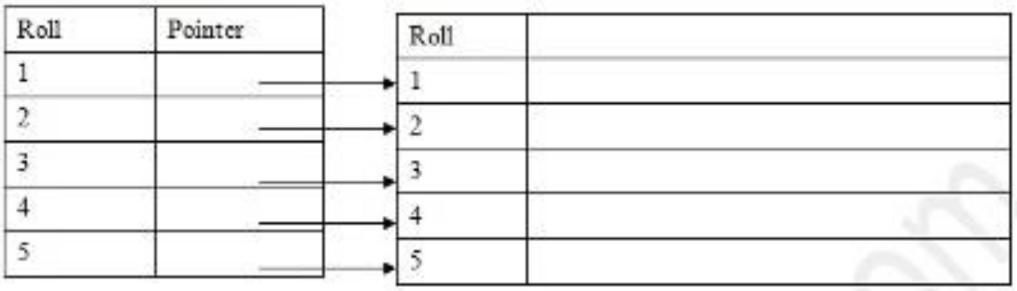


(A). PRIMARY INDEX:

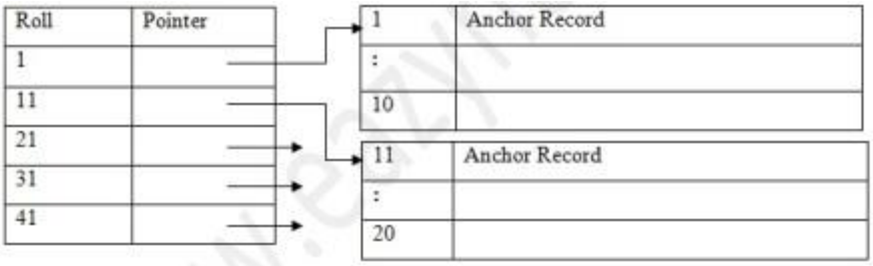
- In primary index, there is a one-to-one relationship between the entries in the index table and the records in the main table.
- Primary index can be of two types:

  1. Dense primary index: the number of entries in the index table is the same as the number of entries in the main table.

- In other words, each and every record in the main table has an entry in the index.



3. Sparse or Non-Dense Primary Index:
    - For large tables the Dense Primary Index itself begins to grow in size.
    - To keep the size of the index smaller, instead of pointing to each and every record in the main table, the index points to the records in the main table in a gap. See the following example.

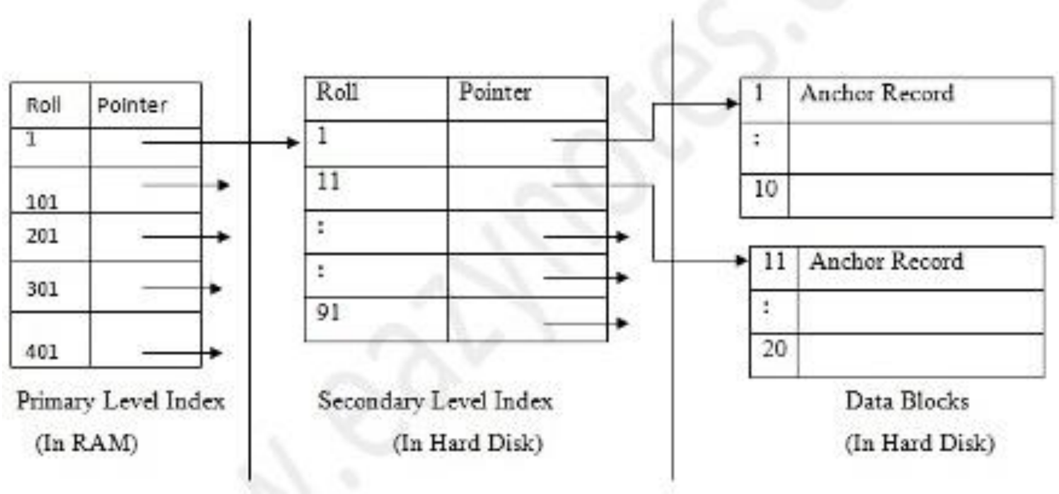


- As you can see, the data blocks have been divided in to several blocks, each containing a fixed number of records (in our case 10).
- The pointer in the index table points to the first record of each data block, which is known as the Anchor Record for its important function.
- If we are searching for roll 14, the index is first searched to find out the highest entry which is smaller than or equal to 14.
- We have 11. The pointer leads us to roll 11 where a short sequential search is made to find out roll 14.

(A). SECONDARY INDEX:

- While creating the index, generally the index table is kept in the primary memory (RAM) and the main table, because of its size is kept in the secondary memory (Hard Disk).
- A table may contain millions of records (like the telephone directory of a large city), for which even a sparse index becomes so large in size that we cannot keep it in the primary memory.
- And if we cannot keep the index in the primary memory, then we lose the advantage of the speed of access.
- For very large table, it is better to organize the index in multiple levels.

- **Example:**



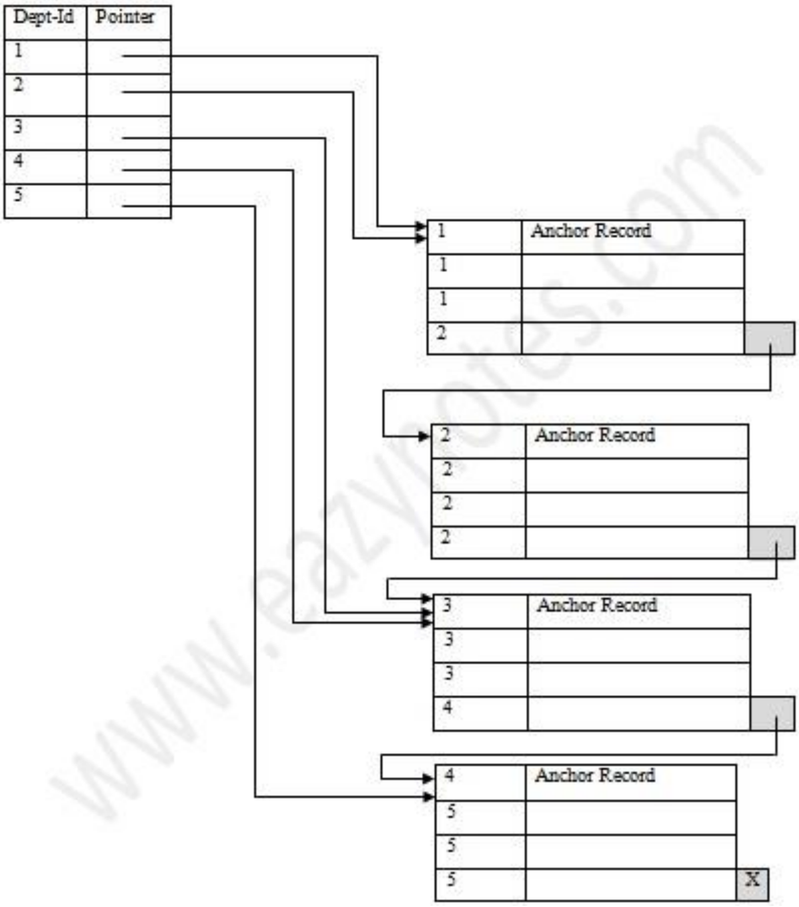
- In this scheme, the primary level index, (created with a gap of 100 records, and thereby smaller in size), is kept in the RAM for quick reference.
- If you need to find out the record of roll 14 now, the index is first searched to find out the highest entry which is smaller than or equal to 14. We have 1.
- The adjoining pointer leads us to the anchor record of the corresponding secondary level index, where another similar search is conducted.
- This finally leads us to the actual data block whose anchor record is roll 11.
- We now come to roll 11 where a short sequential search is made to find out roll 14.

Q.16. Clustering indices may follow faster access to data than is afforded by a non- lustering index. When we must create a non clustering index despite the advantages of a non- clustering index. Explain your answer.

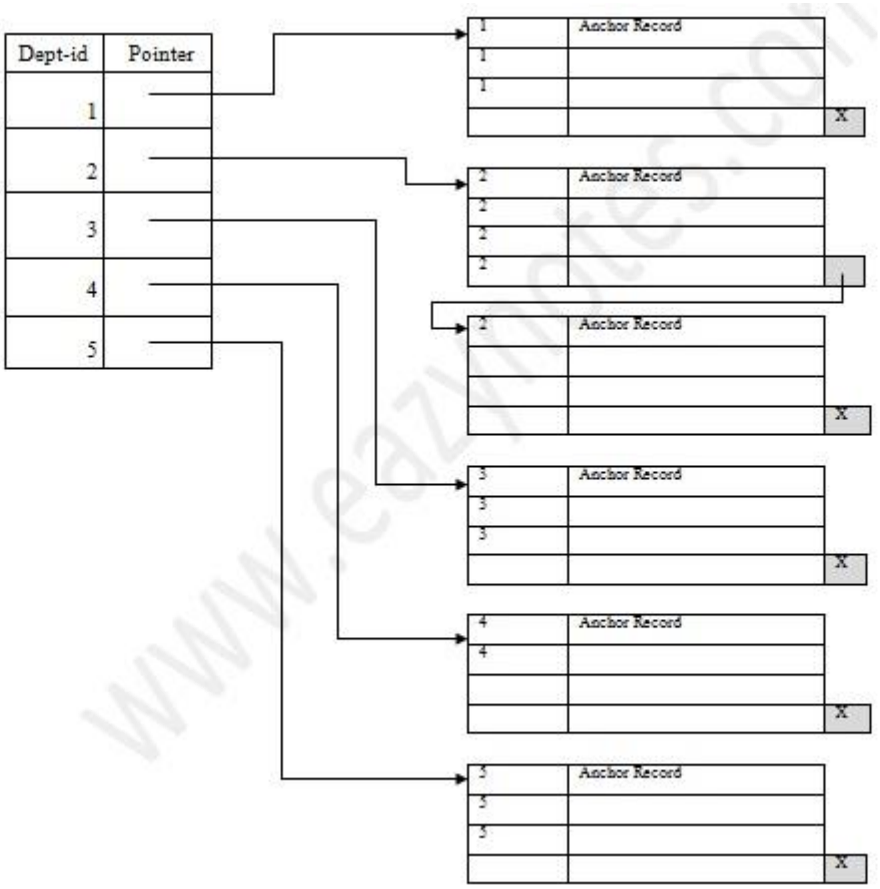
CT: W-06 7M

Ans.

- It may happen sometimes that we are asked to create an index on a non-unique key, such as Dept-id.
- There could be several employees in each department.
- Here we use a clustering index, where all employees belonging to the same Dept-id are considered to be within a single cluster, and the index pointers point to the cluster as a whole.



- Let us explain this diagram. The disk blocks contain a fixed number of records (in this case 4 each).
- The index contains entries for 5 separate departments.
- The pointers of these entries point to the anchor record of the block where the first of the Dept-id in the cluster can be found.
- The blocks themselves may point to the anchor record of the next block in case a cluster overflows a block size.
- This can be done using a special pointer at the end of each block (comparable to the next pointer of the linked list organization).
- The non-clustering index might become a little confusing because one disk block might be shared by records belonging to different cluster.
- A better scheme could be to use separate disk blocks for separate clusters. This is shown in figure below:



- In this scheme, we have used separate disk block for the clusters.
- The pointers, point to the anchor record of the block where the first of the cluster entries would be found.
- The block pointers only come into action when a cluster overflows the block size, as for Dept-id 2.
- This scheme takes more space in the memory and the disk, but the organization in much better and cleaner looking.

HASHING

Q.17. Explain the concept of hashing and its advantages with proper diagram. 6M

Ans.

- One disadvantage of sequential file organization is that we must access an index structure to locate data, or must use binary search, and that results in more I/O operations.
- File organizations based on the technique of **hashing** allow us to avoid accessing an index structure.
- Hashing also provides a way of constructing indices.
- Hashing can be used not only for file organization, but also for index-structure creation.
- A **hash index** organizes the search keys, with their associated pointers, into a hash file structure.
- **Hash Organization**
- **Bucket:** Hash file stores data in bucket format. Bucket is considered a unit of storage. Bucket typically stores one complete disk block, which in turn can store one or more records.
- **Hash Function:** A hash function h, is a mapping function that maps all set of search-keys K to the address where actual records are placed. It is a function from search keys to bucket addresses.
- In a **hash file organization**, we obtain the address of the disk block containing a desired record directly by computing a function on the search-key value of the record.
- A bucket is typically a disk block, but could be chosen to be smaller or larger than a disk block.
- Formally, let K denote the set of all search-key values, and let B denote the set of all bucket addresses.
- A **hash function** h is a function from K to B.
- Let h denote a hash function.
- To insert a record with search key Ki, we compute $h(Ki)$, which gives the address of the bucket for that record.
- Assume for now that there is space in the bucket to store the record.
- Then, the record is stored in that bucket.
- To perform a lookup on a search-key value Ki, we simply compute $h(Ki)$, then search the bucket with that address.
- Suppose that two search keys, $K5$ and $K7$, have the same hash value; that is, $h(K5) = h(K7)$.
- If we perform a lookup on $K5$, the bucket $h(K5)$ contains records with search-key values $K5$ and records with searchkey values $K7$.
- Thus, we have to check the search-key value of every record in the bucket to verify that the record is one that we want.
- Deletion is equally straightforward. If the search-key value of the record to be deleted is Ki, we compute $h(Ki)$, then search the corresponding bucket for that record, and delete the record from the bucket.

Q. 18. Explain the types of Hashing in detail. 6M

Ans.

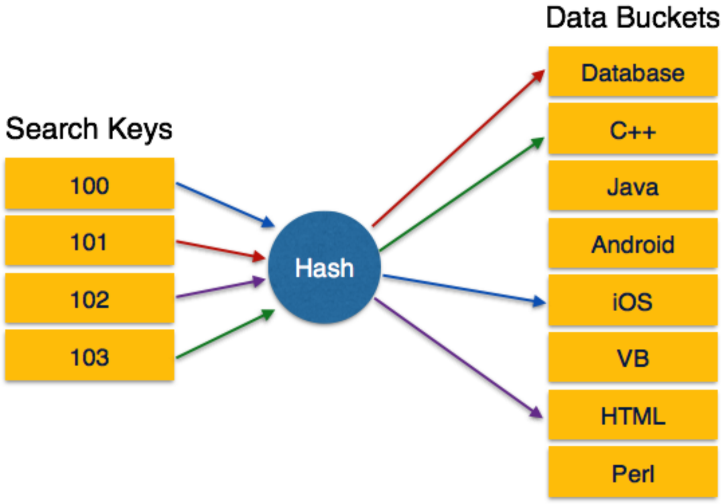
- There are two types of hashing:
  1. Static hashing:

In static hashing, the hash function maps search-key values to a fixed set of locations.

2. Dynamic hashing:
   In dynamic hashing a hash table can grow to handle more items. The associated hash function must change as the table grows

1. **Static Hashing**

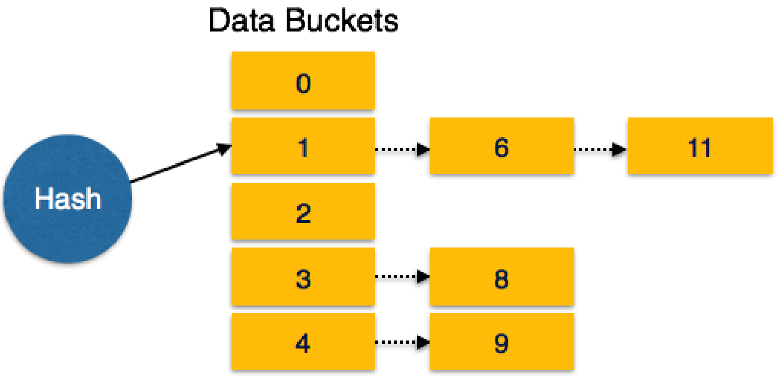
- In static hashing, when a search-key value is provided the hash function always computes the same address.
- For example, if mod-4 hash function is used then it shall generate only 5 values.
- The output address shall always be same for that function.
- The numbers of buckets provided remain same at all times.



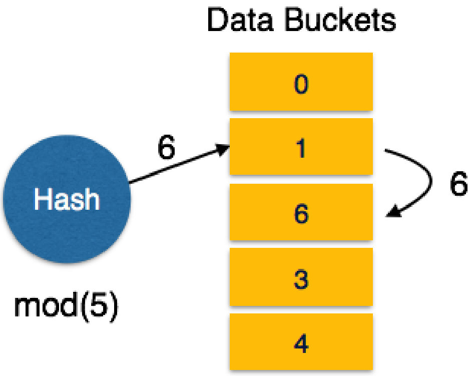
[*Image: Static Hashing*]

Operation:

- **Insertion:** When a record is required to be entered using static hash, the hash function h, computes the bucket address for search key K, where the record will be stored.
  Bucket address = h(K)
- **Search:** When a record needs to be retrieved the same hash function can be used to retrieve the address of bucket where the data is stored.
- **Delete:** This is simply search followed by deletion operation.
  BUCKET OVERFLOW:
  The condition of bucket-overflow is known as collision. This is a fatal state for any static hash function. In this case overflow chaining can be used.
- **Overflow Chaining:** When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called Closed Hashing.



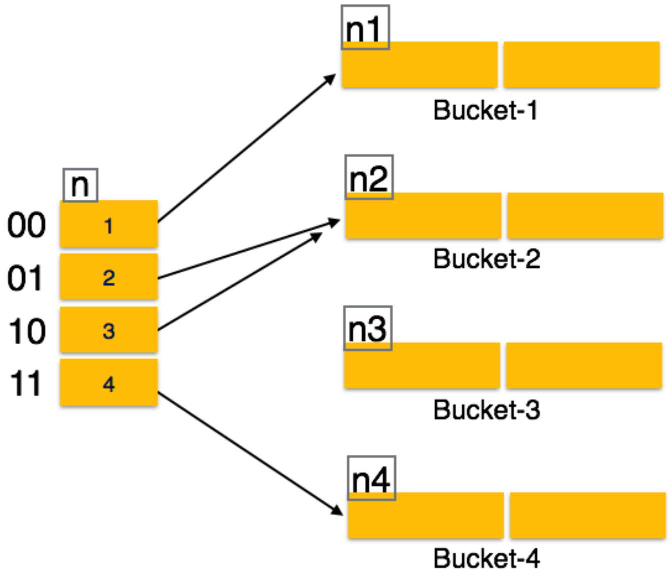
- **Linear Probing:** When hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called Open Hashing.



- For a hash function to work efficiently and effectively the following must match:
- Distribution of records should be uniform
- Distribution should be random instead of any ordering

2. Dynamic Hashing

- Problem with static hashing is that it does not expand or shrink dynamically as the size of database grows or shrinks.
- Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand.
- Dynamic hashing is also known as extended hashing.
- Hash function, in dynamic hashing, is made to produce large number of values and only a few are used initially.



- The prefix of entire hash value is taken as hash index.
- Only a portion of hash value is used for computing bucket addresses.
- Every hash index has a depth value, which tells it how many bits are used for computing hash function.
- These bits are capable to address 2n buckets.
- When all these bits are consumed, that is, all buckets are full, then the depth value is increased linearly and twice the buckets are allocated.

OPERATION

- **Querying:** Look at the depth value of hash index and use those bits to compute the bucket address.
- **Update:** Perform a query as above and update data.
- **Deletion:** Perform a query to locate desired data and delete data.
- **Insertion:** compute the address of bucket
  - If the bucket is already full
    - Add more buckets
    - Add additional bit to hash value
    - Re-compute the hash function
  - Else
    - Add data to the bucket
  - If all buckets are full, perform the remedies of static hashing.

- Advantages and Disadvantages of Hashing:

1. Hashing is not favorable when the data is organized in some ordering and queries require range of data.
2. When data is discrete and random, hash performs the best.
3. Hashing algorithm and implementation have high complexity than indexing.
4. All hash operations are done in constant time.

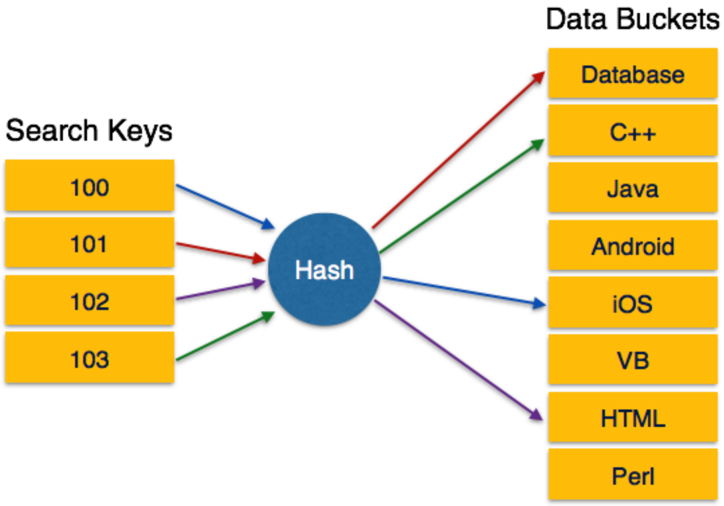
Q.19. Discuss static hashing with suitable example. Also list its drawbacks.

CSE: S-13 8M

Ans.

3. Static Hashing

- In static hashing, when a search-key value is provided the hash function always computes the same address.
- For example, if mod-4 hash function is used then it shall generate only 5 values.
- The output address shall always be same for that function.
- The numbers of buckets provided remain same at all times.



[*Image: Static Hashing*]

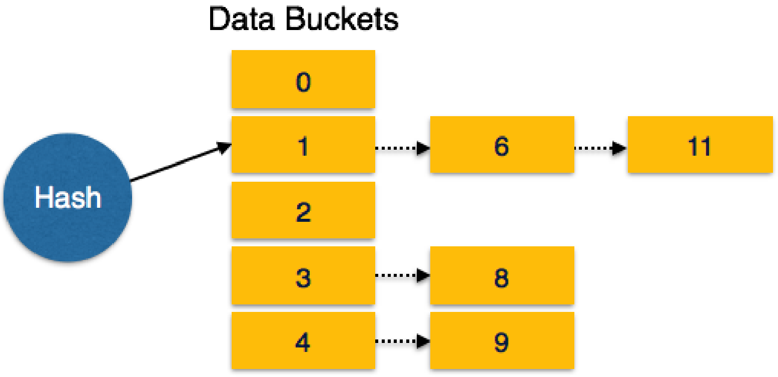
Operation:

- **Insertion:** When a record is required to be entered using static hash, the hash function h, computes the bucket address for search key K, where the record will be stored.
  Bucket address = h(K)
- **Search:** When a record needs to be retrieved the same hash function can be used to retrieve the address of bucket where the data is stored.
- **Delete:** This is simply search followed by deletion operation.

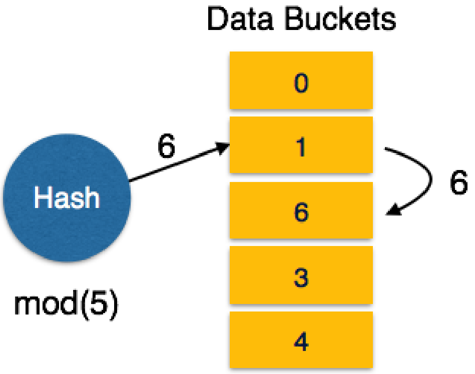
BUCKET OVERFLOW:

The condition of bucket-overflow is known as collision. This is a fatal state for any static hash function. In this case overflow chaining can be used.

- **Overflow Chaining:** When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called Closed Hashing.



- **Linear Probing:** When hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called Open Hashing.



- For a hash function to work efficiently and effectively the following must match:
- Distribution of records should be uniform
- Distribution should be random instead of any ordering

- Drawbacks of Static Hashing:
  1. Long overflow chains can develop and degrade performance.
  2. Most databases grow larger over time.
  3. The Hash functions are static.
  4. The need to fix the set *B* of bucket addresses presents a serious problem with the static hashing technique.

Q.20. Compare hashing and indexing.

CSE: W-11 4M

Ans.

Sr.no.	**Indexing**	**Hashing**
1	Indexing is a way of sorting a number of records on multiple fields.	Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string
2	Creating an index on a field in a table creates another data structure which holds the field value, and pointer to the record it relates to. This index structure allows Binary Searches to be performed on it.	Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.
3	Indexing guarantee that the distinct values	Hashing do not guarantee that distinct values

	will output the required record.	will hash to distinct address.
4	It does not result in overflow	Hashing results in Overflow.
5	Indexing does not generate collision	Collision is there in Hashing.
6	There is command to define Indexing	There is no command to define hashing.
7	Indexing is linear Searching method	It is an advanced searching method.
8	It is easier to implement	It is difficult to implement
9	In cases where ranges are infrequent hashing provides faster insertion, deletion, and lookup then ordered indexing.	In cases where ranges are infrequent, it performs better.
10	Hashing is less efficient if queries to the database include ranges as opposed to specific values	It is more efficient if queries to the database include ranges as opposed to specific values

Q.21. What are the advantages and disadvantages of hash indices relative to B+ tree indices? How might the types of index available influence the choice of a query processing strategy?

CT: S-08 6M

Ans. HASHING

- Hashing provides a means for accessing data without the use of an index structure.
- Data is addressed on disk by computing a function on a search key instead.
- A **hash index** organizes the search keys, with their pointers, into a hash file.
- Hash indices never primary even though they provide direct access.
- Advantages of Hashing over B+ tree:

1. The main advantage of Hashing over B+ tree is the speed of operation. It is more visible when the number of entries are large.
2. In contrast to B+ Tree, The hashing technique is difficult to implement.
3. The cost of a good hash function is much higher than the inner loop of the lookup algorithm for a search tree.
4. If the keys in hashing are not stored, there may be no easy method to enumerate the keys

- Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes.
- Particularly for the **MEMORY** storage engine that lets you choose B-tree or hash indexes.
- **B+-Tree Index Characteristics**
- A B+-tree index can be used for column comparisons in expressions that use the =, >, >=, <, <=, or **BETWEEN**operators.
- The index also can be used for **LIKE** comparisons if the argument to **LIKE** is a constant string that does not start with a wildcard character.
- **Hash Index Characteristics :**
- Hash indexes have somewhat different characteristics from those of B+ tree:

1. They are used only for equality comparisons that use the **=** or **<=>** operators (but are *very* fast).
2. They are not used for comparison operators such as **<** that find a range of values.
3. Systems that rely on this type of single-value lookup are known as “key-value stores.
4. The optimizer cannot use a hash index to speed up `ORDER BY` operations.
5. Only whole keys can be used to search for a row.

Q.22. Explain the distinction between closed and open hashing. Discuss the relative Merits of each technique in database application.

CSE: W-12 5M

Ans.

Sr. no	OPEN HASHING	CLOSED HASHING
1	In open hashing, keys are stored in linked lists attached to cells of a hash table.	In closed hashing, all keys are stored in the hash table itself without the use of linked lists.
2	The "open" in "open addressing" tells us the index (aka. address) at which on object is store in the hash table is not completely determined by its hash code. Instead, the index may vary depending	The "closed" in "closed hashing" refers to the fact that we never leave the hash table; every object is stored directly at an index in the hash table's internal array.

	on what's already in the hash table.	
3	"open" refers to the freedom we get by leaving the hash table, and using a separate list.	None of the objects are actually stored in the hash table's array; instead once an object is hashed, it is stored in a list which is separate from the hash table's internal array.
4	In open hashing set of buckets are fixed.	The set of buckets are not fixed.
5	There are no overflow chains	There are overflow buckets present.
6	If the bucket is full, system inserts records in some other bucket in the initial set of bucket.	If the bucket is full system inserts the new record in the new overflow bucket.
7	Deletion operation is troublesome in open hashing	Deletion is easier in closed hashing.
8	The system does not guarantee solution to overflow buckets.	The system regularly checks for the overflow buckets
9	Overflow of the buckets can be handled easily.	The overflow of the buckets cannot be easily handled.
10	Provides sufficient number of buckets through bucket chaining.	Does not provide sufficient number of buckets.

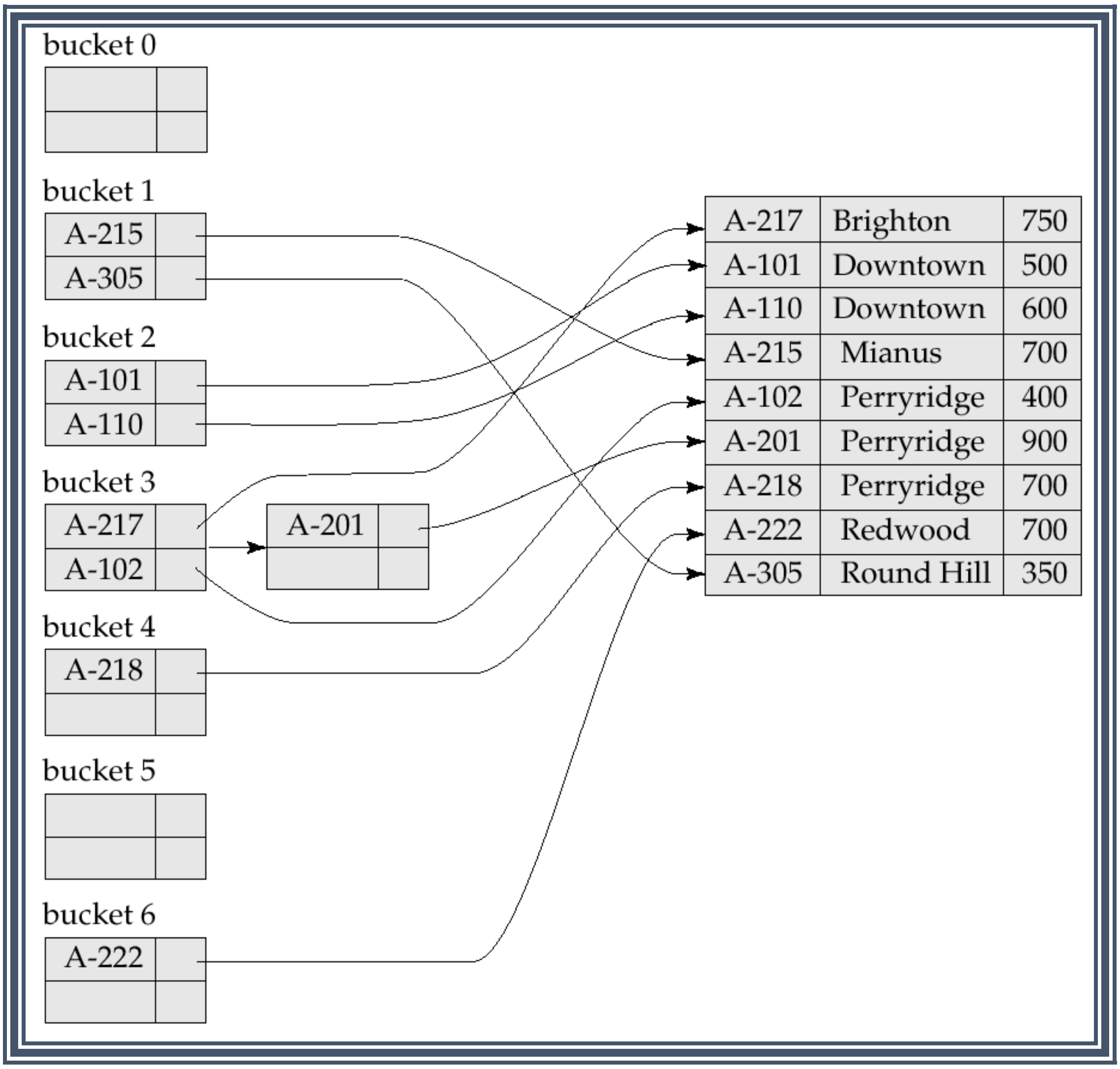
Q.23. Why is the hash structure not the best choice for a search key on which range queries are likely? Explain with an example.

CT: S-08 8M

Ans.

- In hash structure, for performing a lookup on a search key values kj, we simply compute h(ki) i.e. the bucket with that address is searched.
- Suppose two search values k5 and k7, have the same hash value i.e h(k5) = h(k7).
- If we try to perform a lookup on the k5, the bucket h(k5) contains records with the search key value k5 and k7.
- Thus, if there are multiple values in the range then it is necessary to search the key values of every record in the bucket to verify that the record is one that we want hence hash structure is not the best choice for a search key on which range queries are likely.

- A **hash index** organizes the search keys, with their pointers, into a hash file.
- Hash indices never primary even though they provide direct access.
- Hash indices are typically a prefix of the entire hash value.
- More then one consecutive index can point to the same bucket.
- The indices have the same hash prefix which can be shorter then the length of the index.
- Advantage: performance does not decrease as the database size increases
- Space is conserved by adding and removing as necessary
- Hashing is less efficient if queries to the database include ranges as opposed to specific values.
- In cases where ranges are infrequent hashing provides faster insertion, deletion, and lookup then ordered indexing.
- The below example shows the hash index structure.
- **Extendable hashing** splits and coalesces buckets appropriately with the database size.
  i.e. buckets are added and deleted on demand.
- The extensible hashing provides flexibility of data addition & deletion from the index structure.



A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

CONCEPT OF B-TREES

Q.24. Explain the node structure of a B+ tree and B- tree with suitable example.

CT: S-06 6M

Ans.

- B+-tree indices are an alternative to indexed-sequential files.

Disadvantage of indexed-sequential files:

• performance degrades as file grows, since many overflow blocks get created.
• Periodic reorganisation of entire file is required.

Advantage of B+-tree index files:

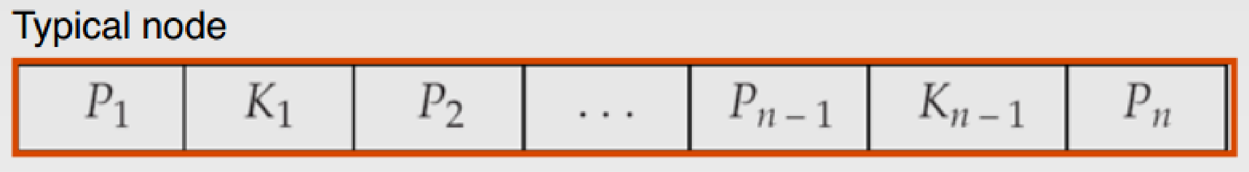
• automatically reorganises itself with small, local, changes, when there are insertions or deletions.
• Reorganisation of entire file is not required to maintain performance.

Disadvantage of B+-trees:

• extra insertion and deletion overhead, space overhead.
• B+-trees are used extensively in database management systems

B+-Tree Node Structure

- Typical node



Typical node

P_1	K_1	P_2	. . .	P_{n-1}	K_{n-1}	P_n

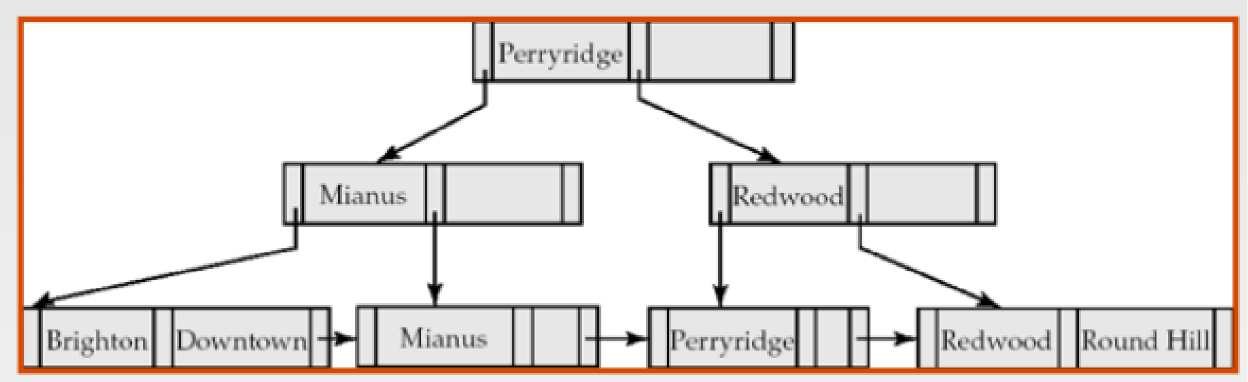
• Ki are the search-key values
• Pi are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered :
K1 < K2 < K3 < . . . < Kn–1
- Usually the size of a node is that of a block

B^+ TREE INSERTION

- B^+ tree are filled from bottom. And each node is inserted at leaf node.
- **If leaf node overflows:**
  - Split node into two parts
  - Partition at i = $\lfloor (m+1)/2 \rfloor$
  - First i entries are stored in one node
  - Rest of the entries (i+1 onwards) are moved to a new node
  - i^{th} key is duplicated in the parent of the leaf
- **If non-leaf node overflows:**

- Split node into two parts
- Partition the node at i = ⌈(m+1)/2⌉
- Entries upto i are kept in one node
- Rest of the entries are moved to a new node

Example:
For n=3



Q.25. What are the disadvantages of B+ tree ? How B+ tree avoids those? Explain with suitable example.

CT: S-06 4M

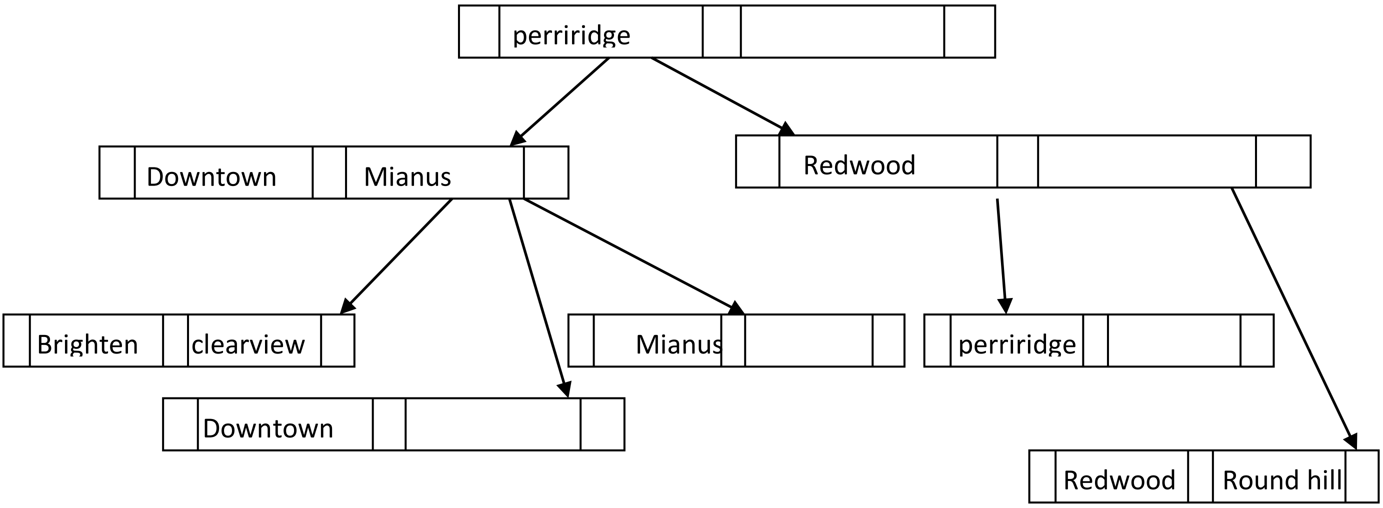
Ans.

- Because of the way B+-Trees store records at the leaf level of the tree, they maximize the branching factor of the internal nodes.
- High branching factor allows for a tree of lower height.
- Lower tree height allows for less disk I/O.
- Less disk I/O means better performance.
- They provide support for range queries in an efficient manner.
- They iterate over an ordered list of elements.
- B-Trees ("B Trees") and B+-Trees ("B Plus Trees") are balanced search trees that perform well on magnetic media and secondary storage because they minimize disk I/O operations.
- B-Tree algorithms are good for accessing pages (or blocks) of stored information which are then copied into main memory for processing.
- It is the branching factor that makes B-Trees so efficient for block storage/retrieval, since a large branching factor greatly reduces the height of the tree

Disadvantages:

1.In B+ tree there is a redundant storage of search key values (i.e. search key values appear more than once).
2. A lookup on the B+ tree requires traversal of a path from the root of the tree to some leaf nodes.

- B-tree eliminates the redundant storage of search keys values .
- For example , in the B+ tree of the figure below, the search keys " Downtown", " Mianus", " Redwood" & "perriridge" appear twice
- Every search key value appears in some leaf node: several are repeated in non-leaf nodes.



- A B-tree allows search key values to appear only once.
- Since search keys are not repeated in the B-tree, index are stored using fewer tree nodes.
- Additional pointer fields can be included as the search keys that appear in non-leaf does not appear anywhere else.

Q.26. Write short note on B+ tree with example. Give its advantages and disadvantages.

CT: S-07 8M

Ans.

- B+ -tree indices are an alternative to indexed-sequential files.
- The disadvantage of indexed-sequential files are as follows:
  1. Performance degrades as file grows, since many overflow blocks get created.
  2. Periodic reorganization of entire file is required.

- Advantage of B+ -tree index files:

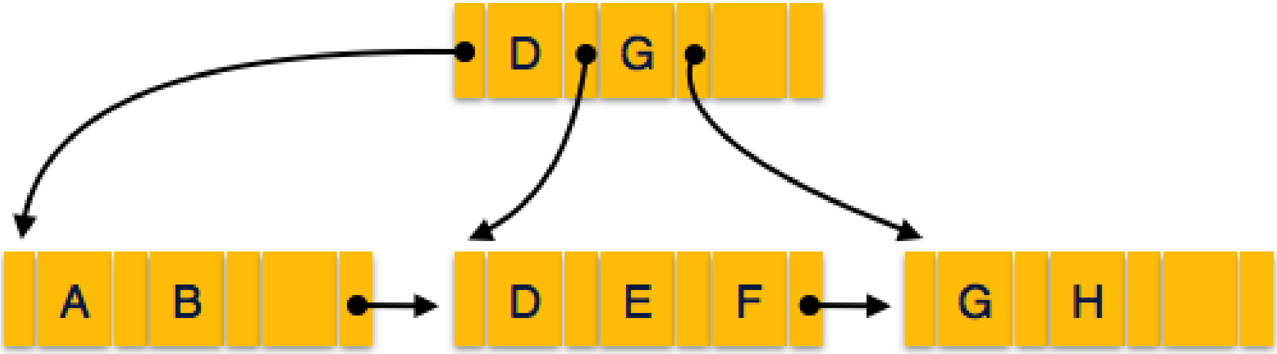
4. Automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
5. Reorganization of entire file is not required to maintain performance.

- Advantages of B+ -trees outweigh disadvantages of extra insertion and deletion overhead, space overhead
- A B+ tree is a rooted tree satisfying the following properties:
  1. All paths from root to leaf are of the same length

2. Each node that is not a root or a leaf has between (n/2) and n children.
3. A leaf node has between □ (n–1)/2□ and n–1 values
4. Special cases:

- If the root is not a leaf, it has at least 2 children.
- If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n–1) values.
- B+ tree is multi-level index format, which is balanced binary search trees.
- Single level index records becomes large as the database size grows, which also degrades performance.
- All leaf nodes of B^+ tree denote actual data pointers.
- B^+ tree ensures that all leaf nodes remain at the same height, thus balanced.
- Additionally, all leaf nodes are linked using link list, which makes B^+ tree to support random access as well as sequential access.
- Structure OF B^+ Tree

4. Every leaf node is at equal distance from the root node.
5. A B^+ tree is of order n where n is fixed for every B^+ tree.



6. **Internal nodes:**

- Internal (non-leaf) nodes contains at least ⌈n/2⌉ pointers, except the root node.
- At most, internal nodes contain n pointers.

Leaf nodes:

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values
- At most, leaf nodes contain n record pointers and n key values
- Every leaf node contains one block pointer P to point to next leaf node and forms a linked list.

Disadvantages:

- Because of the way B+-Trees store records at the leaf level of the tree, they maximize the branching factor of the internal nodes.
- High branching factor allows for a tree of lower height.
- Lower tree height allows for less disk I/O.
- Less disk I/O means better performance.
- They provide support for range queries in an efficient manner.
- They iterate over an ordered list of elements.
- B-Trees ("B Trees") and B+-Trees ("B Plus Trees") are balanced search trees that perform well on magnetic media and secondary storage because they minimize disk I/O operations.

- B-Tree algorithms are good for accessing pages (or blocks) of stored information which are then copied into main memory for processing.
- It is the branching factor that makes B-Trees so efficient for block storage/retrieval, since a large branching factor greatly reduces the height of the tree

Subject I/C

Mr. Jayant Rohankar