I-Unit

Database :- it is a collection of inter-related data which is used to retrive, insert and delete the data efficiently. it is also used to organize the data in the form of a table, Schema, views etc.

Ex:- college Database organizes the data about the admin, staff, students and faculty etc.

Database Management System :-

- it is software which is used to manage the database.

Ex:- MySQL, Oacle etc are very popular commercial database which is used in different applications.

- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, etc.
- it provides protection and security to the database. In case of multiple users, it also maintain data consistency.

Characteristics of DBMS :-

- it can reduce the complex relationship between data.
- it contain ACID properties which maintain data in a healthy state in case of failure.

Advantages of DBMS :-

- Controls database redundancy :- it can control data redundancy because it stores all the data in one single database file and that recorded data is placed in database.
- Data sharing: In DBMS, the authorized users of an organization can share the data among multiple users.

Scanned by CamScanner

- easily maintenance :- it can be easily maintainable due to centralized nature of database system.
- Reduce time :- it reduces development time and maintenance need. the DBMS supports important functions that are common to many applications accessing data in DBMS.
- Concurrent Access and crash Recovery :-
  A DBMS schedules concurrent accesses to the data in such a manner that users can think of data as being accessed by only one user at a time. DBMS Protects users from the effects of system failures.

File Systems Vs DBMS

DBMS	File System
• DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	• File System is a collection of data In this System, the user has to write procedures for manging the database.
• DBMS gives an abstract view of data that hides the details.	• File System provides the detail of the data representation and Storage of data.
• it provides a crash recovery mechanism.	• it doesn't have a crash mechanism.
• DBMS provides a good protection mechanism.	• it is very difficult to protect a file under the file System
• DBMS contains a wide variety of sophisticated techniques to store and retrieve the data	• File System can't efficiently store and retrieve the data.
• DBMS takes care of concurrent acces of data using some form	• In the file System, Concurrent acces has many problems like redirecting the file while other deleting some information or

Scanned by CamScanner

Applications where we use Database Management Systems are:

- Telecom: There is a database to keeps track of the information regarding calls made, network usage, customer details etc. without the database Systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- Industry: distribution centre each one needs a database to keep the records of ins and outs.
  Ex:- distribution centre should keep a track of product units that supplied into centre as well as products that got delivered out from the distribution centre on each day.
- Banking System: For storing customer info, tracking day to day Credit and debit transactions, generating bank statements etc.
- Sales :- To store customer information, production information and invoice details.
- Airlines :- To travel though airlines, we make early reservations, this reservation informations, this reservation information along with flight schedule is stored in database.
- Education Sector :- Database Systems are frequently used in schools and colleges to store and retrive the data regarding Student details, Staff details, course details, exam details, attandance details, fee details etc.
- online shopping :- Amazon, Flipkart etc sites store product information, your address and preferences, credit details and provide you the relevant list of products based on your query.

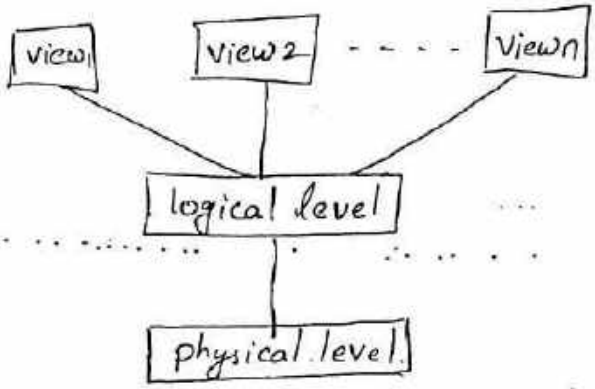
Scanned by CamScanner

View of Data:-

Abstraction is one of main features of database Systems. In three level of DBMS architecture, Top level of that architecture is 'view level'. The view level provides the "view of data" to users & hides the irrelevant details.

1) Data abstraction :-
2) Instance and schema. 3) Datamodel.

1) Data abstraction :-



Three levels of data abstraction.

physical level :- This is lowest level of data abstraction. It describes how data is actually stored in database.

logical level :- This is a middle level of 3-level data abstraction architecture. it discribes what data is stored in database.

view level :- Highest level of data abstraction. This level discribes the user interaction with database System.

- At physical level these records can be described as blocks of storage in memory
- At logical level these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented.
- At view level, user just interact with system with the help of GUI and enter the details at screen, they are not aware of how the data is stored and what data is stored.

Scanned by CamScanner

2) Data models:

It is a collection of conceptual tools for describing data, data relationships, data semantics (instances), and consistency constrains.

The data models are classified in four different categories:

- Relational Model :- it uses a collection of tables to represent both data and Relationships among those data. Each table has multiple columns, each column has a unique name.
  Ex:- recdd based model.
- Entity Relationship model :- ~~ent~~ that consists of collection of basic objects, called entities and Relationships among those objects.
- Object-Based Data model :-
  it is an extending the ER model with notions of encapsulation, methods and object identity.
- Semi structured Data model :-
  This model permits specification of data where individual data items of same type may have different set of attributes.
  above three models are not allowed.
  Ex :- Extensible markup Language (XML)

Refer — 4. Schema & Instance.

Scanned by CamScanner

Database Languages :-

ddL to specify the database schema and DML to express database queries and updates.

DML two types :-

- procedural DML – require a user to specify what data are needed and how to get those data.
- Declarative DML (non procedural DML) – require a user to specify what data are needed without specifying how to get those data.

DDL

it is also used to specify additional properties of data. so ~~this~~ ~~is~~ special type of DDL is called data storage and definition language. These statements define implementation details of database schemas, which are usually hidden from users.

- Domain Contraints :- A domain of possible values must be associated with every attribute (ex:- integer, character, date/time type. Declaring an attribute to be of a particular domain acts as a constraint on the values it can take.
- Referential integrity :-

A value appears in one relation for some set of attributes also appears for a certain set of attributes in another relation. Database modifications causes violations of referential integrity.

- Assertions :- it is any condition that database must always satisfy. ex:- In bank account every one should maintain min bala
- Authorization :- Differentiate the users to access their ~~one~~ permitted on various data values in database.

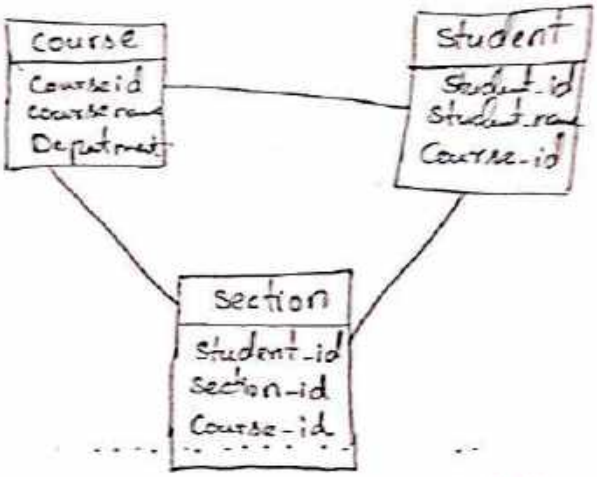
read authorization.
insert authorization etc
update ..

Scanned by CamScanner

2) Instance and schema:

Schema :- Design of database is called the schema
Schema is of three types: physical schema, logical schema and view schema.

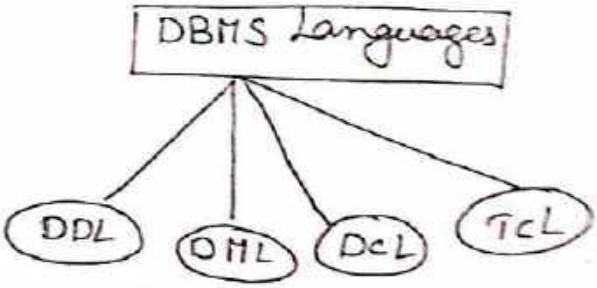
- Schema shows the relationship between three tables. Course, Student and Section.



Instance :- The data stored in database at a particular moment of time is called instance of database.

Database languages :-

Database languages can be used to read, store and update the data in the database.



1) Data Definition language (DDL)

- it is used to define database structure or pattern.
- it is used to create schema, tables, indexes, constraints etc. in database.
- by using DDL statements we can create skeleton of database.

Scanned by CamScanner

DDL statements :-

Create :- it is used to create objects in the database

Alter :- it is used to alter the structure of the database

Drop :- it is used to delete objects from database

Truncate :- it is used to remove all records from a table.

Rename :- it is used to rename an object.

2. ~~Data Manipulation Language :-~~

1) Create Table Students (Sid char(20),
name char(30),
age integer,
gpa Real) ;

2) Alter Table <Table name> Student ADD (column name data-type, --) ;

- Alter table student Drop column age ;
- Alter table student Modify (gpa integer(10)) ;

3) Drop table student ;

4) Truncate Table tablename ;

5) Alter Table student Rename column name to Hname ;

DML (Data Manipulation Language) :

SQL commands that deals with manipulation of data present in database

- Select - is used to retrieve data from the data base :

Select * from student ;

- Insert - is used to insert data into a table

ex:- Insert into student values (1 , Puji, 12 , 87) ;

- Update - is used to update existing data within a table

ex:- Update table-name SET sid = 10, name = PPP where gpa = 87 ;

Scanned by CamScanner

Delete :- is used to delete records from database table

Delete From student where sno=10;

3. DCL (Data Control language):

- It is used to retrieve the stored or saved data.
- Grant : it is used to give user access privileges to database
- Revoke : it is used to take back permissions from the user

4. Transaction control language (TCL)

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Commit : it is used to save the transaction on the database

Rollback : it is used to restore the database to original since the last commit

Savepoint :- sets a savepoint within a transaction.

ex :- Delete from student where age = 20;
Commit;

ex :- Delete from student where age = 20;
RollBack;

ex :- RollBack to Savepoint_Name;

ex :- Savepoint sp1;
//Save point created.
Delete from student where age = 20;
// deleted
Savepoint SP2;
// Savepoint Created.

Scanned by CamScanner

Relational Databases :-

Relational Database is based on relational model and uses a collection of tables to represent both data and relationships among those data.

1) Tables :- Each table has multiple columns, and each column has a unique name. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes.

SNo	SName
1	Anu
2	Ammar
3	Raju.

2) Data manipulation Language :-

The query language of SQL is non procedural. It takes as input several tables and always return a single table.

Ex:- SQL query that finds the names of all customers who reside in Harrison.

```
select customer.customer-name
from customer
where customer.customer-city = 'Harrison';
```

Ex:- Finds the account numbers and corresponding balances of all accounts owned by customer with customer_id 192-83-7465

```
select account.account_number, account.balance
from depositor, account
where depositor.customer_id = '192-83-7465' and
depositor.account_number = account.account_number;
```

Scanned by CamScanner

3) Data-definition Language:

SQL provides DDL that allows to define tables, integrity constraints, assertions etc.

Ex:- defines the account table

Create table account (account_number char(10), balance integer)

4) Database Access from Application Programs.

Application programs are programs that are used to interact with database.

Ex:- In Banking System are programs that generate debit accounts, credit accounts or transfer funds between accounts

- To access the database, DML statements need to be executed from host language. There are two ways:
  - → By providing an application program interface that can be used to send DML and DDL statements to database and retrive the result.
  - → By extending host language syntax to embed DML calls with in the host language program. DML precompiler, converts DML statements to normal procedure calls in host language.

Scanned by CamScanner

Database design:

It is organisation of data according to database model. The designer determines what data must be stored and how the data elements interrelate. With this information, they can begin to fit the data to database model.

1) Design process:

Database designer needs to interact extensively with domain experts. The outcome of this phase is specification of user requirements.

- Next designer choose data model., it ~~conver~~ translates these requirements into a conceptual schema of database. This gives detailed overview of enterprize.
- In conceptual design process involves what attributes we want to capture in database and how to group these attributes to form the various tables.
- Fully develop conceptual schema will also indicate the functional requirements of enterprise. In these requirements users describe the kinds of operations that will be performed on data.

Scanned by CamScanner

2) Database Design for Banking Enterprise

The proces of database for banking enterprise how could be designed.

- Initial specification of user requirements based on interviews & designer's own analysis of enterprise.

⇒ Major characterstics of banking enterprise.

- Bank organized into branches. Each branch is located in particular city and is identified by a unique name.
- Bank customers are identified by their customer-id values, the bank stores each customer's name, Street and city. customer may have accounts & taker loans.
- Bank offers two types of accounts - Saving and checking accounts. customer can have more then one account. each account is assigned a unique account number. Bank maintain a record of each account's balance and most recent date on account operated. etc.
- Bank provides loans to customers. A loan is identified by unique loan number. For each loan, the bank keeps track of loan amount and loan payments. The date and amount are recorded for each payment.

Scanned by CamScanner

. Bank employees are identified by their employ... values. The bank administration stores the name a... telephone number of each employee, names of each employee dependents & joining date etc.

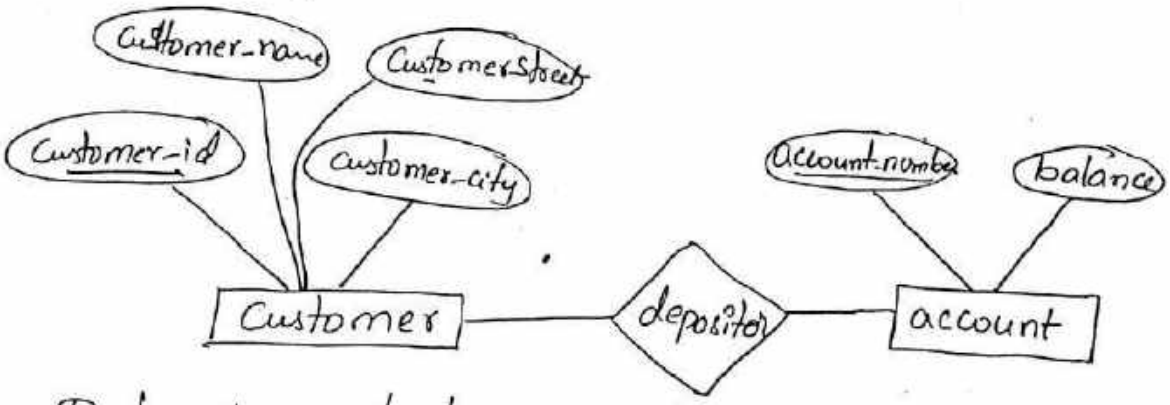
3) entity Relationship model.

. entities are described in database by a set of attributes. ex:- account_number and balance

. Relationship is an association among several entities. ex:- depositor relationship associates a customer with each account that ~~has~~ he/she has.

∴ set of all entities of same type and set of all relationship of same type are termed as an entity set and Relationship set.

E-R diagram components :-



- Rectangle – which represents entity sets
- Ellipse – which represent attributes
- Diamonds – which represent set of relationships among a member from each of several entity sets
- lines – which link attributes to entity sets and entity sets to relationships.

Scanned by CamScanner

- mapping cardinalities – which express the number of entities to which another entity can be associated via a relationship set.

4) normalization :-

The goal of normalization is to store information without unnecessary redundancy.

- Repetition of information
- Inability to represent certain information.

customer-id	acc-number	balance.
001	A-101	500
002	A-201	900
003	A-215	700
004	A-210	400
010	A-217	350
020	A-201	900

- suppose we wish to change account balance of A-201. from $900 to $950. the change reflected in two rows that is against to original design. we will show different balances for ~~different~~ account A-201.
- if any value is unknown then we keep the null values these are occure mainly combing the two or more table information.

Scanned by CamScanner

Data Storage and Querying:

A database System is partitioned into modules that deal with each of responsibilities of overall System. The functional components of a database System can be broadly divided into Storage manager and Query process Components.

Storage Manager:-

- it provides the interface between the low level data Stored in database and application programs and queries Submitted to System.
- it is responsible for interaction with file manager. where the raw data is stored on disk.
- storage manager translates various DML statements into low level file System commands.
- it is responsible for storing, retriving and updating data in database.

Storage Manager Components:-

- Authorization and integrity manager; which tests for satisfaction of integrity constraints and checks the authority of users to access data.
- Transaction manager, which ensures that database remains in a consistent state despite System failures.
- File manager, which manages the allocation of space on disk storage and data structures used to represent information stored on disk.
- Buffer manager; which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory.

Storage manager implements several data structures as part of physical System implementation.

Scanned by CamScanner

- data files, which store database itself
- Data Dictionary, which store metadata about structure of database.
- indices, which can provide fast access to data items.

2) Query processor :-

Query processor components

- DDL interpreter, which interprets DDL statements and records the definitions in data dictionary.
- DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low level instructions that 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 query optimization. i.e it picks the lowest cost evaluation plan from among alternatives.

- Query evaluation engine, which executes low level instructions generated by DML compiler

Transaction management :-

A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency.

- Atomicity means either all actions complete or none occur (abort)
- Consistency, if a certain transaction occurs that attempts to introduce inconsistent data, the entire transaction is rolled back and an error returned to user.
- Durability, The database should be durable enough to hold all its latest updates even if system fails or restarts

Scanned by CamScanner

and commits, then the database will hold the modified data. if a transaction commits but the system fails before the data could be written on to disk, then that data will be updated once the system springs back into action.

- isolation - In a database System where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all transaction will be carried out and executed as if it is only transaction in system. No transaction will affect existence of any other transaction.
- A transaction is very small unit of program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation and Durability. These are known as ACID properties.

Database Architecture :-

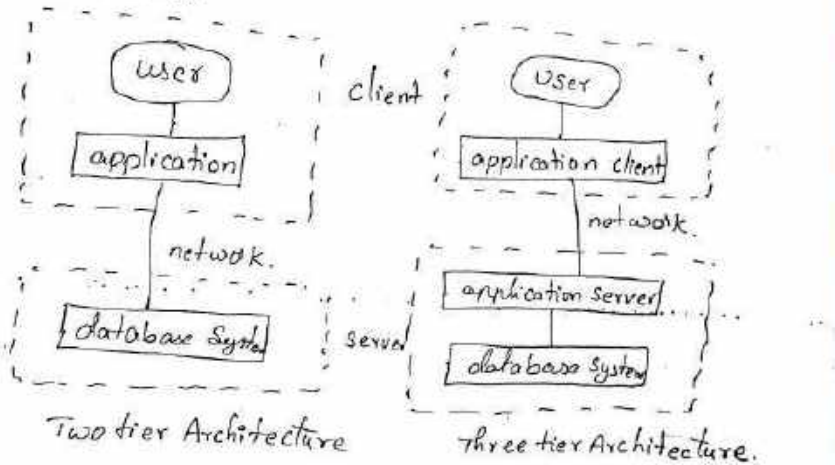
- Database Systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines.
- Now adays database system are not present at side of database system, but connect to it through a network. ∴ client machines, on which remote database users work, and server machines, on which database system runs.
- Database applications are partitioned into two or three parts.

⇒ two tier architecture, the application is partitioned into a component that resides at client machine, which invokes database system functionality at server

Scanned by CamScanner

machine through query language statements.

⇒ three-tier architecture. the client machine acts as front end, client end communicates with an application server, usually through a forms interface. the application server in turn communicates with database system to access data.



Two tier Architecture

Three tier Architecture.

components of DBMS are broadly classified as follows:

1) Query Processor:
   a) DML compiler
   b) Embedded DML pre-compiler
   c) DDL Interpreter
   d) Query evaluation engine.

2) Storage Manager:
   a) Authorization and integrity Manager
   b) Transaction Manager
   c) File Manager
   d) Buffer Manager.

Scanned by CamScanner

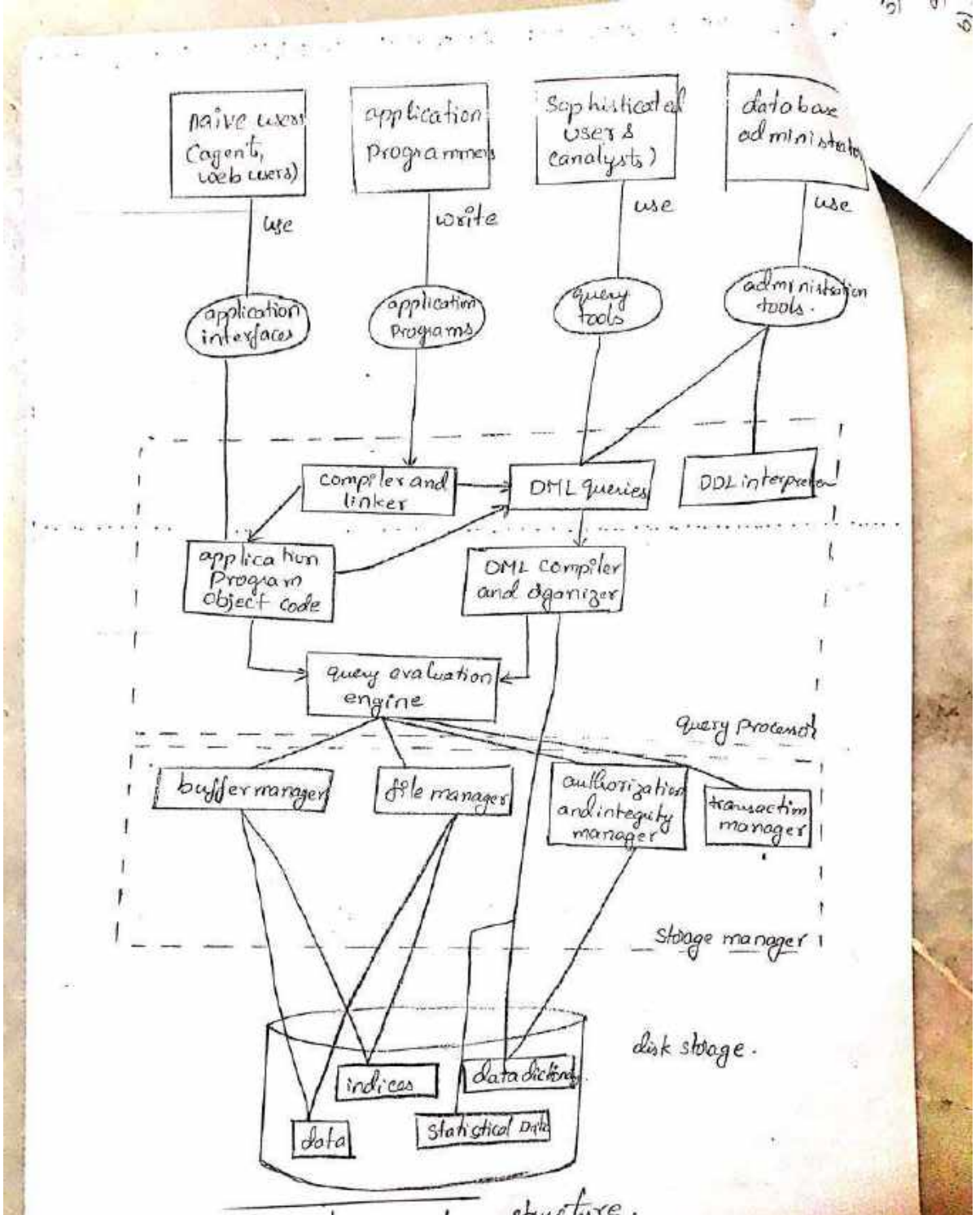


fig: System structure.

Scanned by CamScanner

5) Data Storage:
a) Data Files
b) Data Dictionary
c) Indices
d) Statistical Data.

1) Query processor components:-
• DML pre-compiler: It translates DML statements in a query language into low level instructions that query evalution engine understands.
• Embedded DML pre compiler: it converts DML statements embedded in an application program to normal procedure calls in host language.
• DDL interpreter: it interprets DDL statements and records them in a set of tables containing meta data & data dictionary.
• Query Evaluation Engine: It executes low-level instructions generated by DML compiler

2) Storage Manager components:
they provide interface bln low level data stored in database and applicatim programs and queries Submitted to system.
• Authorization and integrity Manager:
it tests for satisfaction of integrity constraits checks ~~and~~ the authority of users to access data.
• Trans

- Transaction Manager : it ensures that the database remains in a consistent state despite the system failures and that concurrent transaction execution proceeds without conflicting.
- File Manager : It manages the allocation of space on disk storage and data structures used to represent information stored on disk
- Buffer Manager ; It is responsible for fetching data from disk storage into main memory and deciding what data to cache in memory.

3. Disk Storage :-

The following data structures are required as a part of physical system implementation.

- Data files : It stores the database
- Data dictionary : It stores meta data about the structure of database
- Indices : provide fast access to data items that hold particular values.
- Statistical data : it stores statistical information about the data in database.

Scanned by CamScanner

Database users and Administrators

The goal of database system is to retrieve information from and store new information in database.

- Database Users and User interfaces.

There are 4 different types of users, differentiated by the way they expect to interact with the system.

1) Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

* ex- user who which wishes to find her account balance over www. Such user may access a form, where she enters her account number. An application program at web server then retrieves the account balance, using given account number, and passes this information back to the user.

* Naive users fill the appropriate fields of form and reads reports generated by the database.

2) Application programmers: These are computer professionals who write application programs.

* Application programmers can choose from many tools to develop user interfaces.

* RAD (Rapid application development) tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.

Scanned by CamScanner

(Analysts).

3) Sophisticated users interact with the system without writing programs. but communicate with database query language.

* They submit each such query to query processor, whose function is to break down DML statements into instructions that storage manager understands.

4) Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. (ex:- applications are graphics data and audio data) and systems that store data with complex datatypes, knowledge base and expert systems and environment - modeling systems.

Database Administrator

DBMS having Central control of both data and programs that access those data. A person who has such central control over the system is called a database administrator (DBA)

functions of DBA :-

- Schema definition : DBA creates a original database schema by executing a set of data definition statements in DDL.
- storage structure and access method definition.

Scanned by CamScanner

- Schema and physical - organization modification.

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

- Granting of authorization for data access.

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

- Routine maintenance :

Examples:

* periodically backing up database, to prevent loss of data
* Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.

Data mining and ~~Analysis~~ information Retrival.

Text databases consist of huge collection of documents. They collect these information from several sources such as news articals, digital libraries, email messages, web pages etc. Due to increase in amount of information, the text databases are growing rapidly.

Ex:- a document may contain a few structured fields, such as title, author, publishing_date, etc But along with structure data, the document also contains unstructured text

Scanned by CamScanner

components such as abstract and contents without knowing what could be in documents, it is difficult to formulate effective queries for analyzing and extracting useful information from data.

* Users require tools to compare documents and rank their importance and relevance.

∴ text mining has become popular and an essential theme in data mining.

Information Retrieval :-

it deals with the retrieval of information from a Large number of text based documents. Some of the database systems are not usually present in information retrieval Systems because both handle different kinds of data.....

ex:- web search system

- online Document management systems.

* if the user has a long term information need., then the retrieval system can also take an initiative to push any newly arrived information item to user.

* This kind of information processing is called information filtering.

The process of information retrieval involves the following stages

~~Specialty~~ Databases :-

1) Representing collections of documents - how to represent, identify and process the collection of documents
2) User-initiated querying - understanding and processing of queries
3) Retrieval of appropriate documents - the searching Mechanism used to obtain and retrieve the relevant documents.

Scanned by CamScanner

specialty databases:

Several application areas for database systems are limited by restrictions of data model. As a result, researchers have developed several data models deal with these application domains, including object-based data models and semi-structured data models.

1) object Based data models :-

Object oriented programming has become dominant software-development. This led to development of an object-oriented data model that be seen as extending ER model with notations of encapsulation, methods and object identity. Inheritance, object identity, and encapsulation hiding with methods to provide an interface to objects, are among key concepts of object oriented programming that have found applications in modeling.

* object-oriented data model also supports a rich type system, structured and collection types.
* In 1980's, several database systems on object-oriented data model were developed.
* The major database vendors presently support the object-relational data, a data model that combines features of object-oriented data model and data model.

2) semi structured Data model :-

It permit specification of data where individual items of same type may have different sets of attributes.
* This is in contrast data models mentioned earlier, where every data item of a particular must have same set of attributes

Scanned by CamScanner

* XML Language was initially designed as a way of adding markup information text documents, but has become important because of its applications in exchange.

* XML provides a way to represent data that have nested structure, furthermore allows a great deal of flexibility in structuring of data, which is for certain kinds of non traditional data.

History of DataBase

Techniques for data storage and processing have evolved over the years.

- 1950s and early 1960s:

Magnetic tapes were developed for data storage. Processing of data consisted of reading data from one or more tapes and writing data to a new tape.

* Data could also be input from punched card, and output to printers.

ex:- Salary raises were processed by entering the raises on punched cards and reading punched Card deck in synchronization with a tape containing master salary details. the records had to be in same sorted order. the updated tape become as a master tape.

- Late 1960s and 1970's : widespread use of hard disks in late 1960's changed data processing. (i.e hard disks allowed direct access to data) By using hard disk we can access any location in tens of milliseconds.

Scanned by CamScanner

* with disks, network and hierarchical databases could be created that allowed data structures such as lists and trees to be stored on disk.
* Codd defined the relational model and non procedural ways of querying data in relational model and relational databases were born.
* Simplicity of relational model and possibility of hiding implementation details completely from user
* Codd won prestigious Association of Computing Machinery Turing award for his work.

• 1980s: relational model having some disadvantages, relational databases couldn't match the performance of existing network and hierarchical databases.
* after IBM is taken and released first relational database product SQL/DS. Initial commercial relational database systems such as IBM DB2, Oracle, Ingres, and DEC Rdb.
* In network/hierarchical databases, programmers using were forced to deal with many low-level implementation details. they had to keep efficiency in mind when designing their programs.
* In Relation database, almost all these low-level tasks are carried out automatically by database
In 1980's the relational model has become supreme among data models.
* 1980s also saw much research on parallel and distributed databases, and initial work on object-oriented databases.

Scanned by CamScanner

- Early 1990s : SQL language was designed primarily for decision support applications & transaction processing applications, which are update intensive.
  * Decision support and querying re-emerged as a major application area for databases.
  * Many database vendors introduced parallel database products in this period.
  * Database vendors also began to add object-relational support to their databases.
- Late 1990s : Major event was the explosive growth of world wide web (www). Databases were deployed much more extensively than ever before.
  * Database systems now support very high transaction processing rates, very high reliability and 24x7 availability.
- Early 2000s : In early 2000s we have seen emerging of XML and associated Query language XQuery as a new database technology.
  * In this time period we have also witnessed growth in 'autonomic computing / auto admin' techniques for minimizing system administration effort.

Scanned by CamScanner

UNIT - II part - I

RELATIONAL ALGEBRA AND CALCULUS

Preliminaries:

Query :-

The inputs and outputs of a query are relations. A Query is evaluated using instances of each input relation and it produces an instance of the output relation. Queries often involve the computation of intermediate results, which are themselves relation instances. and if we use field names to refer to fields.

The definition of query language constructs must specify the names of fields for all intermediate relation instances, on the other hand, field names make queries more readable.

Sample queries using the following schema:

Sailors (sid : integer, sname : string, rating : integer, age : real)

Boats (bid : integer, bname : string, color : string)

Scanned by CamScanner

Reserves (Sid : integer, bid : integer, day

Relational

The Key fields are underlined, and the dom
of each field is listed after the field name.

sid → key for sailors.

bid → key for boats.

sid, bid, day → key for Reserves.

In this all 3 field value may or may not take as primary key.

values In instance of S_1 of sailors, R_1 of reserves and S_2 of sailors are as follows:

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.0
58	Rusty	10	35.0

(1) Instance S_1 of sailors.

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

(2) Instance S_2 of sailors.

sid	bid	day
22	101	10/10/96
58	103	11/12/96

(3) Instance R_1 of Reserves.

Scanned by CamScanner

Relational Algebra:

Relational algebra is one of the two formerly query languages associated with relational model. Queries in algebra are composed using a collection of operators.

A fundamental property is every operator in the algebra accepts (one or two) relation instances as argument and returns a relation instance as the result.

relational algebra expression is recursively defined to be a relation, unary operator applied to single expression, binary operator applied to two expression.

The operators used in Relational algebra is : (i) Selection (ii) Projection (iii) Cross product (iv) Set-difference.

Scanned by CamScanner

Each relational query describes a step procedure for computing the desired answer based on order in which operators are applied in the query.

Relational algebra is procedural query.

2.1.1 Selection (σ) & Projection (π):

Relational algebra includes operators to Select rows from a relation (σ) and to project column (π).

These operations allows us to manipulate data in a single relation.

Ex:-

Consider the instance of sailor relation of fig (2) denoted as S_2, we can retrieve rows corresponding to expert sailors with rating > 8.

∴ The expression is represented as

$$\sigma_{rating > 8}(S_2).$$

Scanned by CamScanner

sid	sname	rating	age
28	Yuppy	9	35.0
58	Rusty	10	35.0

$\sigma_{rating>8}(S_2)$.

The rating > 8, specifies the selection criteria to be applied while retrieving tuples.

Selection condition includes boolean combination (i.e attributes using connectives $\wedge, \vee$) and attribute op constant or attribute1 op attribute2

op ⇒ Compalision opeation $<, <=, >=, =, \neq, >=$ or

>.

Projection:

The projection opeator π allows us to extract columns from a relation:

Eg:-

We can find out all sailor names and ratings by using π.

The expression is:

$\pi_{sname, rating}(S_2)$

Scanned by CamScanner

If we want to find out only the ... sailors. The expression is:

$\pi_{age}(S_2)$

age
35.0
55.0

(4)

We can compute the names and ratings of highly rated sailors by combining two of the preceding queries.

Expression is:

$\pi_{sname, rating}(\sigma_{rating > 8}(S_2))$

sname	rating
Yuppy	9
Rusty	10

(5) $\pi_{sname, rating}(\sigma_{rating > 8}(S_2))$

2.1.2 Set operations:

The following standard operations on sets are also available in relational algebra:
union (∪), intersection (∩), set-difference (−), and cross product (×).

Scanned by CamScanner

Union: R∪S returns a relation instance containing all tuples that occur in either relation instance R or relation instance S (or both).

R & S must be union-compatible, if the following conditions hold:

- they have the same number of the fields and
- corresponding fields, taken in order from left to right, have the same domains.

Intersection: R∩S returns a relation instance containing all tuples that occur in both R and S. The relation R and S must be union-compatible, and schema of the result is defined to be identical to the schema of R.

Set-difference: R−S returns a relation instance containing all tuples that occur in R but not in S. The relation R & S must be union-compatible. And schema of the result is defined to be identical to the schema of R.

Scanned by CamScanner

Cross-product: R × S returns a relation whose schema contains all the fields of (in the same order as they appear in R) followed by all the fields of S (in the same order as they appear in S).

The cross product operation is sometimes called Cartesian product.

Ex:-

Sid	Sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.0
58	Rusty	10	35.0
28	Yuppy	9	35.0
44	Guppy	5	35.0

S1

Sid	Sname	rating	age
31	Lubber	8	55.5
58	Rusty	10	35.0

S1 × S2

Scanned by CamScanner

sid	sname	rating	age
22	Dustin	7	45.0

S1 - S2.

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubbes	8	55.0	22	101	10/10/96
31	Lubbes	8	55.0	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

S1 × R1

Renaming :-

A relational algebra expression inherits field name from its arguments, however name conflicts can arise in some cases. Ex:- S1×R1 the repeated field names will be seen in relational algebra expression. for this purpose "rename" operator concept has been introduced. It is represented as ρ.

The expression $\rho(R(\bar{F}), E)$ takes an

Scanned by CamScanner

algebra expression E and returns an instance of (new) relation called R.
R contains the same tuples as the result of E and has the same schema as E.
but some fields are renamed.

We can rename as oldname → newname (or)
position → new name.

Ex:- The expression,

$\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2, S1 \times R1)$ returns a relation that contains the tuples.

∴ C(sid1 : integer, sname : string, age : real,
sid2 : integer, bid : integer, day : date).

Scanned by CamScanner

The Join operation is one of the most useful operations in relational algebra and the most commonly used way to combine information from two or more relations.

Although a join can be defined as a cross-product followed by selections and projections.

There are 3 types of Joins.
(1) Conditional Join (2) Equi join (3) Natural Join.

(1) Conditional Join:

The general version of the join operation accepts a join condition C and a pair of relation instances as arguments and returns a relation instance.

The Join condition is identical to a selection condition in the form. The operation is defined as follows.

$$R \bowtie_C S = \sigma_C(R \times S)$$

Scanned by CamScanner

Thus ⋈ is defined to be a cross-product followed by a selection. The condition can refer to attributes of both R and S.

Ex:- $S_1 \bowtie S_1.sid < R_1.sid\ R_1$

(sid)	sname	rating	age	sid	bid	day
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

These values are taken from Ex of Sailors and Reserves table.

② Equi Join:-

A common special case of the join operation R ⋈ S is when the join condition consists of equalities (connected by ∧) of the form R.name1 = S.name2.

Scanned by CamScanner

In this case, obviously there is some redundancy in retaining both attributes in the result.

The join operation with this refinement is called equijoin.

The schema of the result of an equijoin contains the fields of R (with the same names and domains as in R) followed by the fields of S that don't appear in join condition.

Ex:- $S_1 \bowtie R.sid = S.sid\ R_1$

sid	Sname	rating	age	bid	day
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

(3) Natural Join:

A further special case of the join operation $R \bowtie S$ is an equijoin in which equalities are specified on all fields having the same name in R and S. In this case we

Scanned by CamScanner

can simply omit the join condition.
is that the join condition is a collec
equalities on all common fields, we call
this special case a natural join.

The property is result of instance relation
not to have two fields with the same
same name.

The Equijoin represented $S_1 \bowtie R_1$ $S.sid = S.sid$ R_1
is actually a natural join and can simply
be denoted $S1 \bowtie R_1$, since the only common
field is sid.

SQL Join:

A SQL join statement is used to combine
data or rows from two or more tables based
on a common field b/w them.

Different types of joins are:
(i) Inner join (ii) Left join (iii) Right join
(iv) full join.

Scanned by CamScanner

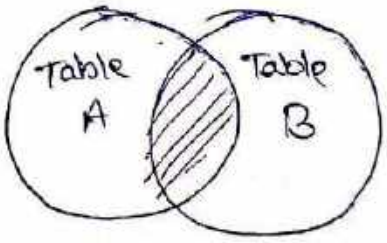
Inner join:-

The Inner join keyword select all rows from both the tables as long as condition satisfies. It combines all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Syntax:

```
Select tablename1.column1, tablename1.column2
,tablename2.column1,..... from table2
Inner join table2 on table1.matching_
column = table2.matching_column;
```

∴ matching_column ⇒ column common to both.



Scanned by CamScanner

⑦ Left-Join:

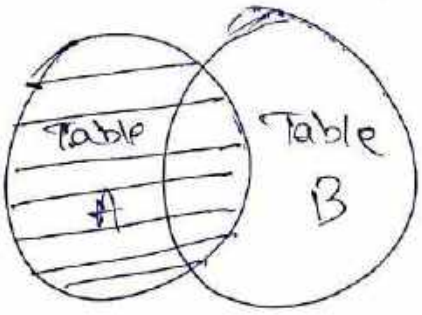
This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join.

The rows for no matching row on right side, the result-set will contain "null".

Left join is also known as Left Outer join.

Syntax:

```
Select table1.column1, table1.column2, table2.column,
..... from table1 Left join table2 on
table1.matching-column = table2.matching-column;
```



⑧ Right Join:

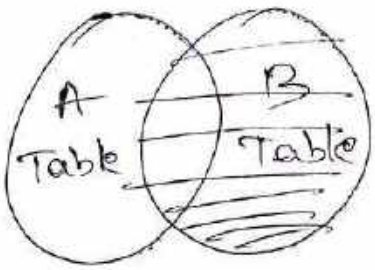
Right Join is similar to left join. This join returns all the rows of the

Scanned by CamScanner

on the right side of join and matching rows for the table on the left side of join. The rows for no matching row on left side, the result set will contain null. Right join is also known as "right outer join"

Syntax:

```
Select table1.column1, table1.column2, table2.column1,
..... from table1 Right join table2 on
table1.matching_column = table2.matching_
                                column
```



(4) Full join (outer join):

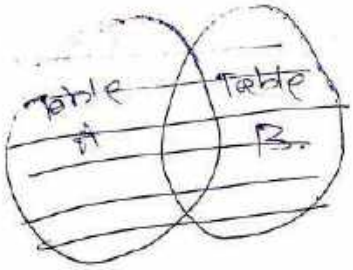
Full join creates the result-set by combining result of both left join & Right join.

The result-set will contain all the rows from both the tables.

Scanned by CamScanner

Syntax:

```
Select table1.column1, table1.column2, table2.column1,
..... from table1 full join table2 on
table1.matching_column = table2.matching_colum.
```



Note:
Queries of Algebra :- Refer in ur notes.

Ex:- Select S.Sid, R.bid from sailors S Natural left outer Join Reserves R;

Relational Calculus :-

Relational calculus is an alternative to relational algebra. In contrast, to the algebra, which is procedural and calculus is non-procedural or declarative. In that it allows us to describe the set of answers without being explicit about how they should be computed.

Scanned by CamScanner

Relational Calculus had a big influence on the design of comercial query languages such as SQL & QBE (Query By Example).

There are 2 categories of Relational Calculus:

(i) Tuple Relational calculus.

(ii) Domain Relational calculus.

The variant of the calculus we present in detail is called the tuple relational Calculus. Variables in TRC take on tuples as values. In another variant called the domain relational calculus. The variable range over field values.

(i) Tuple Relational calculus:

A tuple variable is a variable that takes on tuples of particular relation schema

as values. i.e every value assigned to a variable has the same number & types of fields.

A TRC query has the from $\{T \mid P(T)\}$, where T is a tuple variable and $P(T)$ denotes a formula that describes T.

Ex:-

① Find all Sailors with a rating above 7.

$\{S \mid S \in Sailors \wedge S.rating > 7\}$.

Syntax of TRC Queries:

Let 'Rel' be a relation,
'R' & 'S' be a tuple variables.
'a' be an attribute of R.
and 'b' be an attribute of S.
'op' denote an operator $\{<, >, =, \leq, \geq, \neq\}$

Scanned by CamScanner

* atomic formula is one of the following.

* $R \in Rel$.
* R.a op S.b
* R.a op constant
  (or)
  constant op R.a

* A formula is recursively defined to be one of the following, where p & q themselves formulas, p(R) denotes a formula in which the variable R appears.
* Any atomic formula.
* $\neg p$, $p \wedge q$, $p \vee q$, or $p \Rightarrow q$.
* $\exists R(p(R))$, where R is a tuple variable.
* $\forall R(p(R))$, where R " " " ".

Semantics of TRC Queries:

* F is an atomic formula, $R \in Rel$, & R is assigned a tuple in the instance of relation rel.
* F is a comparison, R.a op S.b
  R.a op constant, or constant op R.a
  R.a and S.b makes the comparison true.

Scanned by CamScanner

* F is of the form $\neg p$ & p is not true or q is true then the form $p \wedge q$ and both p & q are true for $p \vee q$, one of them is true and for $p \Rightarrow q$ q is true whenever p is true.

* F is the form $\exists R(p(R))$, $p(R)$ is true.

* F is the form $\forall R(p(R))$, $p(R)$ is true.

Example Queries of TRC:

(1) Find the names and ages of sailors with a rating above 7.

$\{P \mid \exists S \in Sailors (S.rating > 7 \wedge P.name = S.sname \wedge P.age = S.age)\}$.

(2) Find the Sailor name, boat id, and reservation date for each reservation.

$\{P \mid \exists R \in Reserves \exists S \in Sailors (R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname)\}$

Scanned by CamScanner

Find the names of sailors who have reserved boat 103.

$\{ p \mid \exists s \in Sailors \ \exists R \in Reserves (R.sid = S.sid \wedge R.bid = 103 \wedge p.sname = S.sname) \}$

(4) Find the names of sailors who have reserved a red boat.

$\{ p \mid \exists S \in Sailors \ \exists R \in Reserves (R.sid = S.sid \wedge p.sname = S.sname \wedge \exists B \in Boats (B.bid = R.bid \wedge B.color = 'Red')) \}$

(5) Find the names of sailors who have reserved at least two boats.

$\{ p \mid \exists S \in Sailors \ \exists R1 \in Reserves \ \exists R2 \in Reserves \ S.sid = R1.sid \wedge R1.sid = R2.sid \wedge R1.bid \neq R2.bid \wedge p.sname = S.sname) \}$

(6) Find the names of sailors who have reserved all boats.

$\{ p \mid \exists S \in Sailors \ \forall B \in Boats (\exists R \in Reserves (S.sid = R.sid \wedge R.bid = B.bid \wedge p.sname = S.sname)) \}$

Scanned by CamScanner

④ Find sailors who have reserved all red boats.

$\{S \mid S \in Sailors \wedge \forall B \in Boats (B.color = 'red' \Rightarrow (\exists R \in Reserves (S.sid = R.sid \wedge R.bid = B.bid)))\}$

Domain Relational calculus:

A domain variable is a variable that ranges over the values in the domain of some attribute (e.g.. the variable can be assigned an integer if it appears in an attribute whose domain is the set of integers).

A DRC query has the form.

$$\{\langle x_1, x_2, \ldots x_n \rangle \mid p(\langle x_1, x_2, \ldots x_n \rangle)\}$$

where $x_1, x_2, \ldots x_n \Rightarrow$ are domain variable

Atomic formula of DRC is one of the following:

* $(x_1, x_2, \ldots x_n) \in Rel$, where rel $\Rightarrow$ relation &
each x_i, $1 \leq i \leq n$ is either a var or a constant.

Scanned by CamScanner

X op constant

or

Constant op X

A formula is recursively defined to be one of the following, where p & q themselves formulas & p(x) denotes a formula in which the var X appears.

* Any atomic formula.
* $\neg p$, $p \wedge q$, $p \vee q$ or $p \Rightarrow q$
* $\exists x(p(x))$ where x is a domain var.
* $\forall x(p(x))$, where x is a domain var

Example of DRC Queries:

① Find all sailors with a rating above 7.

$\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{sailors} \wedge T > 7 \}$

② Find the names of sailors who have reserved boat 103.

$\{ \langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in \text{Sailors} \wedge \exists Ir, Br, D (\langle Ir, Br, D \rangle \in \text{Reserves} \wedge Ir = I \wedge$

Scanned by CamScanner

$Br = 103)) \}$

(2) Find the names of sailors who have reserved a red boat.

$\{ \langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in Sailors \wedge \exists (I, Br, D) \in Reserves \wedge \exists (Br, BN, 'red') \in Boats) \}$

(3) Find the names of sailors who have reserved atleast two boats.

$\{ \langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in Sailors \wedge \exists Br1, Br2, D1, D2 (\langle I, Br1, D1 \rangle \in Reserves \wedge (I, Br2, D2) \in Reserves \wedge Br1 \neq Br2)) \}$

(4) Find the names of sailors who have reserved all boats.

$\{ \langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in Sailors \wedge \forall B, BN, C (\neg(\langle B, BN, C \rangle \in Boats) \vee (\exists \langle Ir, Br, D \rangle \in Reserves (I = Ir \wedge Br = B)))) \}$

Scanned by CamScanner

reserves

. ind Sailors who have reserved all red boats.

$\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in Sailors \wedge \forall \langle B, BN, C$

$\in Boats \ (C = 'red' \Rightarrow \exists \langle Ir, Br, D \rangle \in Reserves$

$(I = Ir \wedge Br = B)) \}$

Scanned by CamScanner

Part - II

sql (queries, constraints, triggers

form of a basic sql query.

The syntax of simple sql query and explains its meaning through a conceptual evaluation strategy - DBMS would typically execute a query in different and more efficient way.

Basic form of an sql query is as follows.

```
select [distinct] select_list
    from from_list
    where qualification;
```

Instance S3 of sailors

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.0
95	Bob	3	63.5

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Instance R2 of reserves.

Scanned by CamScanner

* Every query must have a select clause which specifies columns to be retrieved in the answer.

* from clause, which specifies a cross product of tables. where clause is optional, it specifies selection conditions on tables.

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Instance B_1 of boats

Ex:- Find the names and ages of all sailors.

```
select distinct s.sname, s.age from sailors s
```

Ex:- find all sailors with a rating above 7:

```
select s.sid, s.sname, s.rating, s.age
    from sailors as s
    where s.rating > 7.
```

fromlist — list of table names

select-list — list of column names of tables named in from list.

qualification — where clause is a boolean combination of conditions (ie, expression using logical connectives

Scanned by CamScanner

Q.6 Find name of sailors who have reserved boat no. 103.

```
select S.sname from sailors S, Reserves R
where S.sid = R.sid and R.bid = 103;
```

Explanation:- This query on instances R1 of Reserves and S4 of sailors.

sid	bid	day
22	101	10/10/96
58	103	11/12/96

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

→ first step to construct Cross Product S×R3.

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

S×R3

* second step apply qualification S.sid = R.sid
R.bid = 103

sname
rusty

Scanned by CamScanner

Examples

1) find sids of sailors who have reserved

select R.sid
from Boats B, Reserves R
where B.bid = R.bid and B.color = 'red';

2) find colors of boats reserved by Lubber.

select B.color
from sailors S, Reserves R, boats b
where S.sid = R.sid and R.bid = B.bid and
S.sname = 'Lubber';

3) Expressions and strings in select command:-
Each item in a select-list can be of form
expression as column-name, where expression is any
arithmetic or string expression over column names and
constants. It also support use of built-in functions
such as sqrt, sin and mod.

Ex:- compute increments for ratings of persons who have
sailed two different boats on the same day.

select S.sname, S.rating+1 as rating
from sailors S, Reserves R1, Reserves R2
where S.sid = R1.sid and S.sid = R2.sid
and R1.day = R2.day and R1.bid <> R2.bid

Scanned by CamScanner

intersect and except.

SQL provides three set manipulation constructs that extend the basic query.

SQL also provides other set operations: IN, Of ANY, or ALL, and exists.

IN, Exists can be prefixed by NOT.

Ex: find the names of sailors who have reserved a red or a green boat.

```
select s.sname
from sailors s, reserves r, boats b
where s.sid = r.sid and r.bid = b.bid
and (b.color = 'red' OR b.color = 'green');
```

Ex: find the names of sailors who have reserved both a red and a green boat.

```
select s.sname
from sailors s, reserves R1, boats B1, reserves
R2, Boats b2 where s.sid = R1.sid and R1.bid
b1.bid and s.sid = R2.sid and R2.bid = b2.bid
and b1.color = 'red' and b2.color = 'green';
```

(OR)

```
select s.sname
from sailors S, Reserves R, boats b
where s.sid = R.sid and R.bid = b.bid and
b.color = 'red' union
select s2.sname
from sailors s2, boats b2, Reserves R2
where s2.sid = R2.sid and R2.bid = b2.bid and b2.color = 'green';
```

Scanned by CamScanner

Q:- find sids of all sailors who have reserved boats but not green boats

```
select s.sid
from sailors S, Reserves R, boats b
where s.sid = R.sid and R.bid = b.bid and b.color = 'red'
except
select s2.sid
from sailors S2, Reserves R2, boats B2
where S2.sid = R2.sid and R2.bid = b2.bid and
  b2.color = 'green';
```

Ex:- find all sids of sailors who have a rating of 10 & reserved boat 104.

```
select s.sid
  from sailors S
  where s.rating = 10
  union
  select R.sid
  from reserves R
  where R.bid = 104.
```

Scanned by CamScanner

Nested Queries:-

Nested query is a query that has another query embedded within it. the embedded query is called subquery

1) Introduction to nested queries:-

Ex:- find the names of sailors who have reserved boat 103.

```
select s.sname
from sailors s
where s.sid in (select R.sid from Reserves R
                where R.bid=103);
```

Explanation:-

nested subquery computes the (multi) set of sids for sailors who have reserved boat 103 and toplevel query retrieves the names of sailors whose sid is in this set.

* In operator:- allows us to test whether a value is in a given set of elements.

Ex:- find the names of Sailors who have not reserved a red boat

Scanned by CamScanner

```
select s.sname
  from sailors s
 where s.sid not in (select R.sid
                       from Reserves R
                      where R.bid in (select b.bid
                                        from boats b
                                       where b.color='red'));
```

2) Correlated Nested queries:-

In nested queries, the inner subquery has been completely independent of outer query.

Ex:- find the names of sailors who have reserved boat number 103.

```
select s.sname
  from sailors s
 where exists (select *
                 from reserves R
                where R.bid=103
                  and R.sid = s.sid);
```

3) set comparison operators:-

Exists, In and unique with their negated versions sql also supports op any and op all where op = arithmetic operators.

Scanned by CamScanner

```
select s.sname
from sailors s
where s.sid not IN (select R.sid
                    from Reserves R
                    where R.bid in (select b.bid
                                    from boats b
                                    where b.color='red'));
```

2.) Correlated Nested Queries:-

In nested queries, the inner subquery has been completely independent of outer query.

Ex:- find the names of sailors who have reserved boat number 103.

```
select s.sname
from sailors s
where exists (select *
              from reserves R
              where R.bid=103
              and R.sid = s.sid);
```

3) set comparison operators:-

Exists, IN and unique with their negated versions sql also supports op any and op all where op = arithmetic operators.

Scanned by CamScanner

Ex:- find sailors whose rating is better than some sailor called Horatio

```
select s.sid
from sailors s
where s.rating > any (select s2.rating
                      from sailors s2
                      where s2.sname = 'Horatio');
```

Ex:- find sailors whose rating is better than every sailor called Horatio.

Just replace above query with all in where clause of outer query.

Ex:- find the sailors with highest rating

```
select s.sid
from sailors s
where s.rating >= ALL (select s2.rating
                       from sailors s2);
```

Aggregate operators:-

To simply retrieving data, we want to perform some computation or summarization.

SQL supports five aggregate operations.

Scanned by CamScanner

1) count ([Distinct] A) : The number of (unique) values in the A column.

2) SUM ([DISTINCT] A) : The sum of all (unique) values in the A column.

3) AVG ([DISTINCT] A) : The average of all (unique) values in the A column.

4) MAX (A) : The maximum value in A column.

5) MIN (A) : The minimum value in the A column.

Ex:- Find the average age of all sailors.

```
select AVG(S.age)
from sailors s;
```

Ex:- Find the average age of sailors with rating 10.

```
select AVG(S.age)
from sailors S
where S.rating = 10;
```

Ex:- Find the names and ages of the oldest sailor.

```
select S.sname, S.age
from sailors S
```

Scanned by CamScanner

```
where S.age = (select max(S.age)
               from sailors S);
```

* Count the number of sailors.

```
select count(*)
from sailors S;
```

1) Group by and having clauses:-

---

we have applied aggregate operations to all rows in a relation. where no. of groups depends on the relation instance.

-> find the age of the youngest sailor to each rating level.

```
select MIN(S.age)
 from sailors S
 where S.rating = i;
```

where i=1,2,.. 10 writing 10 such queries is tedious. To write such queries, we need a major extension to basic SQL query form, Group by clause.

```
select [DISTINCT] select_list
from from_list
where qualification
Groupby grouping list
Having group_qualification;
```

Scanned by CamScanner

∴ Using the group by clause, we can write above query as follows.

```
select s.rating, MIN(s.age)
from sailors s
Group by s.rating;
```

ex:- find the age of the youngest sailor who is eligible to vote for each rating level with atleast two such sailors.

```
select s.rating, MIN(age) as minage
from sailors s
where s.age >= 18
Group by s.rating
having count(*) > 1;
```

Null values:-

Common values in row are always known but some column values can be unknown.

-> SQL provides a special column value called null.

-> use null when column value is either unknown or inapplicable.

1) Comparisons using null values:-

If the row of field having null values so we can't do comparision with that row of the field.

-> SQL provides a special comparision operator IS NULL

to test whether a column value is null.

2) logical connectives AND, OR, and NOT

ex:- rating = 8 OR age < 40 and rating = 8 and age < 40 and if two arguments evaluates to false if either arguments evaluates to and to unknown if one argument evaluates to unknown and other evaluates to true (or) unknown.

3) Impact on SQL constructs

Arithmetic operations +, −, *, and / all evaluvn null, if one of this arguments is null.

* All aggregate operations simply discard the null values.

4) outer join:-

SQL supports outer joins on doing operations on null values.

ex:- sailors, Reserves two tables, sailors ⋈ Reserves here two table having common column according that condition two tables are combined if one or more records. doesn't having common information

Scanned by CamScanner

then it filled by null values.

* left outer join
* Right outer join
* full outer join.

select s.sid, R.bid from sailors s natural left outer join Reserves R.

Note:- natural - join condition is equality on Common attributes where clause is not required.

sid	bid
22	101
31	null
58	103

fig: left outer join of sailors and reserves

5) Disallowing null values:-

Disallow null values by specifying not null as a part of field definition.

Ex:- Sname Char(20) not null..

Complex integrity constraints

1) Constraints over a single table.

We specify complex constraints over a single table, using table constraints

Check — conditional Expression

Ex:- Create table sailors(sid integer,
sname char(10),
rating integer,
Primary key (sid),
Check (rating >= 1 and rating <= 10));

Ex:- To enforce the constraint that Interlake boats cannot be reserved.

Create table Reserves (sid Integer, bid Integer,
day date,
foreign key (sid) References
sailors, foreign key (bid)
References boats
Constraint no Interlake Res
CHECK ('Interlake' <>
select B.bname
from Boats B

Scanned by CamScanner

where B.bid = Reserves.bid))) ;

2) Domain Constraints and distinct types!-

User can define a new domain using the CREATE domain statement. which uses check constraints.

Create domain rating val Integer dfaut 1
Check (value >= 1 and value <= 10)

following line is the schema declaration.

rating ratingval.

3) Assertions!- Ics over several Tables!-

Table constraints are associated with single table, conditional expression in check clause can refers to other tables. SQL supports the creation of assertion for creating condition involving two (or) more table. (i.e which are constraints not associated with any one table).

Ex!- we wish to enforce the constraint that the number of boats plus the number of sailors should be less than 100.

Scanned by CamScanner

```
Create table sailors(sid integer,
                     sname char(10),
                     rating integer,
                     age real,
                     Primary key (sid),
                     Check(rating>=1 and rating<=10)
                     Check((select count(s.sid) from
                     sailors s)+(select count(b.bid)
                     from boats b)<100) ).
```

Solution suffers from two drawbacks:-

(a) sailors and boats are involves completely symmetric way.

(b) if sailors table is empty then boat table having more than 100 records.

Solution for above query is creating an assertion

```
Create ASSERTION small club.
    Check ((select count(s.sid) from sailors s)
           +(select count(B.bid) from boats B)<100);
```

Scanned by CamScanner

TRIGGERS AND ACTIVE DATABASES

* trigger is a procedure that is automatically invoked by DBMS in response to specified changes to database, and typically specified by DBA.

* A database that has a set of associated triggers is called active database.

* trigger description contains three parts:

Event - A change to database that activates the trigger.

Condition - A query or test that is run when the trigger is activated.

Action - A procedure that is executed when the trigger is activated and its condition is true.

* trigger action can examine the answers to query in condition part of trigger, so the performed as trigger is fixed.

Scanned by CamScanner

```
Create table sailors (sid Integer,
                      sname char(10),
                      rating Integer,
                      age real,
                      Primary key (sid),
                      Check(rating>=1 and rating<=10)
                      Check((select count (s.sid) from
                        sailors s) + (select count (B.bid)
                        from boats b) <100) ).
```

solution suffers from two drawbacks!-

(a) sailors and boats are involves completely symmetric way.

(b) if sailors table is empty then boat table having more than 100 records.

Solution for above query is creating an assertion

```
Create ASSERTION small club.
    Check ((select count(S.sid) from sailors S)
        + (select count (B.bid) from boats B) <100);
```

Scanned by CamScanner

to old and new values of tuples modified. (16)

Syntax:-

```
create trigger [trigger_name]
    [before/after]
    {insert/update/delete}
    on [table_name]
    [for each row]
    [trigger_body]
```

Before/after triggers:-

→ Before trigger runs the trigger action before the triggering statement is run.

→ After trigger runs the trigger action after the triggering statement is run.

→ [for each row]:- This specifies a row-level trigger, ie, the trigger will be executed for each row being affected.

→ [trigger body] This provides the operation to be performed as trigger is fired.

Scanned by CamScanner

Create trigger init count before Insert on students /* event */

Declare
count Integer;
/* Action */
Begin
count = 0
End

Ex:- Create trigger incr_count after Insert on Students
/* event */

when (new.age < 18) /* condition */
for each row
Begin
count = count + 1;
End.

* int_count trigger is executed just once per insert statement, regardless of number of records inserted. because omitted for each row phase. such trigger is called statement level trigger.

Scanned by CamScanner

Designing Active databases:-

An active database is a database that includes an event driven architecture (often in the form of ECA rules) which can respond to conditions both inside and outside the database. Possible uses include security monitoring, alerting, statistics gathering and authorization.

Active database management systems support event monitoring. They store events in event history as an event type and time; the former represents any kind of primitive event, while the latter represents time the event occured.

ADBMS clearly define rule semantics such as event consumption policy, event detection and coupling modes along with instance or set oriented semantics.

A common event consumption policy includes the following parameter contexts:

Scanned by CamScanner

→ Cumulative: All instances of Primitive events are consumed if a complex event occurs.

→ Chronicle:- Events are consumed in time order.

→ Recent:- The latest instances of Primitive Events that are part of complex events are consumed in time order.

Scanned by CamScanner

III unit Schema Refinement AND Normal Forms

Introduction to schema Refinement

The schema refinement is intended to addres and a refinement approach based on decompositions. Redundant storage of information is the root cause of these problems.

i) Problems caused by Redundancy :-

Storing same information redundantly, i.e in more than one place with in a database, can lead to several problems.

- Redundant storage :- Some information is stored repeatedly
- Update Anomalies : If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.
- Insertion Anomalies : It may not be possible to store certain information unless some other, unrelated, information is stored as well.
- Deletion Anomalies : It may not be possible to delete certain information without losing some other, unrelated, information as well

Scanned by CamScanner

2) Decompositions :-

redundancy arises because of functional dependencies. the essential idea is that many problems arising from redundancy can be addressed by replacing a relation with a collection of 'smaller' relations.

ssn	name	lot	rating	hourly_wages	hours_worked
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

An Instance of Hourly_Emps Relation.

hourly_wages attribute is determined by rating attribute this IC is an example of functional dependency. It leads to posible redundancy in relation Hourly_Emps.

- Redundant storage : the rating value 8 corresponds to hourly wage 10, and this association is repeated three times.
- Update Anomalies : hourly_wages in first tuple could be updated without making a similar change in second tuple.
- Insertion Anomalies : we can't insert a tuple for an employee unless we know the hourly wage for employee's rating value.
- Deletion Anomalies : if we delete all tuples with a given rating value we lose the association between that rating value and its hourly_wage value.

Scanned by CamScanner

So we can decompose Hourly-Emps into two relations.
Hourly-Emps2 (ssn, name, lot, rating, hours_worked)
wages (rating, hourly_wages)

ssn	name	lot	rating	hours_worked
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

rating	hourly_wages
8	10
5	7

fig: Instances of Hourly-Emps2 and wages.

we can easily record the hourly wage for any rating simply by adding a tuple to wages, even if no employee with that rating appears in the current instance of Hourly-Emps.

3) Problems Related to Decomposition :-

decomposing a relation schema can create more problems than it solves.

a) Do we need to decompose a relation?

b) what problems does a given decomposition cause?

3a) To help with first question, several normal forms have been proposed for relations. Considering the normal form of a given relation schema can help us to decide whether or not to decompose it further.

With respect to second question, loss less join property enables us to recover any instance of decomposed relation from corresponding instances of smaller relations.

Scanned by CamScanner

- dependency-preservation – any constraint on original relation by simply enforcing some constraints on each of smaller relations.

Functional Dependency :-

Functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute with in a table.

$X \rightarrow Y$

Left side of FD is known as determinant, right side of production is known as a dependent.

for example :-

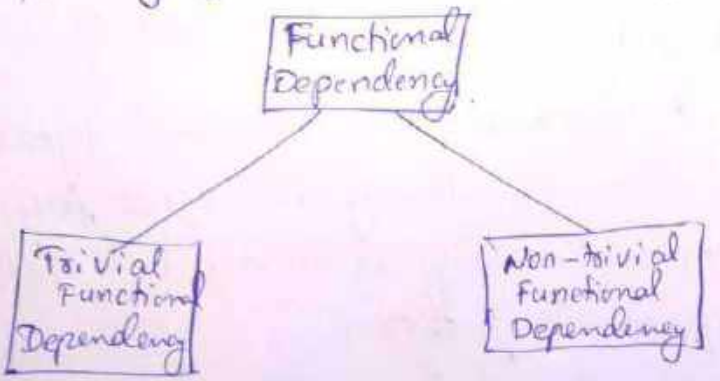
we have an employee table with attributes :

Emp-Id, Emp_name, Emp-Address.

Here Emp-id attribute can uniquely identify the Emp-Name attribute of employee table because if we know Emp-id, then know the employee name & etc

Emp_id → Emp_Name.

- Types of functional dependency.



Scanned by CamScanner

a) Trivial functional dependency

- $A \to B$ has trivial functional dependency if B is a subset of A.

∴ the following dependencies are also trivial like:

$A \to A$, $B \to B$.

- $X \cap Y \neq \emptyset$
- $X \to Y$ then Y is subset of X

example :-

Consider a table with two columns Employee-id & employee-Name.

{Employee-id, Employee-Name} → Employee id is a trivial functional dependency as Employee-id is a subset of {Employee-id, Employee-Name}

2) Non-trival functional dependency

- $A \to B$ has a non-trivial functional dependency if B is not a subset of A.
- when A intersection B is NULL, then $A \to B$ is called as complete non-trivial.

- $X \to Y$ here Y is not subset of X
- $X \cap Y = \emptyset$.

Ex:- ID → Name

Name → DOB

Reasoning ABOUT FDS :-

set of FDs over a relation schema R,

workers (ssn, name, lot, did, Since)

ssn → did, since ssn is the key, and FD did → lot

∴ if two tuples have same values for ssn & did then lot also have the same value.

∴ FD ssn → lot also holds on workers.

i) Closure of a set of FDs:-

The set of all FDs implied by a given set F of FDs is called closure of F, denoted by F^+

Scanned by CamScanner

How to compute closure of a given set F of FDs is using Armstrong's Axioms, can be applied repeatedly to infer all FDs implied by a set F of FDs. we use X, Y, and Z to denote sets of attributes over a relation schema R.

- Reflexivity : If $X \geq Y$, then $X \rightarrow Y$
- Augmentation : If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Transitivity : If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

* Armstrong's Axioms are sound, they generate only FDs in F^+ when applied to a set F of FDs they are also complete, for repeated applications of these rules will generate all FDs in closure F^+

- They are some additional rules, derivable from Armstrong's Axioms.
  - Union : if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - Decomposition : if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Ex:- Suppose we are given a relation scheme $R = (A, B, C, G, H, I)$ and set of functional dependencies :

$A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H$.

Scanned by CamScanner

* additional relations in the external schema can be defined using view mechanism. it provides the support for logical data independence in relational model.

Applying Axioms we can derive the following.

i) $A \rightarrow H$

take $A \rightarrow B$

~~$CG \rightarrow H$~~

$B \rightarrow H$ ~~& $CG \rightarrow H$~~ by transitive decomposition rule

$\therefore A \rightarrow B$, $B \rightarrow H \Rightarrow A \rightarrow H$

by transitivity rule.

ii) $CG \rightarrow HI$

take $CG \rightarrow H$

$CG \rightarrow I$

$\therefore CG \rightarrow HI$ by union rule.

2) Attribute closure :-

The set of attributes that are functionally dependent on attribute A is called Attribute closure of A and it can be represented as A^+

Steps to Find Attribute closure of A.

- Add A to S
- Recursively add attributes which can be functionally determined from attributes of set S until done.

Scanned by CamScanner

Given table

R(E-ID, E-Name, E-city, E_state)

FDS = {E-ID → E-Name, EID → E-CITY, E-ID → E-STATE, E-CITY → E_STATE}

Steps

1) Add E-ID to set {E-ID}
2) Add Attributes which can be derived from any attribute of set. In this case E-Name and E-city, E-state can be derived from E-ID.

$(E\text{-}ID)^+ = \{E\text{-}ID, E\text{-}Name, E\text{-}CITY, E\text{-}STATE\}$

$(E\text{-}Name)^+ = \{E\text{-}Name\}$

$(E\text{-}city)^+ = \{E\text{-}city, E\text{-}State\}$

Q:- Find the attribute closures of given FDs

R(A B C D E) = {AB → C, B → D, C → E, D → A}

To Find $(B)^+$.

closure	FD used
{B}	B → D
{B, D}	D → A
{B, D, A}	AB → C
{B, D, A, C}	C → E
{B, D, A, C, E}	

Scanned by CamScanner

Normal FORMS

Data base Normalization is a technique of organising the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, update and Deletion Anamolies.

- update anomalies ← if data items are scattered and are not linked to each other properly, then it could lead to strange situations. For ex:- when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- Deletion anomalies – we tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- Insert anomalies – we tried to insert data in a record that does not exist at all.

1) First Normal form :-
It is defined in definition of relations itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

course	content
Programming	Java, C++
web	HTML, PHP, ASP

we rearrage the relation as below, to convert it to First Normal form.

Scanned by CamScanner

course	content
Programming	Java
Programming	C++
web	HTML
web	PHP
web	ASP

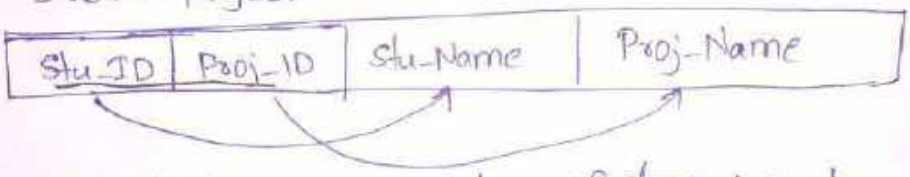
Each attribute must contain only a single value from its predefined domain

Second Normal form.

- Prime attribute – An attribute, which is a part of the candidate-key,
- Non prime attribute – An attribute, which is not a part of the prime-key.

In second Normal form, every non prime attribute should be fully functionally dependent on prime key attribute

Student_Project



Student-project relation that prime key attributes are Stu-ID and proj-ID According to the rule, non-key attributes, ie. Stu_Name and proj_Name must be dependent upon both and not on any of prime key attribute. But we find that Stu_Name can be identified by stu-ID and proj_Name can be identified by Proj_ID independently, this is called Partial dependency, which is not allowed in second Normal Form.

Scanned by CamScanner

Student

Stu_ID	Stu_Name	Proj_ID

Project

Proj_ID	Proj_Name

Third Normal Form :-

it must be in second Normal form and following must satisfy

- No non-prime attribute is transitively dependent on prime key attribute
- For any non-trivial functional dependency, $X \rightarrow A$, then either
  - X is a superkey or
  - A is prime attribute

Student_Detail

Stu_ID	Stu_Name	city	ZIP

Stu_ID is key and only prime key attribute. we find that city can be identified by Stu_ID as well as ZIP itself. Stu_ID → ZIP → city there exist transitive dependency

- To bring this relation into third normal form break the relation into two relations.

Student_Detail

StuD	Stu_Name	ZIP

ZIP codes

ZIP	city

Scanned by CamScanner

Boyce-codd Normal Form :- (BCNF)

Boyce-codd Normal Form is an extension of third Normal Form on strict terms.

BCNF states that -

- For any non-trivial functional dependency $X \rightarrow A$, X must be a Super-key.

In above example, Stu_ID is the Superkey in Relation StudentDetail and zip is the Superkey in Relation zipcodes.

- Stu-ID → stu-Name, zip

zip → city

which confirms that both relations are in BCNF.

Properties of Decompositions :-

Decomposition is a tool that allows to eliminate redundancy.

1) Lossless-join Decomposition :-

Let R be relation schema and let F be a set of FDs over R. A decomposition of R into two Schemas with attribute sets x and y is said to be lossless-join decomposition with respect to F.

$\Pi_{X}(r) \bowtie \Pi_{Y}(r) = r$

S	P	D
S_1	P_1	d_1
S_2	P_2	d_2
S_3	P_1	d_3

instance(r)

S	P
S_1	P_1
S_2	P_2
S_3	P_1

$\Pi_{SP}(r)$

P	D
P_1	d_1
P_2	d_2
P_1	d_3

$\Pi_{PD}(r)$

S	P	D
S_1	P_1	d_1
S_2	P_2	d_2
S_3	P_1	d_3
S_1	P_1	d_3
S_3	P_1	d_1

$\Pi_{SP}(r) \bowtie \Pi_{PD}(r)$

(i.e) lossy decomposition.

Scanned by CamScanner

- when a relation in relational model is not in appropriate normal form then decomposition of a relation is required.
- If relation has no proper decomposition, then it may lead to problems like loss of information
- Decomposition is used to eliminate some of problems of bad design like anomalies, inconsistencies and redundancy.

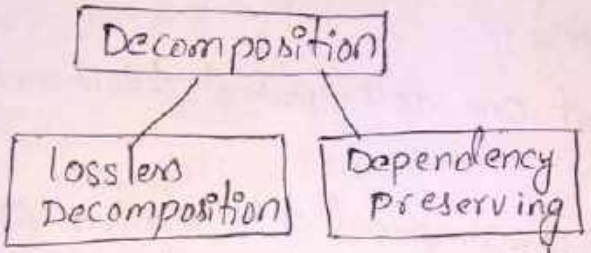


fig :- types of Decomposition

lossless Decomposition :-

Ex:-

EmpID	Empname	Empage	Empcity	Dept ID	Dept name
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE

EMPID	EmpName	EmpAge	empcity
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	stephan	30	Bangalore
52	katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT

DeptID	EmpID	Deptname
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Scanned by CamScanner

Now, when these two relations are joined on common column EMPID, then resultant relation is EMPLOYEE ⋈ Department.

EMPID	EMPName	EMPage	EMPCity	DeptID	DeptName
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stepham	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing.

Dependency preserving :-

- In this, at least one decomposed table must satisfy every dependency.
- if relation R is decomposed into relation R1 and R2, then dependencies of R either must be a part of R_1 & R_2 or must be derivable from the combination of functional dependencies of R1 and R2

Ex:- R(A,B,C,D) with functional dependency set (A → BC).
the relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A→BC is part of relation R1(ABC)

Normalization :-

It is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data. it divides larger tables to smaller tables and links them using relationships. converting the relations into BCNF or 3NF.

Scanned by CamScanner

2NF:-

INF, every non prime attribute is dependent on fully functionally dependent on key attributes

→ How to check if partial dependency is there then table is not in 2NF.

→ condition :- LHS having proper subset of key attributes

AND

RHS - non prime attributes

ex:- FD ⇒ { C → F, E → A, EC → D, A → B }

finding candidate key by using closer property.

$(EC)^+ = ECD$

$= ECDA$

$= ECDAF$

$= ECDAFB$

Candidate key (EC)

C → F

LHS RHS

proper subset AND nonprime

T AND T → T

PFD

E → A

LHS RHS

Proper subset AND non prime.

So partial dependency

EC → A

LHS RHS

F AND T — F.

no partial dependency.

A → B

non prime AND non prime

F AND T → F

no partial dependency

Scanned by CamScanner

3NF :- according to EF codd Rules.
1NF & 2NF; there should be no transitive dependency in table.

Rollno	state	city
1	AP	Guntur
2	Tel	Hyd
3	AP	Guntur
4	Tel	Hyd
5	Tamil nadu	A

FD= Rollno → state
State → city
∴ Rollno → city
From transitive Rule.

Primary key column — Rollno
non prime attributes — state & city.

∴ Transitive property

ex:- R [A B C D]
AB → C , C → D
finding candidate keys.
$(AB)^+$ = ABC
ABCD
Prime attributes - {A,B}
non prime attributes - {C,D}

→ Checking transitive Rule :-

ex: LHS - candidate key OR RHS prime (or) Super key.

Scanned by CamScanner

Ex:- R (ABCD) FD: AB → CD
D → A.

$(AB)^+ = ABCD$

$(DB)^+ = DB.$
$= ADB$
$= ABCD$

Candidate key (AB, DB)

Prime attributes = {A, B, D}

non prime attributes = {C}

checking Trinsivity Rule :-

AB → CD

LHS OR RHS

T OR = T

no transitive

D → A

not candidate or prime

F or T → T

∴ no transitive.

∴ it is in 3NF.

Scanned by CamScanner

BCNF :-

Boyce codd Normal form is a higher version of Third Normal form.

- 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.
- in other words :-
  - i) R must be in 3rd Normal form
  - ii) For each functional dependency ($X \rightarrow Y$), X should be a super key or candidate key.

ex :- Student enrollment for the course

Enrollment

Sid	course	Teacher
1	C	Ankita
1	Java	Poonam
2	C	Ankita
3	C++	Supriya
4	C	Archana

- One student can enrolled for multiple courses
  - ex :- Sid = 1 can enroll for C as well as Java
- For each course, a teacher is assigned to Student
- there are multiple teachers teaching one course.
  - ex :- course C can be taught by both teachers namely - Ankita and Archana.

Scanned by CamScanner

- Candidate key for above table is (Sid, Course)
- Functional dependency.
  (Sid, course) → Teacher
  Teacher → course.
  the above Table is not in BCNF. because dependency teacher → course. Teacher is not Super key & candidate key.
- To convert above table to BCNF we must decompose above table into. student and course tables.

Student

Sid	Teacher
1	Ankita
1	Poonam
2	Ankita
3	Supriya
4	Archana

Sid → Teacher

Courses

Teacher	course.
Ankita	C
Poonam	Java
Supriya	C++
Archana	C

Teacher → course

∴ Tables in BCNF

Ex:- R(ABCDEF) check highest Normal form?

FD: {AB→C, C→DE, E→F, F→A}.

Ans:- Step 1: find all candidate keys.

$(AB)^+ \rightarrow ABCDEF$

$(FB)^+ \rightarrow ABCDEF$

$(EB)^+ \rightarrow ABCDEF$

$(CB)^+ \rightarrow ABCDEF$

C→DE
C→D, C→E
decomposition Rule.

Scanned by CamScanner

∴ Candidate keys – {AB, FB, EB, CB}.

~~Prime attributes – {A, B, C, E, F}~~

Step 2 :- write all prime attributes

PA = {A, B, C, E, F}

Step 3 :- write all non prime attributes

NPA = {D}

	AB → C	C → DE	E → F	F → A
BCNF	✓	×	×	×
3NF	✓	×	✓	✓
2NF	✓	×	✓	✓
1NF	✓	✓	✓	✓

SO :- 1NF.

Ex :- R(ABCDEF), FD: AB → C, C → D, C → E
E → F, F → A.

Ck = {AB, FB, EB, CB}. Prime {A, B, C, E, F}
non prime {D} Reduce the Redundancy upto 0%.

Ans :- R(ABCDEF)

Scanned by CamScanner

2NF

AB → C

F AND F → F — FD

C → D

T && T → T — PFD

C → E

T && F → F — FD

E → F

T && F → F → FD

F → A

T and F → F → FD

Not in 2NF because C → D.

R(ABCDEF)

R1(ABCEF)
AB → C, C → E
E → F, F → A

(CD) R2
C → D
Ck = {C}

∴ Relation in 2NF.

Ck = {AB, EB, BC, FB}

3NF :- LHS OR RHS
ck Prime.

AB → C ✓ C → D ✓

C → E ✓

E → F ✓

F → A ✓

∴ 3NF

Scanned by CamScanner

BCNF :- LHS must be CK.

$AB \to C$ ✓

$C \to E$ ✗

$E \to F$ ✗ NOT in BCNF.

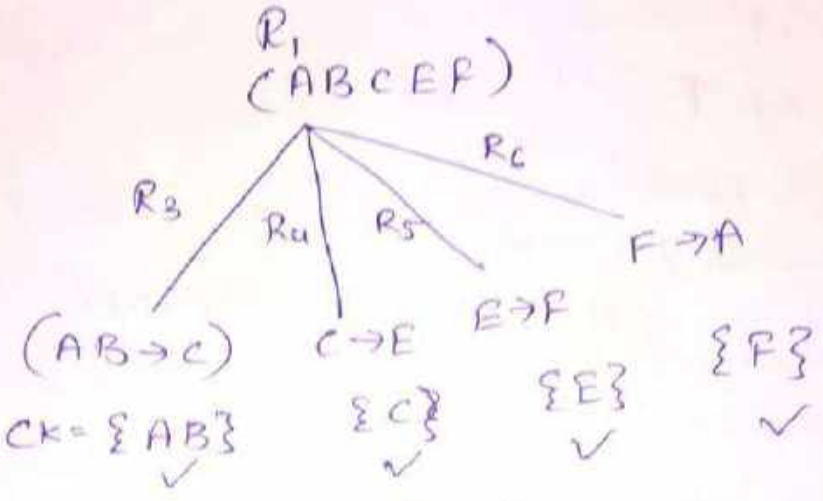
$F \to A$ ✗

$C \to D$ ✓

R_2 is BCNF

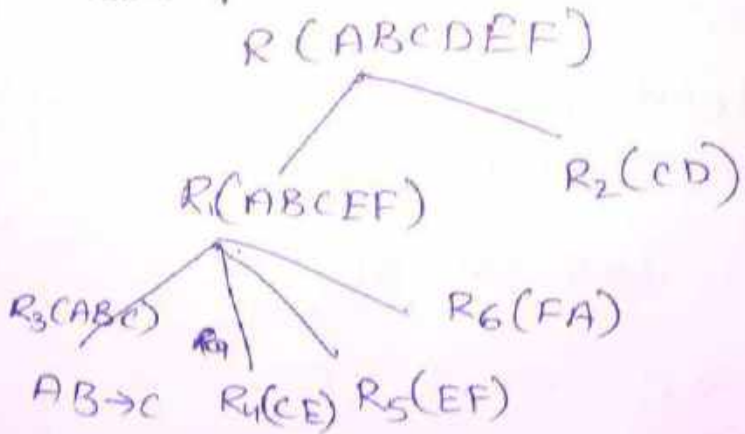
0% Redancy.

R_1 is not in BCNF



0% Redundancy.

∴ BCNF



Scanned by CamScanner

other kinds of Dependencies :-

i) Multivalued dependencies :-

A table is said to ~~be~~ have multivalued dependency, following conditions are true

i) For dependency $A \to B$, if for a single value of A, multiple values of B exists, then table may have multi-values dependency.

ii) A table should have at-least 3 columns for it to have a multi-valued dependency.

iii) for relation R(A, B, C) if there is a multi valued dependency between A and B then B and C should be independent of each other

if all these conditions are true for any relation it is said to be multivalued dependency

multivalued dependency is denoted by $\to\to$

Ex:- student

sid	course	language
1	C C++	English German
2	Java	English French

Here sid = 1 leads to multiple values for courses and languages.

Scanned by CamScanner

Student

Sid	course	languages
1	C	English
1	C++	German
1	C	German
1	C++	English
2	Java	English
2	Java	French

Here Sid and course are dependent and course, language are independent. Multivalued dependency denoted by.

Sid →→ course

Sid →→ languages.

if there exists a multivalued dependency then table is not in 4th Normal form.

2) 4NF :-

if should be BCNF, the table should not have any multivalued dependency.

ex:- take above table. Student

Now convert above table to 4NF, we must decompose table into following two tables.

student_coursetable

Sid	Course
1	C
1	C++
2	Java

Key : (Sid, course)

student-lang Table

Sid	languages
1	English
1	German
2	English
2	French

Key : (sid, languages)

∴ Tables are in 4NF

Scanned by CamScanner

Fifth Normal form (5NF)

A relation is in 5NF if it is in 4NF and Not having any join dependency and joing in should be lossless.

A relation that has a join dependency can not be divided into two (or more) relations such that the resulting tables can be combined to form the original table.

then $(R_1 \bowtie R_2) \bowtie R_3$ (or)

$R_1 \bowtie (R_2 \bowtie R_3)$

* also called as (PNF) Project Normal form.

Schema Refinement in Database Design

* we already know how normalizatim can eliminate redundancy and serveral approaches to normalizing a relation. apply these ideas on conceptual design.

1) Constraints on an Entity Set :-

consider Hourly_Emps relation. ssn attribute is key then FD are.

{ssn} → {ssn, name, lot, rating, hourly-wages, hours-worked}.

write single letters for thuses attributes are

Scanned by CamScanner

FD as S → SNLRWH

here constraint is that hourly-wages attribute is determined by rating attribute (ie) R → W.

This FD leads to redundant storage of rating-wage associations. So decompose table into two tables having ~~association~~ Relationship is Has-wages.

Two tables (or) Entities are Hourly-Emps and wage table.

2) Constraints on a Relationship set :-

Suppose we have entity sets — Parts, Suppliers, and Departments. Schema for contracts as CQPSD

C - contract id, Q - quantity, P - part, D - Department

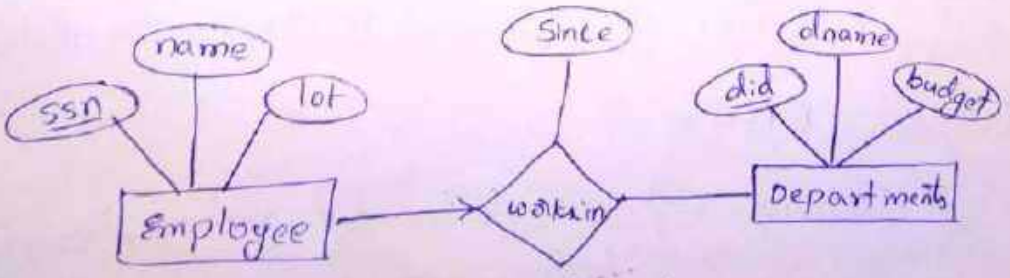
S - supplier.

here constraint is a policy that a department purchases at most one part from any given supplier.

Several constracts between same supplier and department.

FD JDS → P. decomposing contracts into two relations with attributes CQSD and SDP.

3) Identifying Attributes of entities :-



works_in Relationship Set.

Scanned by CamScanner

using key constraint, we can translate this ER diagram into two relations.

workers (ssn, name, lot, did, since)

Departments (did, dname, budget)

Entity set employees and relationship set works_In are mapped to single relation, workers

FD: did → lot

the redundancy in this design can be eliminated by decomposing the workers relation into two relations.

workers2 (ssn, name, did, since)

Dept_lots (did, lot)

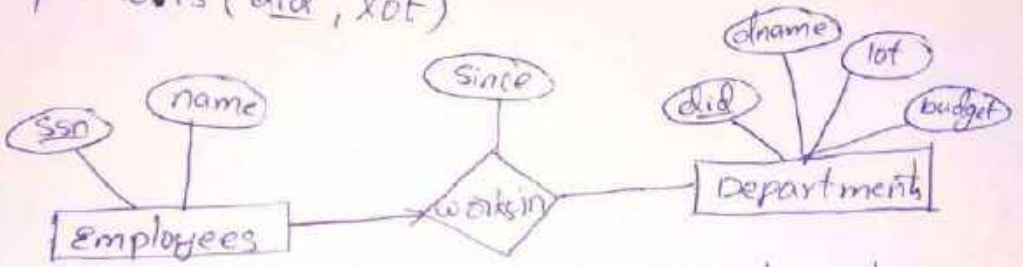


fig: Refined works_IN Relationship set

Translating this diagram into relational model would yield:

workers2 (ssn, name, did, since)

Departments (did, dname, budget, lot)

4) Identifying entity sets :-

Constraint is every sailor uses a unique credit card for reservations. ie FD: S → C. In relation Reserves, we store credit card number for a sailor as often reservations for that sailor, we have redundency and potential update anomalies. decompose Reserves into two relations with attributes SBD and SC

Scanned by CamScanner

Transaction in DBMS :-

A transaction is a logical unit of processing in a DBMS which entails one or more database access operation.

Let's take an example of a simple transaction. Suppose a bank employee transfer Rs 10000 from A's account to B's account.

The following are the steps involved in the above transaction.

1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B);

In the above transaction R refers to the Read operation and W refers to the write operation.

Transaction failure in between the operations :-

The main problem that can happen during a transaction is that the transaction can fail before finishing all the operations in the set. This can happen due to Power failure, System crash etc. To solve the problem, we have the following two operations.

Commit :- If all the operations in a transaction are completed successfully then commit those changes to the database permanently.

Roll back :- If any of the operation fails then rollback all the changes done by previous operations.

Scanned by CamScanner

may arise during transaction but they are not sufficient when transactions are running concurrently. To handle these problems we need to understand database ACID Properties.

ACID Properties:-

Atomicity:- This Property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transact is left partially completed. states should be defined either before the execution of the transaction or after the execution/abortion/fail of the transaction.

Consistency:- The database must remain in a consistent state after an transaction. no transaction should have any adverse effect on the data residing in the database. If the database was in a consiste state before the execution of a transaction, it must remain consis after the execution of the transaction as well.

Durability:- The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the databa will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that dat will be updated once the system springs back into action.

Isolation:- In a database system where more than one transaction are being executed simultaneously and in

Scanned by CamScanner

allel, the property of isolation states that all the transactions will be executed and executed as if it is the only transaction in the system. no transaction will affect the existence of any other transaction.

DBMS serializability :-

When multiple transactions are running concurrently then there a possibility that the database may be left in an inconsistent state. serializability is a concept that helps us to check which schedules a serializable. A serializable schedule is the one that always leaves the database in consistent state.

A serializable schedule always leaves the database in consistent state. A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finish execution. However a non-serial schedule needs to be checked for serializ

A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of these n transactions. A serial schedule doesn't allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

Types of serializability.

There are two types of serializability.

1. conflict serializability.
2. view ~~serializability~~.

Scanned by CamScanner

Conflict serializability :- is one of the type of serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.

A ~~conflict~~ schedule is called conflict serializable if we can convert into a serial schedule after swapping its non-conflict operations.

Conflicting operations.

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Let see some examples to understand this:

Ex:- operation w(x) of a transaction T1 and operation R(x) of transac T_2 are conflicting operations, because they satisfy all the three conditions mentioned above. They belong to different transactions, ll are working on same data item X, one of the operation in write ope

Conflict Equivalent schedules:-

Two schedules are said to be conflict Equivalent if one schedule c be converted into other schedule after swapping non-conflicting operatio

Conflict serializable check:- Lets check whether a schedule is confli serializable & not. If a schedule is conflict equivalent to its serial schedule then it is called conflict serializable schedule.

Lets take few examples of schedules.

Scanned by CamScanner

Example of conflict serializability :-

Lets consider this schedule.

T_1	T_2
R(A)	
R(B)	
	R(A)
	R(B)
	W(B)
W(A)	

To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T_2 with the W(A) operation of transaction T_1. However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is not conflict serializable.

Lets take another example!-

T_1	T_2
R(A)	
	R(A)
	R(B)
	W(B)
R(B)	
W(A)	

Lets swap non-conflicting operations.

Scanned by CamScanner

After swapping R(A) of T1 and R(A) of T2 we get:

T_1	T_2
	R(A)
R(A)	
	R(B)
	W(B)
R(B)	
W(A)	

After swapping R(A) of T1 and R(B) of T2 we get:

T_1	T_2
	R(A)
	R(B)
R(A)	
	W(B)
R(B)	
W(A)	

After swapping R(A) of T1 and R(B) of T2 we get:

T_1	T_2
	R(A)
	R(B)
R(A)	
	W(B)
R(B)	
W(A)	

After swapping R(A) of T_1 and W(B) of T_2 we get:

Scanned by CamScanner

T_1	T_2
	R(A)
	R(B)
	W(B)
R(A)	
R(B)	
W(A)	

we finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is conflict serializable.

View serializability:-

View serializability is a process to find out that a given schedule is view serializable or not.
To check whether a given schedule is view serializable, we need to check whether the given schedule is view equivalent to its serial schedule. Lets take an example to understand what I mean by that.

Given schedule:-

T_1	T_2
R(x)	
W(x)	
	R(x)
	W(x)
R(y)	
W(y)	
	R(y)
	W(y)

Scanned by CamScanner

Serial schedule of the above given schedule:-

As we know that in serial schedule a transaction only starts when the current running transaction is finished so the serial schedule of the above given schedule would look like this:

T_1	T_2
R(x)	
w(x)	
R(y)	
w(y)	R(x)
	w(x)
	R(y)
	w(y)

If we can prove that the given schedule is view equivalent to its serial schedule then the given schedule is called view serializable.

View Equivalent:- Lets learn how to check whether the two schedules are view equivalent.

Two schedules T_1 and T_2 are said to be view equivalent if they satisfy all the following conditions.

1. Initial read:- Initial read of each data item in transactions must match in both schedules.
2. Final write:- Final write operations on each data item must match in both the schedules.

Scanned by CamScanner

2. Write Read:- If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item.

View Serializable:-

If a schedule is view equivalent to its serial schedule then the given schedule is said to be view serializable. Lets take an example.

Non-serial S1		Serial S2	
T1	T2	T1	T2
R(X)		R(X)	
W(X)		W(X)	
	R(X)	R(Y)	
	W(X)	W(Y)	
R(Y)			R(X)
W(Y)			W(X)
	R(Y)		R(Y)
	W(Y)		W(Y)

Storage structure:-

Storage structure is the memory structure in the system. It is mainly divided into two categories.

Volatile memory:- These are the primary memory devices in the system, and are placed along with the CPU. These memories can store only small amount of data, but they are very fast. Ex:- main memory, cache memory etc. these memories cannot endure system crashes- data in these memories will be lost on failure

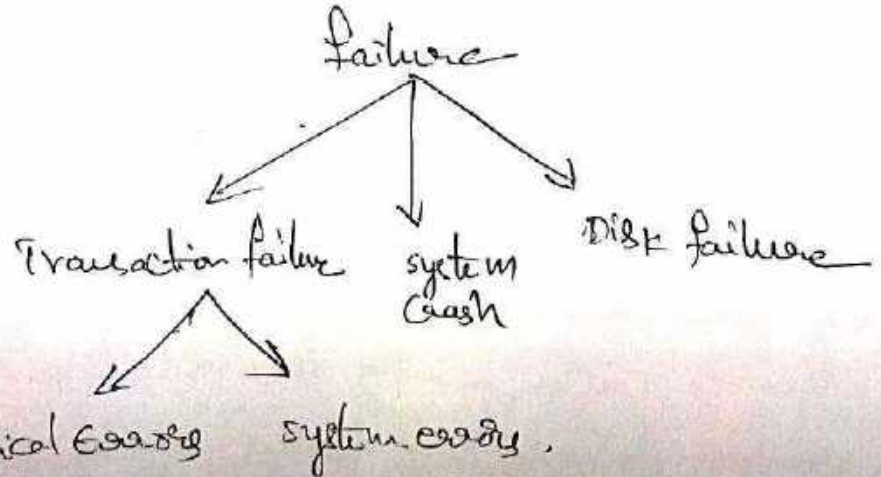
Scanned by CamScanner

Non-volatile memory:-

These are secondary memories and are huge in size, but slow in processing. Eg:- flash memory, hard disk, magnetic taps etc. these memories are designed to withstand system crashes.

stable memory:- This is said to be third form of memory structure but it is same as non volatile memory. In this case, copies of same non volatile memories are stored at different places. This is because, in case of any crash and data loss, data can be recovered from other copies. This is even helpful if there one of non-volatile memory is lost due to fire & flood. It can be recovered from other network location. But there can be failure while taking the backup of DB into different stable storage devices. even it may fail to transfer all the data successfully either it will Partially transfer the data to remote devices or Completely fail to store the data in stable memory. Hence extra caution has to be taken while taking the backup of data from one stable memory to other.

failure classification:-



Scanned by CamScanner

Isolation Levels :

As we know that, in order to maintain consistency in a database, it follows ACID properties. Among these four properties isolation determines how transaction integrity is visible to other users and systems. It says that a transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a data system. Isolation levels defines the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system. A transaction isolation levels are defined by the following phenomena—

→ Dirty Read :- A dirty read is the situation when a transaction reads a data that has not yet been committed. For example, let's say transaction 1 updates a row and leaves it uncommitted, meanwhile transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.

→ Non Repeatable Read :- Non Repeatable read occurs when a transaction reads same row twice, and get a different value each time. For example, suppose transaction T_1 reads a data. Due to concurrency, another transaction T_2 updates the same data and commit, now if transaction T_1 reads the same data it will retrieve a different value.

Scanned by CamScanner

-> Phantom Read :- Phantom read occurs when two same transactions are executed, but the rows retrieved by the two, are different. For example, Suppose transaction T_1 retrieves a set of rows that satisfy some search criteria. Now, transaction T_2 generates some new rows that match the search criteria for transaction T1. If transaction T_1 re-executes the statement that reads the rows, it gets a different set of rows this time.

Based on these phenomena, The SQL standard defines four isolation levels:

1. Read uncommitted :- Read uncommitted is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transaction, thereby allowing dirty reads. In this level, transactions are not isolated from each other.
2. Read committed :- This isolation level guarantees that any data read is committed at the moment it is read. Thus it does not allow dirty read. The transaction holds a read or write lock on the current row, and thus prevent other transactions from reading, updating or deleting it.
3. Repeatable Read :- This is the most restrictive isolation level. The transaction holds read locks on all rows it

inconsistency and write locks on all rows it inserts, updates or deletes. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.

4. Serializable!- This is the highest isolation level. A serializable execution is guaranteed to be serializable. Serializable Execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

Concurrency Control!-

Concurrency control is the Procedure in dbms for managing simultaneous operations without conflicting with each other.

Potential Problems of concurrency.

There are some issues which you will likely to face while using the concurrency control method.

-> Lost update!- occurs when multiple transactions select the same row and update the row based on the value select

-> uncommitted dependency issues occur when the second transaction is ~~trying to access~~ selects a row which is updated by another transaction (dirty read).

Scanned by CamScanner

Non-Repeatable read:- occurs when a second transaction trying to access the same row several times and reads different data each time.

Incorrect summary issues occurs when one transaction takes summary over the value of all the instances of a repeated data-item and second transaction update few instances of ~~a~~ repeat that specified data-item. In that situation, the resulting summary does not reflect a correct result.

Concurrency control Protocols:-

Different concurrency control Protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.

-> lock-Based Protocols
-> Two Phase
-> Timestamp-Based Protocols
-> Validation-Based Protocols.

Lock-based protocols:-

A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks help synchronize access to the database items by concurrent transactions.

All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

Scanned by CamScanner

Binary locks:- A Binary lock on a data item can either locked or unlocked states.

shared/exclusive:- This type of locking mechanism ~~separates~~ the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

1. Shared lock (S):-

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.

for ex:-, consider a case where two transactions are reading the account balance of a person. the database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

2. Exclusive lock (X):-

With the Exclusive lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same idea item. X-lock is requested using lock-x instruction. Transaction may unlock the data item after finishing the 'write' operation.

for example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing lock on it. Therefore when the second transaction wants to read or

Scanned by CamScanner

with exclusive lock granted to it's operations.

3. Simplistic Lock Protocol:-

- This type of lock-based Protocols allows transactions to obtain a lock on every object before beginning operation Transactions may unlock the data-item after finishing the 'write' operation.

4. Pre-claiming locking:-

Pre-claiming lock Protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process. In the situation when all locks are granted, the transaction exec After that, all lock release when all of its operations are

Starvation:- Starvation is the situation when a transacti needs to wait for an indefinite Period to acquire a lock.

following are the reasons for starvation.

→ when waiting scheme for locked items is not properly manage

→ In the case of resource leak.

→ The same transaction is selected as a victim repeatedly.

Scanned by CamScanner

Multiversion schemes

-> multiversion concurrency control keeps the old values of a data item when it is updated.

-> when a transaction requires access to an item, an appropriate version is chosen to maintain the serializability of the currently executing schedule, if possible.

-> An obvious drawback is that more storage is needed to maintain multiple versions of the database items.

Multiversion technique based on TO — In this method, for each version, the value of version x_i and the following two timestamps are kept:

-> Read_TS(x_i): the largest of all the timestamps of transactions that have successfully read version x_i.

-> write_TS(x_i): the timestamp of the transaction that wrote the value of version x_i.

TO ensure serializability, the following two rules are used:

-> If transaction T issues a write_item(x) operation, and x_i has the highest write_TS(x_i) of all versions of x that is also less than or equal to TS(T), and Read_TS(x_i) > TS(T), then abort and roll back transaction T; otherwise create a new version x_j of x with

Read_TS(x_j) = write_TS(x_j) = TS(T)

Scanned by CamScanner

→ If transaction T issues a Read_item(x) operation, find x_i that has the highest write_TS(x_i) of all versions of x that is also less than or equal to TS(T); then return the Read_TS(x_i) to the larger of TS(T) and the current Read_TS(x_i).

Multiversion 2PL using certify locks:- In this scheme there are three locking modes for an item; Read, write and certify.

→ This scheme allows other transactions 'T' to read an item x while a single transaction T holds a write lock on x. This is done by allowing two versions for each item x.

→ once T is ready to commit, it must obtain a certify lock on all items that is currently holds write locks on before it can commit.

Lock compatibility tables:-

→ A compatibility table for read/write locking scheme.

→ A compatibility table for read/write/certify locking scheme.

	Read	write
Read	yes	no
write	no	no

Scanned by CamScanner

	Read	write	Conflict
Read	Yes	yes	no
write	Yes	no	no
Conflict	no	no	no

Recovery system - failure classification:-

The durability and robustness of a dbms depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

failure classification.

To see where the problem has occured, we generalize a failure into various categories, as follows.

Transaction failure.

A transaction has to abort when it fails to execute & when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Scanned by CamScanner

Reasons for a transaction failure could be —

-> logical errors — where a transaction cannot complete because it has some code error or any internal error condition.

-> system errors — where the database system itself terminates an active transaction because the dbms is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash:-

There are problems - external to the system - that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in Power supply may cause the failure of underlying hardware or software failure.

Ex:- operating system errors.

Disk failure:- In early days of technology evolution, it was a common problem where hard disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Scanned by CamScanner

Storage Structure :-

We have already described the storage system. In brief the storage structure can be divided into two categories.

Volatile storage - As the name suggests, a volatile storage cannot survive system crashes. volatile storage devices are placed very close to the CPU. normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.

Non-volatile storage :- These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. examples may include hard disks, magnetic tapes, flash memory and non-volatile RAM.

Recovery and Atomicity :-

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations which are atomic in nature. But according to ACID Properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none

Scanned by CamScanner

When a DBMS recovers from a crash, it should maintain the following-

-> It should check the states of all the transactions, which were being executed.

-> A transaction may be in the middle of some operation, the DBMS must ensure the atomicity of the transaction in this case.

-> It should check whether the transaction can be completed now or it needs to be rolled back.

-> No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction -

-> maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.

-> maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Scanned by CamScanner

Log- based Recovery:-

Log is a sequence of records, which maintains the records of actions Performed by a transaction. It is important that the logs are written Prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows—

-> The log file is kept on a stable storage media.

-> when a transaction enters the system and starts execution, it writes a log about it.

<Tn, start>

when the transaction modifies an item x, it write logs as follows—

<Tn, x, V1, V2>

It reads Tn, has changed the value of x, from V1 to V2.

-> when the transaction finishes, it logs-

<Tn, commit>.

The database can be modified using two approaches.

Deferred database modification:- All logs are written on

Scanned by CamScanner

to the stable storage and the database is updated when a transaction commits.

. > Immediate database modification:- Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Recovery with Concurrent transactions:-

When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

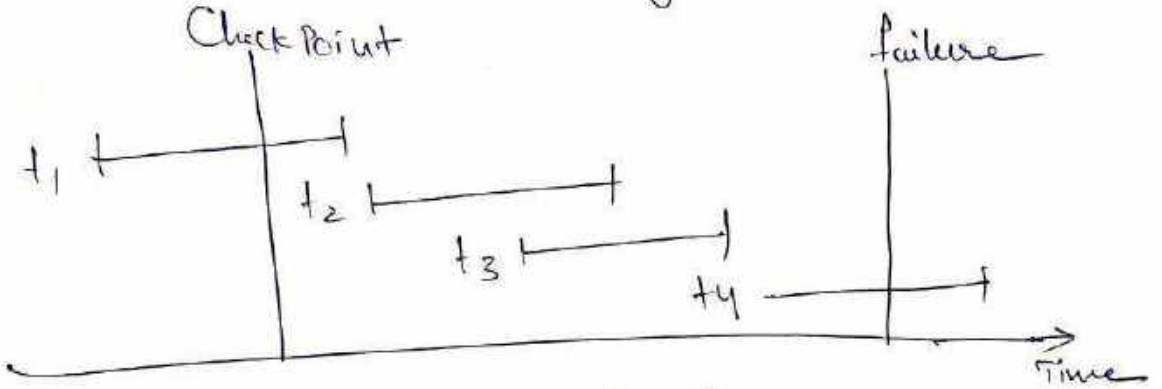
Check Point:-

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently

in a storage disk. checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery:-

When a system with concurrent transactions crashes and recovers, it behaves in the following manner-



-> The recovery system reads the logs backwards from the end to the last Check Point.

-> It maintains two lists, an undo-list and a redo-list.

-> If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{commit} \rangle$ it puts the transaction in the redo-list.

-> If the recovery system sees a log with $\langle T_n, \text{start} \rangle$ but no commit or abort log found, it puts the transaction in undo-list.

Scanned by CamScanner

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

Buffer management :-

-> Database maintains an in-memory buffer of data blocks.

-> when a new block is needed, if buffer is full an existing block needs to be removed from buffer.

-> If the block chosen for removal has been updated, it must be output to disk.

-> Database buffer can be implemented either

-> in an area of real main-memory reserved for the database or in virtual memory.

-> Implementing buffer in reserved main-memory has drawbacks.

-> memory is partitioned before-hand between database buffer and applications, limiting flexibility.

-> needs may change, and although operating system knows best how memory should be divided up to any time, it cannot change the partitioning of memory.

Scanned by CamScanner

-> Database buffers are generally implemented in virtual memory in spite of some drawbacks.

-> when operating system needs to evict a page that has been modified, to make space for another page, the page is written to swap space on disk.

-> when database decides to write buffer page to disk, buffer page may be in swap space and may have to read from swap space on disk and output to the database on disk, resulting in extra I/O.

-> known as dual paging problem.

-> Ideally when swapping out a database buffer page, operating system should pass control to database, which in turn outputs page to database instead of swap space (making sure to output log records first).

-> Dual paging can thus be avoided, but common operating systems do not support such functionality.

Failure with loss of nonvolatile storage.

-> so far we assumed no loss of non-volatile storage

-> Techniques similar to checkpointing used to deal with loss of nonvolatile storage.

Scanned by CamScanner

→ Periodically dump the entire content of the database to stable storage.

→ no transaction may be active during the dump procedure, a procedure similar to checkpointing must take place.

→ output all log records currently residing in main memory onto stable storage.

→ output all buffer blocks onto the disk.

→ copy the contents of the database to stable storage.

→ output a record <dump> to log on stable storage.

→ To recover from disk failure

→ restore database from most recent dump.

→ Consult the log and redo all transactions that committed after the dump.

→ Can be extended to allow transactions to be active during dump; known as fuzzy dump or online dump.

→ Recovery Algorithm.

→ Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite failures.

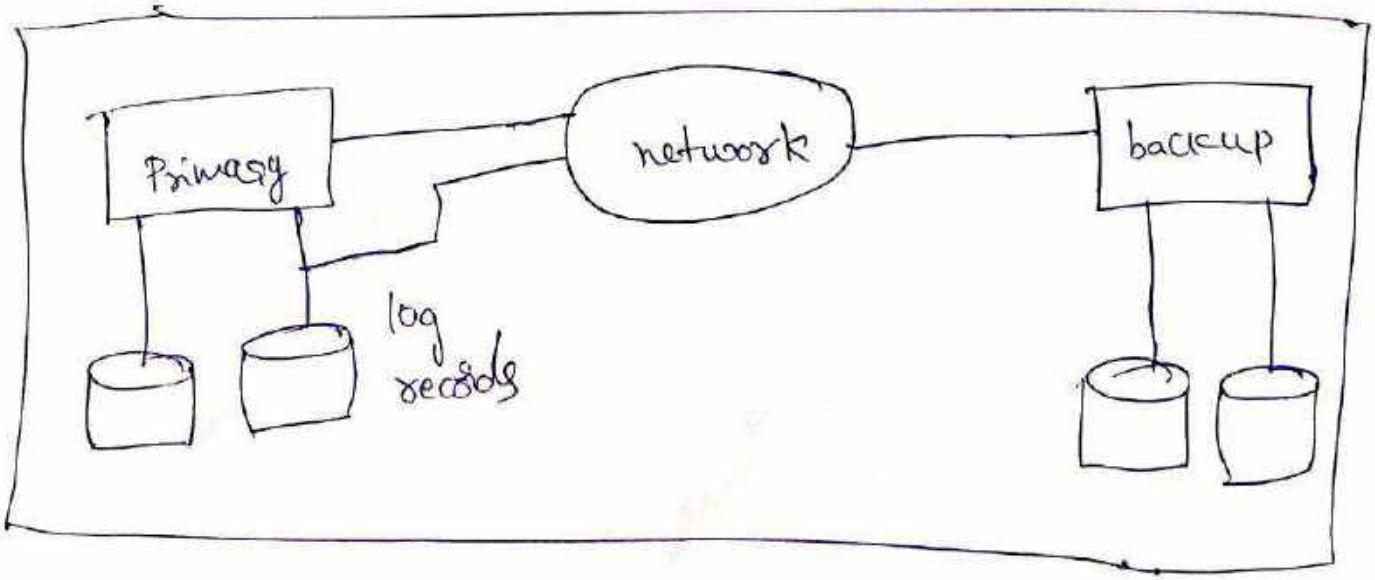
Scanned by CamScanner

-> Focus of this :-

1. Actions taken during normal transaction Processing to ensure enough information exists to recover from failures
2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability.

Remote Backup systems.

Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.



Scanned by CamScanner

Detection of failure :- Backup site must detect when Primary site has failed

→ to distinguish Primary site failure from link failure maintains several communication links between the Primary and the remote backup.

Transfer of control :-

→ To take over control backup site first performs recovery using its copy of the database and all the log records it has received from the Primary.

→ Thus, completed transactions are redone and incomplete transactions are rolled back.

→ when the backup site takes over Processing it becomes the new Primary.

→ To transfer control back to old Primary when it recovers, old Primary must receive redo logs from the old backup and apply all updates locally.

Time to recover :- To reduce delay in takeover, backup site periodically Processes the redo log

Scanned by CamScanner

- records, performs a checkpoint, and then can delete earlier parts of the log.

Hot-spare!— Configuration permits very fast takeover.

-> Back up continually processes redo log record as they arrive, applying the updates locally.

-> when failure of the Primary is detected the backup rolls back incomplete transactions, and is ready to process new transactions.

-> Alternative to remote 'back up' distributed database with replicated data

-> Remote backup is faster and cheaper, but less tolerant to failure.

-> Ensure durability of update by delaying transaction commit until update is logged at backup; avoid this delay by permitting lower degrees of durability.

One-safe!— Commit as soon as transactions commit log record is written at Primary.

-> Problem!— updates may not arrive at backup before it takes over.

Scanned by CamScanner

Two-very-safe:- commit when transaction's commit log record is written at Primary and backup.

-> Reduced availability since transactions cannot commit if either site fails.

Two-safe:- Proceed as in two-very-safe if both Primary and backup are active. If only the Primary is active, the transaction commits as soon as its commit log record is written at the Primary.

-> Better availability than two-very safe, avoids problem of lost transactions in one-safe.

Scanned by CamScanner

Concurrency Control:

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each other. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical database, would have a mix of reading & WRITE operations and hence the concurrency is a challenge.

Concurrency control is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases.

Problems of Concurrency:

Here are some issues, which you will likely to face while using the concurrency control method:

(1) Lost updates occur when multiple transactions select the same row and update the row based on the value selected.

(2) Uncommitted dependency issues occurs when the second transaction is trying to access the same row several times and reads different data each time.

Scanned by CamScanner

(3) Non-Repetable Read occurs when a second transaction is trying to access the same row several times and reads different data each time.

(4) Incorrect Summary issue occurs, when one transaction takes summary over the value of all the instances of repeated data-item, and second transaction update few instances of that specific data-item. In that situation the resulting summary does not reflect a correct result.

Concurrency control protocols:

(1) Lock-Based protocols:

A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks help synchronize access to the database items by concurrent transactions.

All lock requests are made to the concurrency control manager. Transactions proceed only once the lock request is granted.

Binary locks: A Binary lock on a data item can either locked or unlocked states.

Scanned by CamScanner

Shared / exclusive: This type of locking mechanism separates the locks based on their use. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

(1) Shared lock (S): A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions, because you will never have permission to update data on the data item.

Ex:- Consider a case where two transactions are reading the account balance of a person. The database will let them read by placing a shared lock. However, if another transaction wants to update that account's balance, shared lock prevent it until the reading process is over.

(2) Exclusive lock (X): With the exclusive lock, a data item can be read as well as written. This is exclusive and can't be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the 'write' operation.

Ex:- When a transaction needs to update

Scanned by CamScanner

the account balance of a person. You can allow this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write exclusive lock prevent this operation,

③ Simplistic Lock protocol: This type of lock-based protocols allows transactions to obtain a lock on every object before begining operation. Transactions may unlock the data item after finishing the 'write' operation.

④ Pre-claiming locking: Pre-claiming lock protocol helps to evaluate operations & create a list of required data items which are needed to initiate an execution process. In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

Starvation: Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Reasons for starvation: ⓐ when waiting scheme for locked items is not properly managed.

ⓑ In the case of resource leak.

ⓒ The same transaction is selected as victim repeatedly.

Scanned by CamScanner

Deadlock : Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

Two phase locking (2PL) protocol :

Two phase locking protocol which is also known as 2PL protocol. It is also called P2L.

In this type of locking protocol, the transaction should acquire a lock after it releases one of its locks.

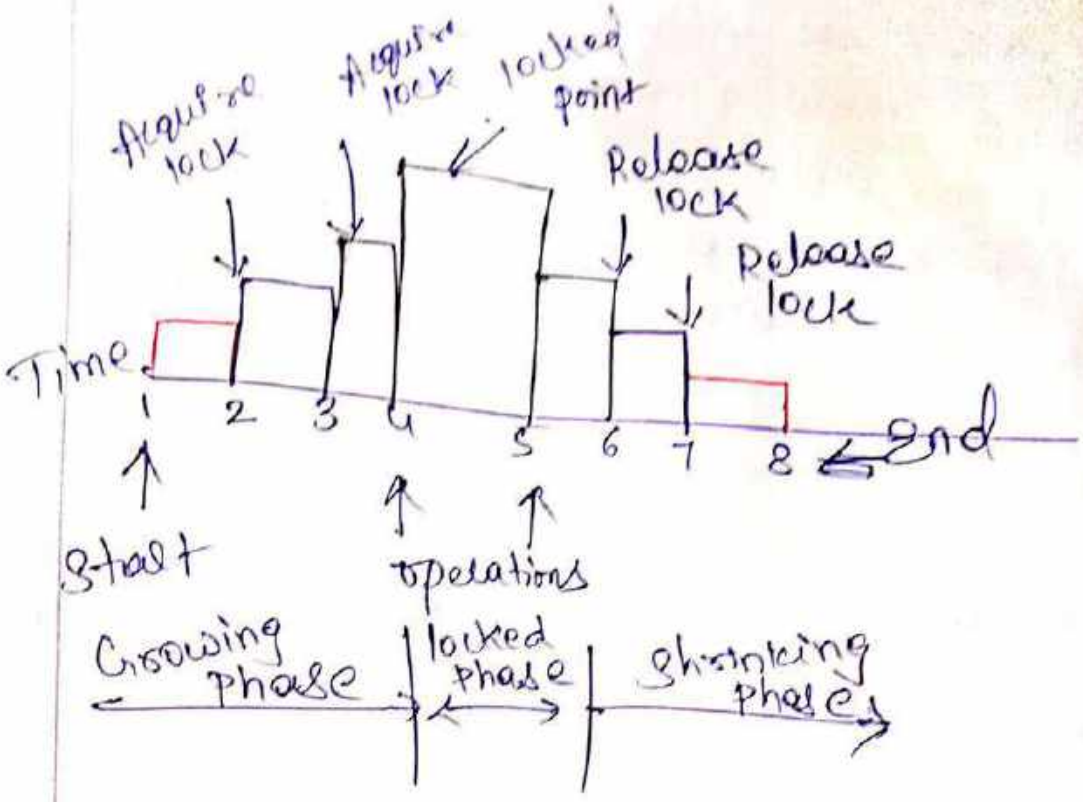
This locking protocol divides the execution phase of transaction into 3 different parts.

(a) In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.

(b) The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.

(c) In this third phase, the transaction cannot demand any new locks. Instead it only releases the acquired locks.

Scanned by CamScanner



The Two-phase locking protocol allows each transaction to make a lock or unlock request in two steps:

(a) Growing phase: In this phase transaction may obtain locks but may not release any locks.

(b) Shrinking phase: In this phase, a transaction may release locks but not obtain any new lock.

Strict Two-phase locking method:

Strict Two-phase locking system is almost similar to 2PL. The only difference is that Strict-2PL never releases a lock after using it. It holds all the locks until the commit point and releases all the locks at one go, when the process is over.

Scanned by CamScanner

Centralized 2PL:

In a centralized 2PL, a single site is responsible for lock management process. It has only one lock manager for the entire DBMS.

Primary copy 2PL:

Primary copy 2PL mechanism, many lock managers are distributed to different sites. After that, a particular lock manager is responsible for managing the lock for the set of data items. When the primary copy has been updated, The change is propagated to slaves.

Distributed 2PL:

In this kind of two-phase locking mechanism, lock managers are distributed to all sites. They are responsible for managing locks for data at that site. If no data is replicated, it is equivalent to primary copy 2PL. Communication costs of distributed 2PL are quite higher than primary copy 2PL.

Timestamp-based protocols:

This timestamp-based algorithm uses a timestamp to serialize the execution of concurrent transactions. This protocol ensures that every conflicting read & write operations are executed in timestamp order.

Scanned by CamScanner

This protocol uses the System Time or logical count as a Time Stamp.

The older transaction is always given priority in this method. It uses System time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.

Lock-Based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.

Ex:-
Suppose 3 transtions occured T_1, T_2, T_3.

T_1 has entered the system at time 0010
T_2 " " " " " 0020
T_3 " " " " " 0030

Priority will be given to transaction T_1, then T_2 & lastly T_3.

Pro's & Con's of concurrency contn.

Advantages: ① Schedules are serializable just like 2PL protocols. ② No waiting for the Transaction, which eliminates the possibility of deadlocks.

Disadvantages: Starvation is possible if the same transaction is restarted & continually aborted.

Scanned by CamScanner

Validation Based protocol: Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases.

(1) Read phase: In this phase, the transaction T is read & executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform the write operations on temporary variables without an update to the actual database.

(2) Validation phase: In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.

(3) Write phase: If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

Start (Ti): It contains the time when Ti started its execution.

Scanned by CamScanner

Validation(T_i) : It contains the time when T_i finishes its read phase and starts its validation phase.

Finish(T_i) : It contains the time when T_i finishes its write phase.

- This protocol is used to determine the timestamp for transaction for serialization using the time stamp of the validation phase.

Hence $TS(T) = validation(T)$.

- The serializability is determined during the validation process.
- while executing the transaction, it ensures a greater degree of concurrency or less no of conflicts.
- Thus it contains Transactions which have less no of rollbacks.

Scanned by CamScanner

UNIT – V

Overview of Storage & Indexing

① The basic abstraction of data in a DBMS is a collection of records or a file and each file consists of one or more pages.

② The files and access methods software layer organizes data carefully to support fast access to desired subsets of records.

③ Understanding ~~data carefully~~ how records are organised is essential to using a database system effectively.

④ File organization is a method of arranging the records in a file when the file is stored on disk.

Each file organization makes certain operation efficient but other operations expensive

⑤ Ex:- Consider a file of employee records, each containing age, name, and Sal fields, which we use as Ex:-.

If we want to retrieve employee records in order of increasing age, sorting the file by age is a good file organisation, but the sort order is expensive to maintain if the file is frequently modified.

Scanned by CamScanner

Further, we are often interested in supporting more than one operation on a given collection of records.

In our ex:- we may also want to retrieve all employees who make more than $5000. we have to scan the entire file to find such employee records

(6) INDEXING :- A technique called indexing can help when we have to access a collection of records in multiple ways, in addition to efficiently support various kinds of selection.

Indexing, an IMP aspect of file-organisation in a DBMS.

Data on external storage :-

(1) A DBMS stores vast quantities of data, and the data must persist (continue in spite of action) over program executions.

Therefore, data is stored on external storage devices such as disks, tapes, and fetched into main memory as needed for processing.

The unit of info read from or written to disk is a page.

The size of a page is a DBMS parameter, and typical values are 4KB or 8 KB.

(2) The cost of page I/O (I/p from disk to main memory and O/p from memory to disk) dominates the cost of typical database operations, and DB carefully optimised to minimise

3) While the details of how files of records are physically stored on disk and how main memory is utilised.

4) The following points are IMP to keep in mind.

(i) DISKS are the most IMP external storage device They allow us to retrieve any page at a (more a less fixed cost per page.

However, if we read several pages in the order that they are stored physically, the cost can be much ~~more~~ less than the cost of reading the same pages in random order.

(ii) Tapes are sequential access devices and force us to read data one page after the other.

They are mostly used to archieve data that is not needed on a regular basis.

(iii) Each record in a file has a unique identifier called a record id, or rid for short.

An rid has the property that we can identify the disk address of the page containing the record by using the rid.

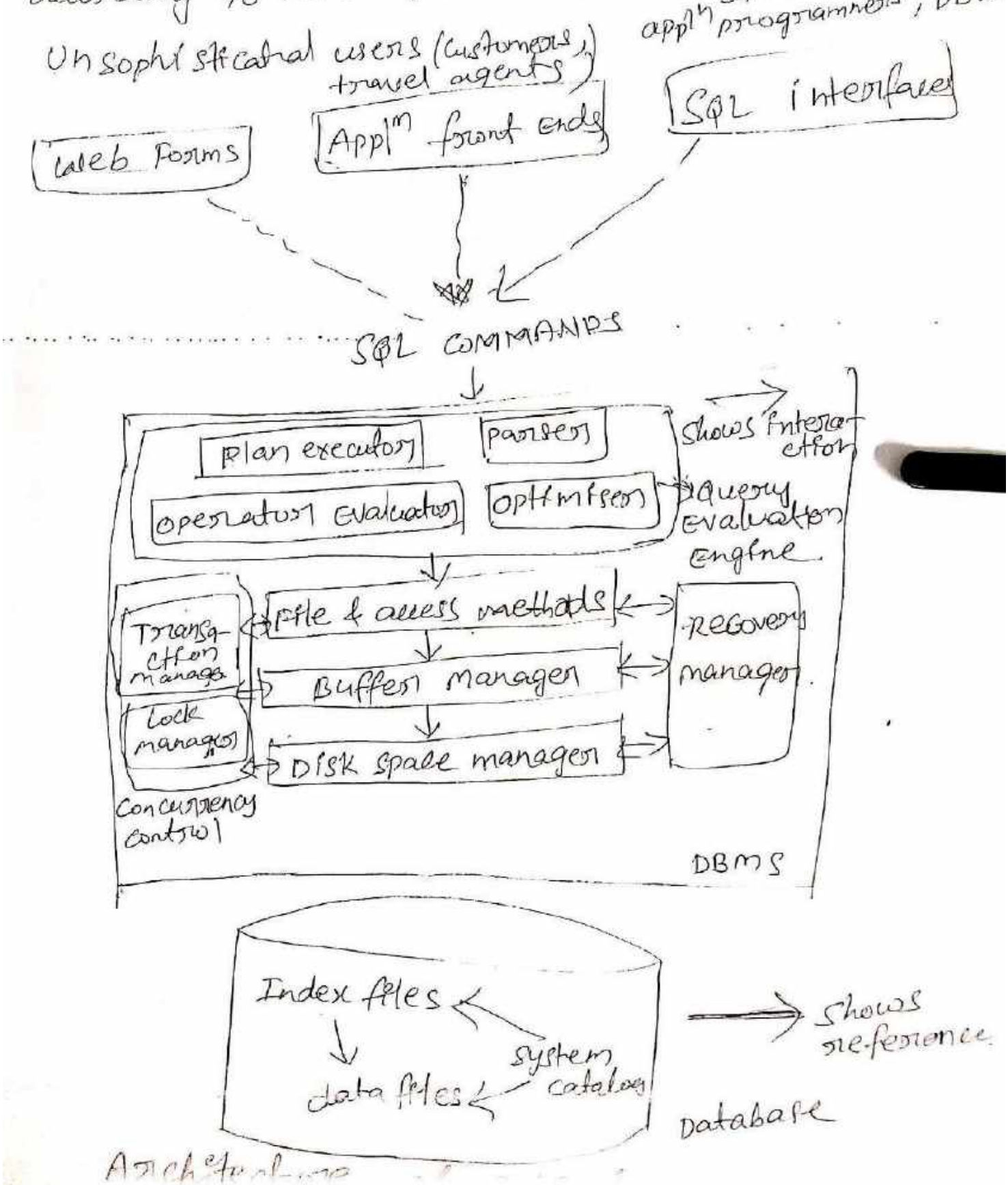
(5) Data is read into memory for processing, and written to disk for persistent storage, by a layer of s/w called the buffer manager.

(6) When the files & access methods layer needs to process a page, it asks the buffer man

to fetch the page, specifying the pages needed.

The buffer manager fetches the page from disk if it is not already in memory.

(7) Space on disk is managed by disk space manager according to DBMS s/w architecture.



Architecture

(8) When the files and access methods layer needs additional space to hold new records in a file, it asks the disk space manager to allocate an additional disk page for the file; it also informs the disk space manager when it no longer needs one of its disk pages.

(9) The disk space manager keeps track of the pages in use by the file layer; if a page is freed by the file layer, the space manager tracks this, and reuse the space if the file layer requests a new page later on.

File organizations and Indexing :-

(1) The files of records is an abstraction in a DBMS, and is implemented by the file and access methods layer of the code. (2) Files can be created, destroyed, and have records into and deleted from it. It also supports scans; a Scan operation allows us to step through all the records in the file one at a time.

(3) A relation is typically stored as file of record

(4) A file layer stores the records in a file in a collection of disk pages.

It keeps track of pages allocated to each file and as records are inserted into and deleted for the file, it also tracks available space within pages allocated to the file.

(5) Heap file :- The simplest file structure is a unordered file, or heap file.

Records in a heap file are stored in random order across the page of the file.

Scanned by CamScanner

A heap file organisation supports retrieval of all records or a particular record specified by its rid;

The file manager must keep track of the pages allocated for the file.

⑥ Index:- An index is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations.

An index allows us to efficiently retrieve all records that satisfy search conditions on the search key fields of index.

We can also create additional indexes on a given collection of data records, each with a different search key, to speed up search operations that are not efficiently supported by the file organisation used to store the data records.

⑦ Consider ex: of employee records.

We can store the records in a file organised as an index on employee age; this is an alternative to sorting the file by age.

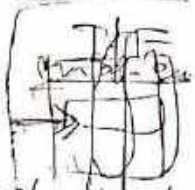
Additionally, we can create an auxiliary index file based on salary, to speed up queries using involving salary.

The 1st file contains emp records, and the 2nd contains records that allow us to locate emp record ~~with a~~ query on salary.

Scanned by CamScanner

(8) We can use the term data entry to refer to the records stored in an index file.

A data entry with search key value K, denoted as K*, contains enough info to locate (one or more) data records with search key value K.



We can efficiently search an index to find the desired data entries, and then use these to obtain data records. (if these are distinct from the data entries)

(9) There are 3 main alternatives for what to store as a data entry ~~as a data entry~~ in an index

(1) A data entry K* is an actual data record. (with search key value k).

(2) A data entry is a (K, rid) pair, where rid is the record id of data record with search key value k

(3) A data entry is a (K, rid-list) pair, where rid-list is a list of record ids of data records with search key value K.

Of course, if the index is used to store actual data records, Alternative (1), each entry K* is a data record with search key value K.

Scanned by CamScanner

we can think of such an index as a special file organization. Such an indexed file organisation can be used instead of, for ex:- a sorted file or an unordered file of records.

Alternatives (2) and (3), which contain data entries that point to data records, are independent of the file organisation that is used for the indexed file (i.e, the file that contains the data record

Alternative (3) offers better space utilisation than Alternative (2), but data entries are variable in length, depending on the no! of data records with a given search key value.

(10) If we want to build more than one index on a collection of data records — for ex!- we want to build indexes on both the age and the sal fields for a collection of emp records — at most one of the index should be Alternative (1) because we should avoid storing data records multiple times

Clustered Indexes!—

(1) When a file is organised so that ordering of data records is the same as or close to the ordering of data entries in some index, we say that the index is clustered; otherwise, it is unclustered index.

Scanned by CamScanner

(2) An index that uses Alternative (1) is clustered by definition. An index that uses Alternative (2) (3) can be clustered index only if the data records are stored on the search key field. otherwise the order of the data records is random, defn purely by their physical order, and there is no reasonable way to arrange the data entries in same order.

(4) In practice, files are rarely kept sorted since this is too expensive to maintain when the data is updated. So in practice, a clustered index is an index that uses Alternative (1), and indexes that use Alternative (2) or (3) are unclustered.

(5) We sometimes refer to an index using Alternative (1) as a clustered file, because the data entries are actual data records, and the index is then file of data records. (As observed earlier, search and scans on an index return only its data entries even if it contains additional info to organise the data entries.)

(6) The cost of using an index to answer a range search query can vary tremendously based on whether the index is clustered.

If the index is clustered i.e, we are using the search key of a clustered file, the rid in qualifying data entries point to a contiguous

Scanned by CamScanner

collection of records, and we need to retrieve the only a few data pages.

If the index is unclustered, each qualifying data entry could contain a rid that points to a distinct data page, leading to as many data page I/O as the no of data entries that match the range selection, as illustrated in Fig. (below).

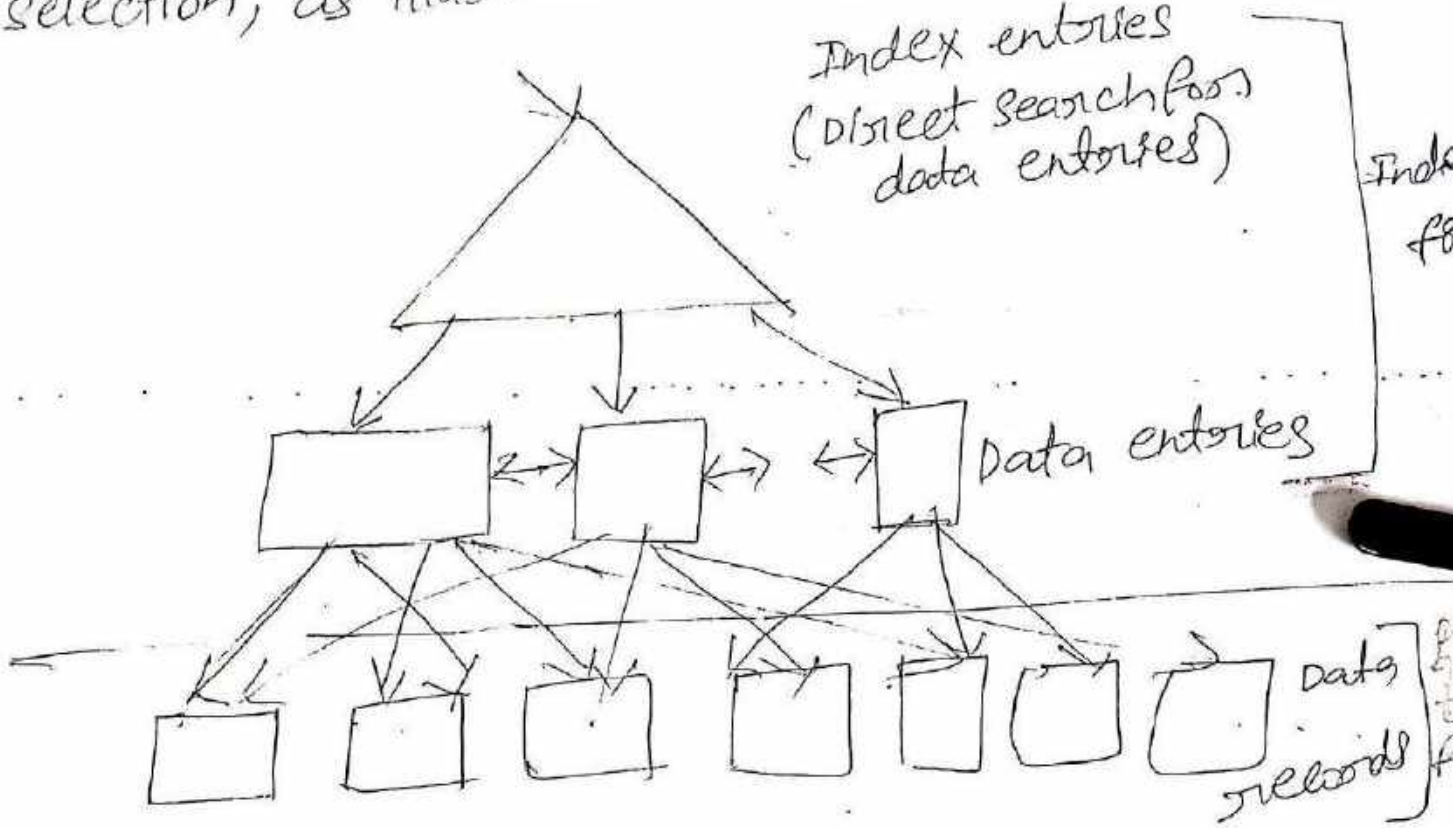


Fig 7 Unclustered Index using Alternative (2)

Primary and Secondary Indexes

(1) An index on a set of fields that includes the primary key is called a primary index; other indexes are called secondary indexes.

(The terms primary index and secondary index are sometimes used with different meaning.

(2) An index that uses Alternative (1) is called a primary index, and one that uses Alternatives (2) or (3) is a secondary index.

Scanned by CamScanner

(3) Two data entries are said to match if they have the same value for the search key field associated with the index.

(4) A primary index is guaranteed not to contain duplicates, but an index on other (collections of) fields can contain duplicates.

(5) In general, a secondary index contains duplicates.

(6) If we know that no duplicates exists, that is, we know that the search key contains some candidate key, we call the index a unique index.

(7) An important issue is how data entries in an index are organised to support efficient retrieval of data entries.

Index data structures:-

(1) One way to organise data entries is to hash data entries on the search key.

(2) Another way to organise data entries is to build a tree-like data structure that directs a search for data entries.

(3) NOTE: The choice of hash or tree indexing technique can be combined with any of 3 alternatives for data entries.

Scanned by CamScanner

(I) Hash-Based Indexing:-

(1) We can organize records using a technique call hashing to quickly find records that have a given search key value.

Ex:- If the file of employee records is hashed on the name field, we can retrieve all records about Jo

(2) In this approach, the records in a file are grouped in buckets, where a bucket consists of a primary page and, possibly, additional pages linked in a chain. The bucket to which a record belongs can be determined by applying a special function, called a hash function to the search key.

(3) Given a bucket number, a hash-based index structure allows us to retrieve the primary pag for the bucket in one or two disk I/Os.

(4) On inserts, the record is inserted into appropriate bucket, with 'overflow' pages allocated as necessary.

To search for a record with a given search key value, we apply the hash function to identify the bucket to which such records belong and look at all pages in that bucket.

If we do not have the search key values for th record, Ex:- The index is based on sal and we wan records with a given age value, we have to scan all pages in the file.

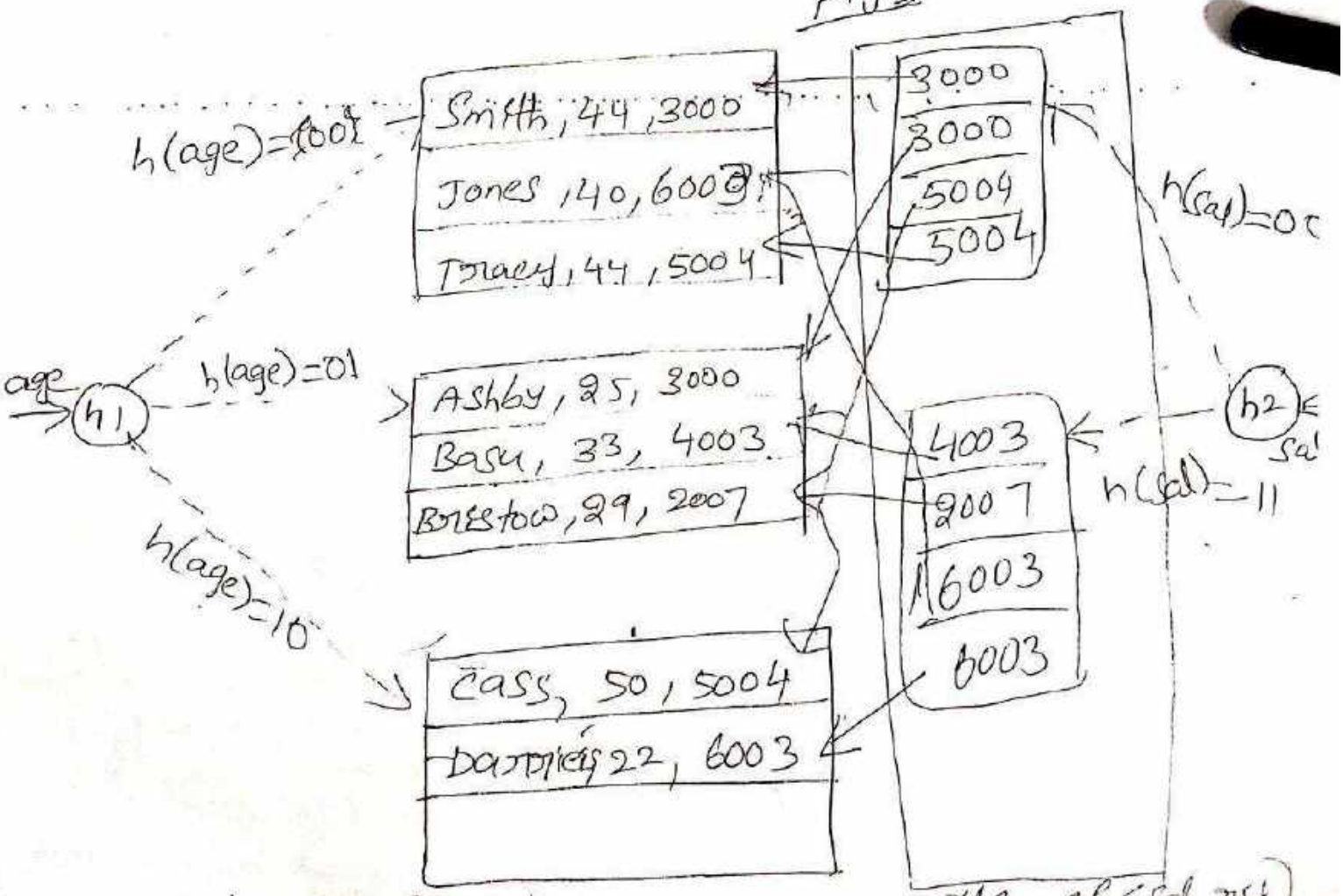
Scanned by CamScanner

(5) Assumption: Applying the hash function to (the search key of) a record allows us to identify and retrieve the page containing the record with one I/O.

In practice, hash based index structures that adjust gracefully to inserts and deletes and allow us to retrieve the page containing a record in one to two I/Os are known.

(6) Hash indexing is illustrated in Fig (below).

Fig 2



Index-organised file hashed on age, with Auxiliary Indexes on sal

Scanned by CamScanner

where the data is stored in a file that is labeled on age; the data entries in this 1st index file are the actual data records.

Applying the hash function to the age field identifies the page that the record belongs to.

The hash function h for this ex: is quite simple it converts the search key value to its binary representation and uses the two least significant bits as the bucket identifier.

Ex: 44 → 101100
40 → 101000
25 → 11001
29 → 11101

(7) The above Fig. also shows that an index with search key Sal that contains (Sal, rid) pairs as data entries.

The rid (short for record id) component of data entry in this second index is a pointer to a record with search key value Sal (and is shown in Fig. as an arrow pointing to the data record).

(8) The file of employee records is hashed on age, and Alternative (1) is used for data entries.

The 2nd index, on Sal, also uses hashing to locate data entries, which are now (Sal, rid of emp record) pairs; that is Alternative (2) is used for data entries.

(9) NOTE: The search key for an index can be any sequence of one or more fields and it need not

Scanned by CamScanner

uniquely identify records.

Ex:- In the Salary Index, two data entries have the same search key value 6003. (There is an unfortunate overloading of the term key in the database literature. A primary key or candidate key - fields that uniquely identify a record)

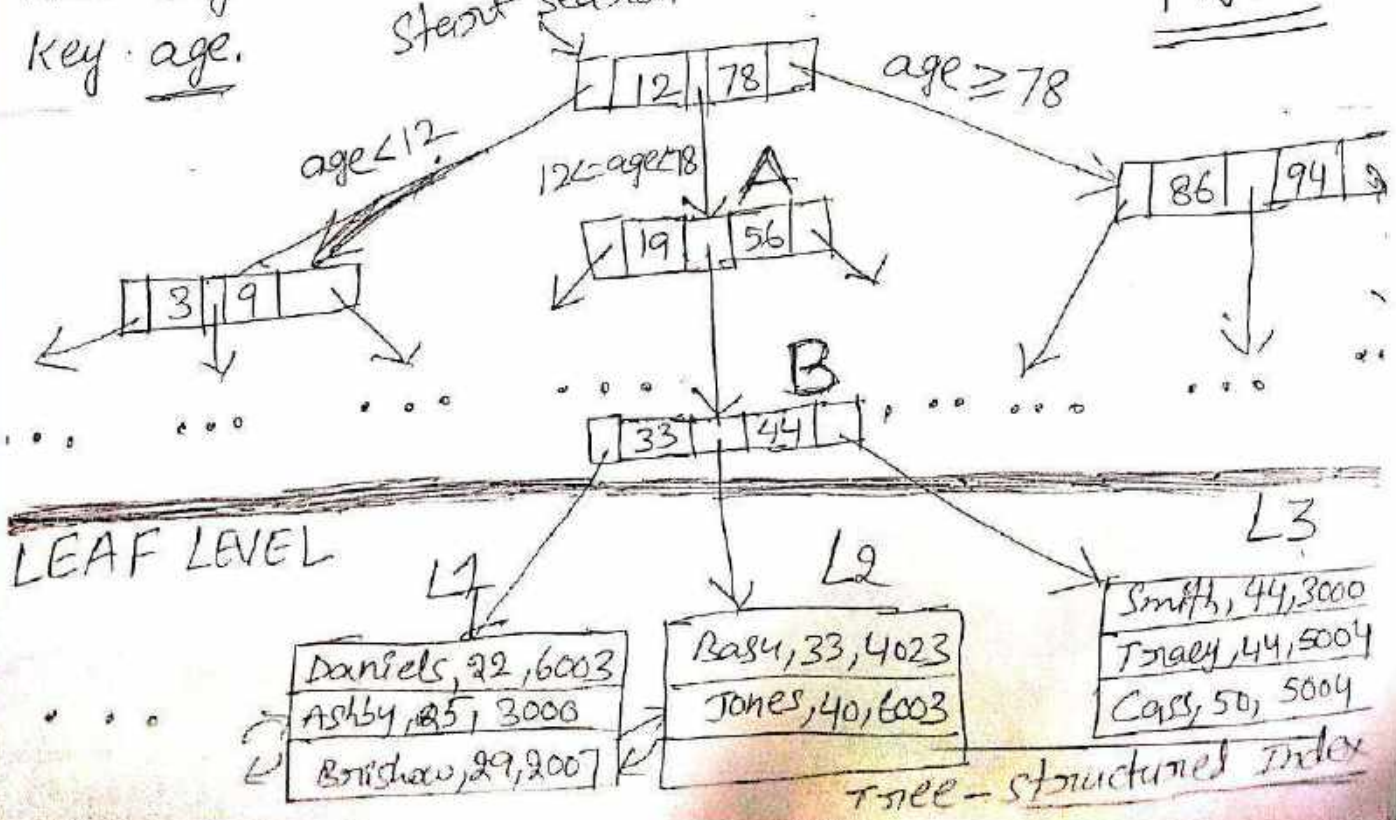
Tree-Based Indexing:-

(1) An alternative to hash-based indexing is to organize records using a tree-like data structure.

The data entries are arranged in sorted order by search key value, and a hierarchical search data structure is maintained that directs searches to the correct page of data entries.

(2) Fig 3 shows the employee records from Fig 2, this time organised in a tree-structured index with search key age.

Fig 3



Scanned by CamScanner

each node in this fig. (e.g., nodes labeled A, B, L1, L2) is a physical page, and retrieving a node involves a disk I/O.

(3) The lowest level of tree, called the leaf level contains the data entries; in our ex:, these are employee records.

To illustrate the ideas better, we have drawn Fig 3 as if there were additional employee records, some with age less than 22 and some with age greater than 50 (the lowest and highest age values that appear in Fig 2.).

Additional records with age less than 22 would appear in leaf pages to the left page L1, and records with age greater than 50 would appear in leaf pages to the right of page L3.

(4) The structure allows us to efficiently locate all data entries with search key values in a desired range.

All searches begin at the topmost node, called the root, and the contents of pages in non-leaf levels direct searches to the correct leaf page.

Non-leaf pages contain node pointers separated by search key values.

✓ The node pointer to the left of a key value K points to a subtree that contains only data entries less than K, ✓ the node pointer to the key value K points to a that contains only

Scanned by CamScanner

(5) In our example, suppose we want to find all data entries with $24 < age < 50$.

→ Each edge from the root node to a child node in Fig 2 has a label that explains what the corresponding subtree contains. (Although the labels for the remaining edges in the Fig. are not shown, they should be easy to deduce.)

→ In our ex: search, we look for data entries with search key value > 24, and get directed to the middle child, node A. Again, examining the contents of this node, we are directed to node B. Examining the contents of node B, we are directed to leaf node L' which contains data entries we are looking for.

(6) Observe that leaf nodes L2 and L3 also contain data entries that satisfy our search criterion.

To facilitate retrieval of such qualifying entries during search, all leaf pages are maintained in a doubly-linked list. Thus we can fetch page L2 using the 'next' pointer on page L1, and then fetch page L3 using the 'next' pointer on L2.

(7) Thus, the no' of disk IOs incurred during a search is equal to the length of a path from the root to a leaf, plus (+) the no' of leaf pages with qualifying data entries.

The B+ tree is an index structure that ensures that all paths from the root to leaf in a given tree are of the same length.

Scanned by CamScanner

that is, the structure is always balanced in height.

→ Finding the current leaf page is faster than binary search of the pages in a sorted file because each non-leaf node can accommodate a very large no' of node-pointers, and the height of the tree is rarely more than three or four in practice.

→ The height of a balanced tree is a length of a path from root to leaf; in Fig 3 the height is 2

→ The no' of I/Os to retrieve a desired leaf page is 4, including the root and leaf page. (In practice, the root is typically in the buffer pool because it is frequently accessed, and we really incur just three I/Os for a tree of height three.)

⑧ The average no' of children for a non-leaf node is called the fan-out of the tree.

→ If every non-leave node has n children, a tree of height h has n^h leaf pages.

→ In practice, nodes do not have the same no' of children but using the average value F for n, we still get a good approximation to the no' of leaf pages, F^h.

→ In practice, F is at least 100, which means a tree of height four contains 100 million leaf pages.

→ Thus, we can search a file with 100 million leaf pages and the page we want using four I/O's; in contrast binary search of the same file would take $\log_2 100,000,000$ (over 25) I/Os.

Scanned by CamScanner

COMPARISION OF FILE ORGANIZATIONS :-

(1) We now compare the costs of simple operations for basic file organisations on a collection of employee records. (2) Assumption: The files and indexes are organized according to composite search key (age, sal and that all selection operations are specified on these fields. (3) The organisations that we consider are the following:

(1) File of randomly ordered employee records, or heap file. (2) File of employee records sorted on <age, sa (3) Clustered B+ tree file with search key <age, sal> (4) Heap file with an unclustered B+ tree index on <age, sal>.

(5) Heap file with an unclustered hash index on <age, sa

~~(6) Heap file with an unclustered hash~~

(6) GOAL:- The IMP of the choice of an appropriate file organisation, and the above list includes the main alternatives to consider in practice.

Obviously, we can keep the records unsorted or sort them. We can also choose to build an index on the data file Note that even if the data file is sorted, an index whose search key differs from the sort order behaves like an index on a heap file!

The operations we consider are these:

(1) Scan:- Fetch all records in the file. The pages in the file must be fetched from disk into the buffer pool. There is also a CPU overhead per record for locating the record on the page (in the

Scanned by CamScanner

(2) Search with Equality Selection:— Fetch all records that satisfy an equality condition; Ex:— "Find the employee record for the employee with age 23 a sal 30". pages that contain qualifying records must be fetched from disk, and qualifying records must be located within retrieved pages.

(3) Search with Range Selection:— Fetch all records that satisfy a range selection. Ex:— "Find all employee records with age greater than 35."

(4) Insert a Record:— Insert a given record into th file. we must identify the ~~record page into~~ page in th file into which the new record must be inserted, fetch th page from disk, modify it to include the new record, a then write back the modified page.

Depending on the file organisation, we may have t fetch, modify, and write back other pages as well.

(5) Delete a Record:— Delete a record that is spec fied using its rid. we must identify the page that contains the record, fetch it from disk, modify it and write it back. Depending on the file organisati coe may have to fetch, modify, and write back othe pages as well.

Scanned by CamScanner

(I) COST Model:-

(1) Cost model that allows us to estimate the cost (in terms of execution time) of different DB operations. we use

B: to denote the no' of data pages when records are packed onto pages with no wasted space, and

R: to denote the no' of records per page

D: The average time to read or write disk page or

C: The average time to process a record (e.g, to compare a field value to a selection constant)

H: In the hashed file organisation, we use a function, called a hash function, to map a record into a range of numbers. the time required to apply the hash function to a record.

F: For tree indexes, we will use F to denote the fan-out which typically is at least 100.

(2) Typical values today are D = 15 milliseconds, C & H: 100 nanoseconds; we therefore expect the cost of I/O to dominate. I/O is often (even typically) the dominant component of the cost of database operations, and considering I/O costs gives us a good ,st approximation to the true costs.

(3) Further, CPU speeds are steadily rising, whereas disk speeds are not increasing at a significant pace. (on the other hand, as main memory sizes increases, a much

Scanned by CamScanner

largest fraction of the needed pages are likely to be in memory, leading to fewer I/O requests!).

(4) We have chosen to concentrate on the I/O component of cost model, and we assume the simple constant C for in-memory per-record processing cost.

(5) Bear the following observations in mind:

→ Real systems must consider other aspects of cost such as CPU costs (and n/w transmission costs in a distributed database).

→ Even with our decision to focus on I/O costs, an accurate model would be too complex for our purposes of conveying the essential issues in a simple way...

→ We therefore use a simplistic model in which we just count the no' of pages read from or written to disk as a measure of I/O.

→ We ignore the IMP issue blocked access in our analy typically disk systems allow us to read a block of contig pages in a single I/O request.

→ The cost is equal to the time required to seek the f page in the block and transfer all pages in the block. Such blocked can be much cheaper than issuing one I/ request per page in the block, especially if these reque do not follow consecutively, because we would have an additional seek cost for each page in the block.

Scanned by CamScanner

Heap Files:-

Scan: The cost is $B(D+RC)$ because we must retrieve each of B pages time D per page, and for each page, process R records taking time C.

Search with Equality Selection: Suppose that we know in advance that exactly one record matches the desired equality selection, i.e, the selection is specified on a candidate key.

On average, we must scan half the file, assuming that the record exists & the distribution of values in the search field is uniform.

For each retrieved data page, we must check all records on the page to see if it is the desired record.

The cost is $0.5B(D+RC)$

If no record satisfies the selection, however, we must scan the entire file to verify this.

If the selection is not a candidate key field (Ex: "Find employees aged 18") we always have to scan the entire file because records with age=18 could be dispersed all over the file, and we have no idea how many such records exist.

Search with Range Selection:- The entire file must be scanned because qualifying records could appear anywhere in the file, and we do not know how many qualifying records exist. The cost is $B(D+RC)$.

Scanned by CamScanner

Insert: we assume that records are always inserted at the end of the file. we must fetch the last page in the file, add the record, write the page back.
The cost is 2D+C.

Delete: we must find the record, remove the record from the page, and write the modified page back. we assume that no attempt is made to ~~often~~ compact the file to reclaim the free space created by deletion for simplicity! The cost is the cost of searching + C+D.

we assume that the record to be deleted is specified using the record id. Since the page id can easily be obtained from the record id, we can directly read in the page. The cost of searching is therefore D.

If the record to be deleted is specified using an equality or range condition on some fields, the cost of searching is $0.5B(D+RC)$, $B(D+RC)$ respectively.

The cost of deletion is also affected by the no' of qualifying records, since all pages containing such records must be modified.

Sorted Files:-

Scan:- The cost is $B(D+RC)$ because all pages must be examined. Note that this case is no better or worse than the case of unordered files. However, the ordering in which records are retrieved corresponds to the [illegible] all records in age order, and for a given

Search with Equality selection:-

(1) We assume that the equality selection matches the sort order (age, sal).

In other words, we assume that a selection condn is specified on at least the 1st field in the composite key (Ex:- age=30). If not (Ex:-, selection sal=50 or dept= the sort order does not help us and the cost is identical to that for a heap file.

(2) we can locate the 1st page containing the desired record or records, should any qualifying records exist, with a binary search in $\log_2 B$ steps. (This analysis assumes that the pages in the sorted file are stored sequentially, and we can retrieve the ith pg. on the file directly in one disk I/O.)

Each step requires a disk I/O and 2 comparisions. once the page is known, the 1st qualifying record can again be located by a binary search of the page at a cost of $C \log_2 R$.

The cost is $D \log_2 B + C \log_2 R$, which is significant improvement over searching heap files.

(3) If several records qualify (Ex:- "Find all employees aged 18"), they are guaranteed to be adjacent to each other due to the sorting on age, and so the cost of retrieving all such records is the cost of locating ~~the 1st such record~~ ($D \log_2 B + C \log_2 R$) plus the cost of reading all the qualifying records in sequential order.

Scanned by CamScanner

Typically, all qualifying records fit on a single pa
If no records qualify, this is established by the
search for the 1st qualifying record, which finds the
page that would have contained a qualifying record, h
one existed and searches that page.

Search with Range Selection:-

① Again assuming that the range selection matche
the composite key, the 1st record that satisfies the
selection is located as for search with equality.
Subsequently data pages are sequentially retrie
until a record is found that does not satisfy the range
selection; this is similar to an equality search with
many qualifying records.

② The cost is the cost of search plus the cost of
retrieving the set of records that satisfy the searc

The cost of search includes the cost of fetch
the 1st page containing qualifying, or matching reco
For small range selections all qualifying record
appears on this page.
For large selections, we have to fetch addition
pages containing matching records.

Insert:- To insert a record while preserv
the sort order, we must 1st find the correct positio
in the file, add the record, and then fetch and record
all subsequent pages (because all the old records are
by one slot, assuming that the
no empty slot

Scanned by CamScanner

On average, we can assume that the inserted record belongs in the middle of the file. Therefore we must read the latter half of the file and then write it back after adding the new record. The cost is that of searching to find the position of the new record plus 2·(0.5B(D+BC)), that is, search cost plus B(D+RC).

Delete: ① we must search for the record, remove the record from the page, and write the modified page back. we must also read and write all subsequent pages because all records that follow the deleted record must be moved up to compact the free space.

The cost is same as for an insert, that is, search cost plus B(D+RC).

Given the rid of the record to delete, we can fetch page containing the record directly.

② If records to be deleted are specified by an equality or range condition, the cost of deletion depends on the no' of qualifying records.

If the condition is specified on the sort field, qualifying records are guaranteed to be contiguous and the 1st qualifying record can be located using binary search.

Scanned by CamScanner

Clustered Files:-

In a clustered file, extensive empirical study has shown that pages are usually at about 67 percent occupancy. Thus, the no' of physical data pages is about 1.5B, and we use this observation in the following analysis.

SCAN:- The cost of a scan is 1.5B(D+RC) because all data pages must be examined; this is similar to the Sorted files, with the obvious adjustment for the increased no' of data pages.

Note that our cost metric does not capture potential differences in cost due to the sequential I/O regard, although a clustered file using ISAM(rather than B+ trees) could be close.

Search with Equality Selection:-

we assume that the equality selection matches the search key <age, sal>.

we can locate the 1st page containing the desired record or records, should any qualifying records exist, in $\log_F 1.5B$ steps, that is, by fetching all pages from the root to the appropriate leaf.

In practice, the root page is likely to be in the buffer pool and we save in I/O, but we ignore this in our simplified analysis.

Scanned by CamScanner

Each step requires a disk I/O and a comparison.
Once the page is known, the 1st qualifying record can again be located by a binary search of the page at a cost of $C \log_2 R$.

The cost is $D \log_F 1.5B + C \log_2 R$, which is a significant improvement over searching even sorted files.

(2) If several records qualify (@Ex) – "Find all employees aged 18"), they are guaranteed to be adjacent to each other due to the sorting on age, and so the cost of retrieving all such records is the cost of locating the 1st such record ($D \log_F 1.5B + C \log_2 R$) plus the cost of reading all the qualifying records in sequential order.

Search with Range Selection:-

Again assuming that the range selection matches the composite key, the 1st record that satisfies the selection is located as it is for search with equality.

Subsequently, data pages are sequentially retrieved (using the next & previous links at the leaf level) until a record is found that does not satisfy the range selection; this is similar to an equality search with many qualifying records.

Insert:- To insert a record, we must 1st find the correct leaf page in the index, reading every page from root to leaf. Then, we must add

Scanned by CamScanner

new record.

most of the time, the leaf page has sufficient space for the new record, and all we need to do is to write out the modified leaf page.

Occasionally, the leaf is full and we need to retrieve and modify other pages, but this sufficiently rare that we can ignore it in this simplified analysis. The cost is therefore the cost of search plus one write, $D\log_F 1.5B + C\log_2 R + D$.

Delete:- we must search for the record, remove the record from the page, and write the modified page back. The discussion and cost analysis for insert applies here as well.

Heap File with Unclustered Tree Index

① The no' of leaf pages in an index depends on the size of data entry.

we assume that each data entry in the index is a 10th the size of an employee data record, which is typical.

The no' of leaf pages in the index is $0.1(1.5B) = 0.15B$. if we take into account the 67 percent occupancy of index pages. Similarly, the number of data entries on a page $10(0.67R) = 6.7R$, taking into a/c the relative size and occupancy.

Scan:- Consider Fig 1, which illustrates an unclustered index. To do a full scan of the file of employee records scan the leaf level of the index and

Scanned by CamScanner

Hash Based Indexing (1)

(1) In this Hash-based Indexing we consider file organisation that are excellent for equality selection

(2) The basic idea is to use a hashing function, which maps values in a search field into range of bucket numbers to find the page on which a desired data entry belongs.

3) We use a simple scheme called Static Hashing to introduce ~~data~~ idea.

This scheme like I.S.A.M, suffers from the problem of long overflow chains, which can affect performance.

4) Two solutions to the problem are presented

- Extendible hashing
- Linear hashing.

5) Extendible hashing scheme uses a directory to support insert & delete efficiently with no overflow pages.

6) Linear Hashing scheme uses a clever policy for creating new buckets and supports inserts and deletes efficiently without use of a directory

Scanned by CamScanner

Although overflow pages are used, the length of overflow chains is rarely more than two.

(7) Hash-based indexing techniques cannot support range searches, unfortunately.

(8) Tree-based indexing techniques, can support range searches efficiently and are almost a good as hash based indexing for equality select

(9) Thus, many commercial systems choose to suppa only tree based Indexes.

(10) Nonetheless, hashing techniques prove to be very useful in implementing relation operations such a joins. (11) In particular, the Index Nested Loops join methods generates many equality selection queries, and the difference in cost b/w a hash-based index and a tree-based index can become significant in this context.

Static hashing:- (1) Static hashing is illustrated in Fig 1 (below)

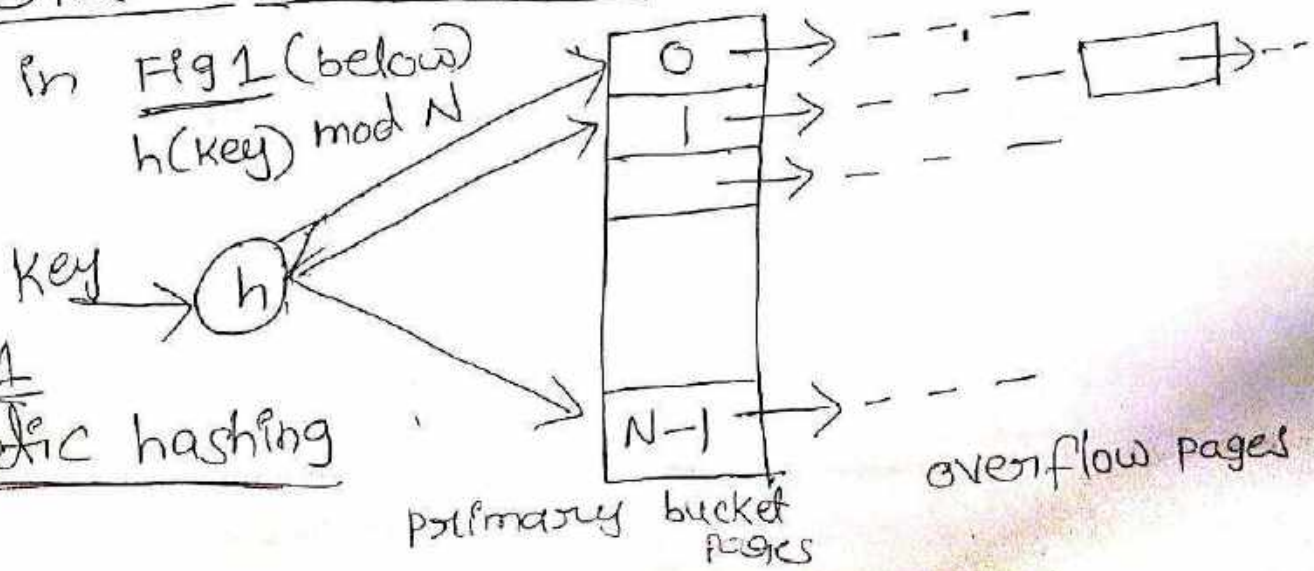


Fig 1
Static hashing

Scanned by CamScanner

The pages containing the data can be viewed as a collection of buckets, with one primary page and possibly additional overflow pages per bucket.

A file consists of buckets 0 through N-1, with one primary page per bucket initially.

Buckets contain data entries, which can be any of the three alternatives.

To search for data entry, we apply a hash function to identify the bucket to which it belongs and then search this bucket.

To speed up search of bucket, we can maintain data entries in the sorted order by search key value; (1) we do not sort entries, and the order of entries within a bucket has no significance.

To insert a data entry, we use the hash function to identify the correct bucket and then put the data entry there.

If there is no space for this data entry, we allocate a new overflow page, put the data entry on this page, and add the page to overflow chain of the bucket.

To delete a data entry, we use the hashing function to identify the correct bucket, locate the data entry by searching the bucket, and then remove it.

If the data entry in the overflow page, the overflow page is removed from the overflow

Scanned by CamScanner

chain of the bucket and added to a list of ov pages.

(9) The hash function is an IMP component of the hashing approach. (10) It must distribute values in the domain of the search field uniformly over the collection of buckets.

(11) If we have N buckets, numbered 0 through N-1, a hash function h of the form h(value) = (a * value + b) works well in practice. (The bucket identified is h(value) mod N.) The constants a and b can be chosen to (time) the hash function.

(12) Since the no of buckets in a static Hashing file is known when the file is created, the primary pages can be stored on successive disk pages.

(13) Hence, a search ideally requires just one disk I/O and insert and delete operations require two IOs (read and write the page), although the cost could be higher in the presence of overflow pages.

(14) As the file grows, long overflow chains can develop Since searching a bucket requires us to search (in general) all pages in its easy to see how performance can deteriorate.

(15) By initially keeping pages 80 percent full, we can avoid overflow pages if the file does not grow too much, but in general the only way to get over chains is to create a new file buckets.

Scanned by CamScanner

b) the main ...
but the no of buckets fixed.
(17) If a file shrinks greatly, a lot of space is wasted; more IMP, if a file grows a lot, long overflow chains develop, resulting in poor performance. Therefore, static hashing can be compared to ISAM structure, which can also develop long overflow chains in case of insertions to the same leaf. (18) Static hashing also has the same advantage as ISAM w.r.t concurrent access -

1) one simple alternative to static hashing is to periodically 'rehash' the file to restore the ideal situation (no overflow chains, about 80 percent occupancy). (20) However, rehashing takes time and the index cannot be used while rehashing is in progress.
) Another alternative is to use dynamic hashing techniques such as Extendible and Linear hashing, which deal with inserts and deletes gracefully.
Notation and Conventions:- Record with search key k, we denote the index data entry by k*, For hash-based indexes, the 1st step in searching for, inserting or deleting a data entry with search key k is to apply a hash function h to k.

Scanned by CamScanner

h(K) and the value h(K) identifies the bucket for the data entry K.

Note that a diff. search keys have the same hash value.

Extendible Hashing:-

① To understand extendible Hashing let us begin by considering a static Hashing file.

② If we have to insert a new data entry into a full bucket, we need to add an overflow page.

③ If we do not want to add overflow pages, one solution is reorganise the file at this point by doubling the no of buckets and redistributing the entries across the new set of buckets.

④ This solution suffers from one major defect - the entire file has to be read, and twice as many page have to be written to achieve the reorganisation.

⑤ This problem, however, can be overcome by a simpl idea: Use a directory of pointers to buckets. and double the size of the no of buckets by doubling just the directory and splitting only the the bucket that overflowed.

⑥ To understand the idea, consider the sample file shown in Fig 2.

Scanned by CamScanner

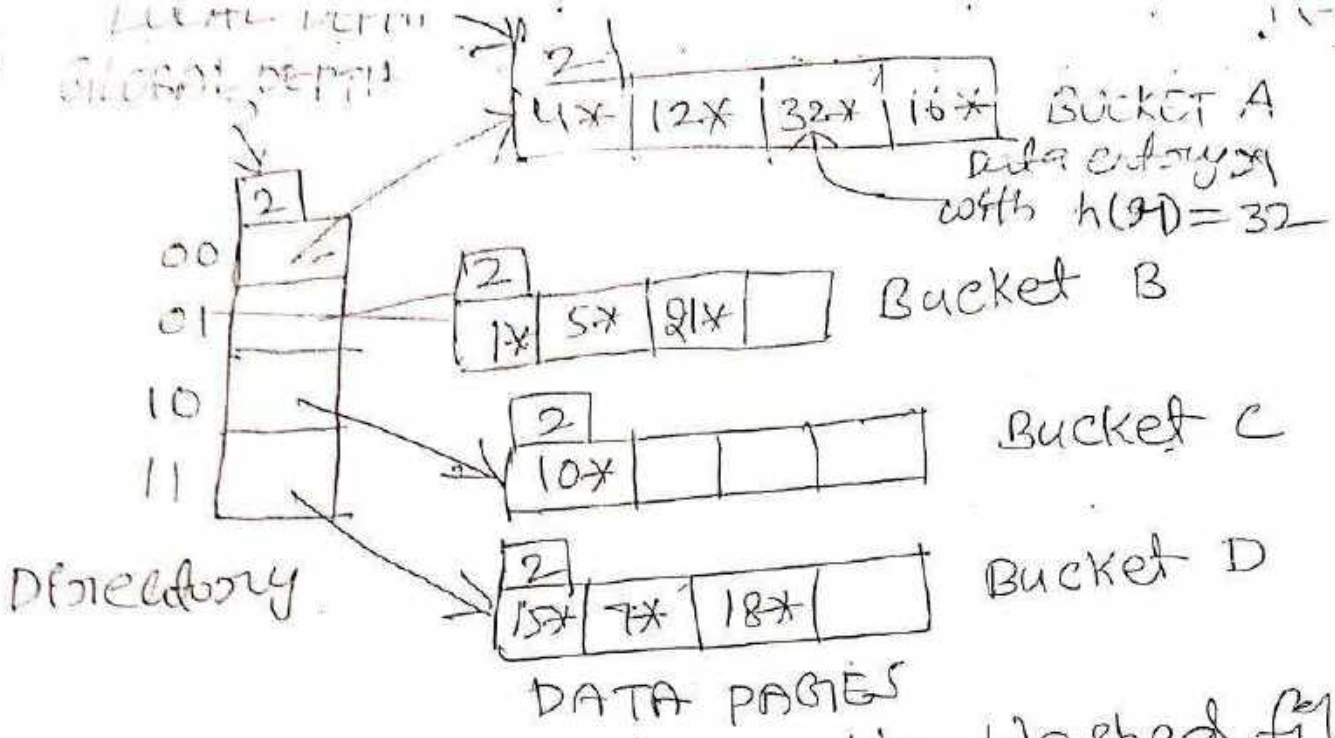


Fig 2 Example of Extendible Hashed file

1) The directory consists of an array of size 4, with each element being a pointer to a bucket.
The global depth & local depth discussed shortly ignore them for now
2) To locate a data entry, we apply a ~~Short~~ hash function to the search field and take the last 2 bits of its binary representation to get a number b/w 0 and 3. The pointer in this array position gives us the desired bucket. (10) We assume that each bucket can hold 4 data entries. (11) Therefore, to locate a data entry with hash value 5 (binary 101), we look at directory element 01 and follow the pointer to the data page bucket B in the figure.)

Scanned by CamScanner

(12) To insert a data entry, we search to find the appropriate bucket. 13 = 1 1 0 1

Ex - To insert data entry with hash value 13 (denoted as 13*), we examine directory element 01 and go to the page containing data entries 1*, 3* and 21*. Since the page has space for an additional data entry, we are done after we insert the entry (Fig 3).

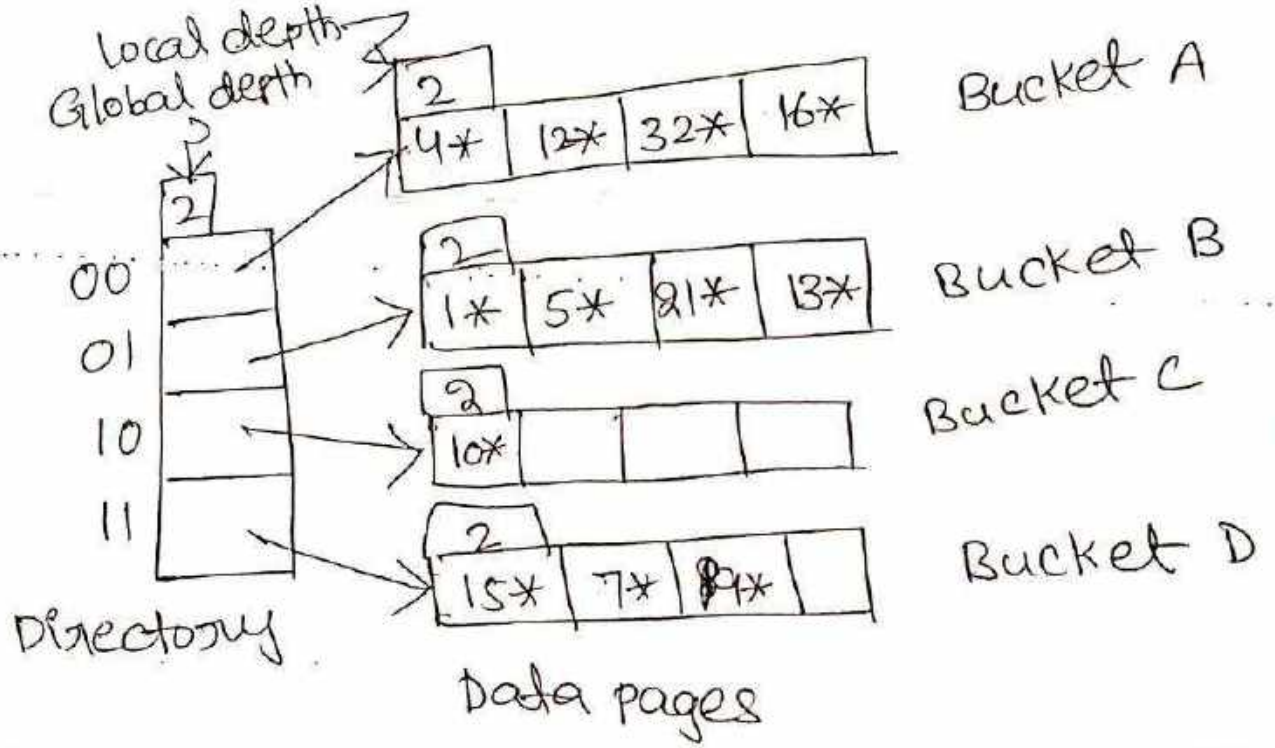


Fig 3 After Inserting Entry r with h(r) = 13

(13) Next, let us consider insertion of a data entry into a full bucket.

(14) The essence of the Extendible Hashing idea lies in how we deal with this case.

Scanned by CamScanner

5) Consider the insertion of data entry 20*.
(binary 10100).
6) Looking at directory element 00, we are led to bucket A, which is already full.
7) We must first split the bucket by allocating a new bucket and redistributing the contents (including the new entry to be inserted) across the old bucket and its 'split image'.

8) To redistribute entries across the old bucket and its split image, we consider the last 3 bits of h(r); the two bits are 00, indicating a data entry that belongs to one of these two buckets, and the 3rd bit discriminates between these buckets. The redistribution of entries is illustrated in Fig. 4

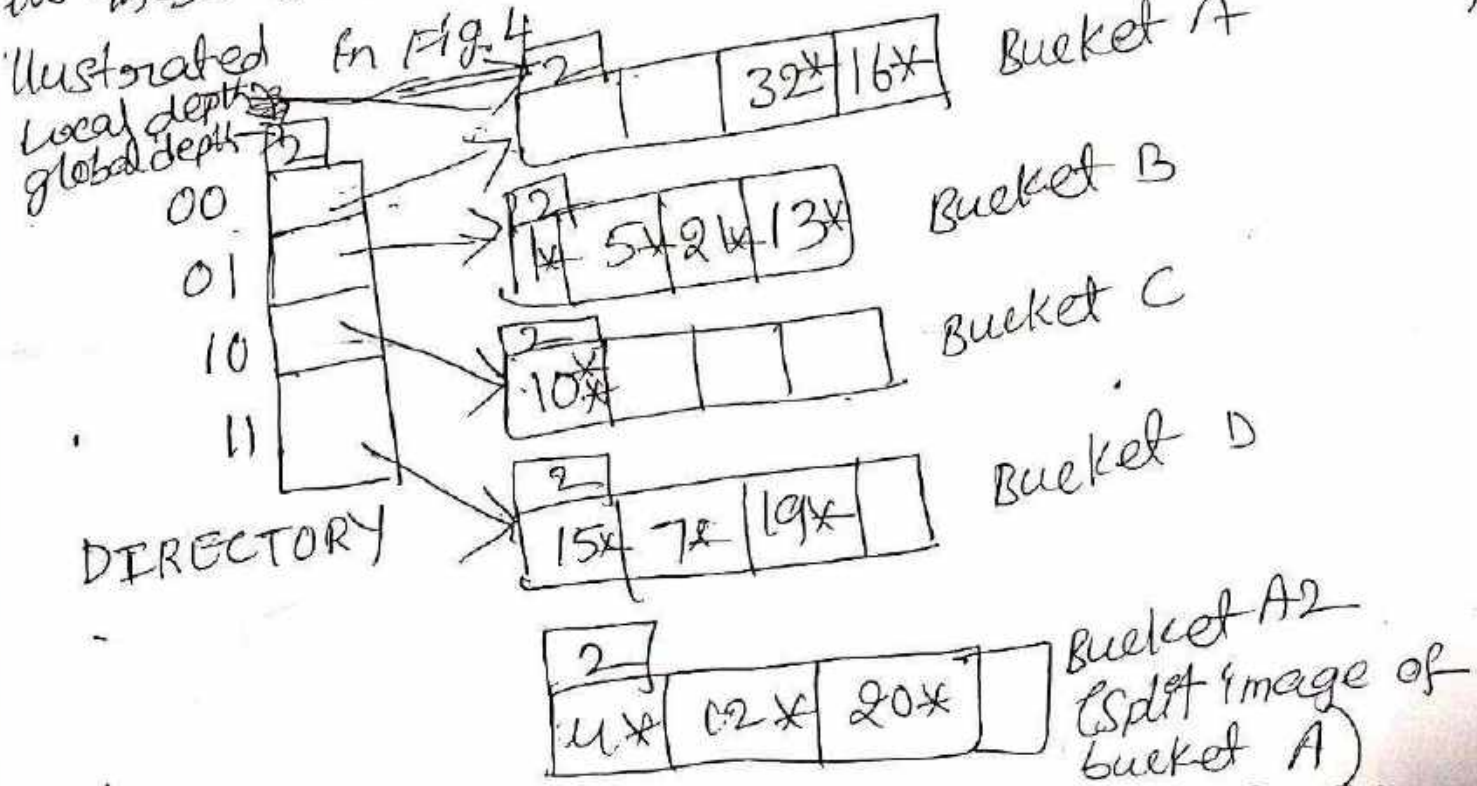


Fig 4 While Inserting Entry 5 with h(r)=20.

Scanned by CamScanner

(19) Note a problem that must now resolve — we need 3 bits to discriminate b/w two of our data pages (A and A2), but the directory has only enough slots to store all two-bit patterns.

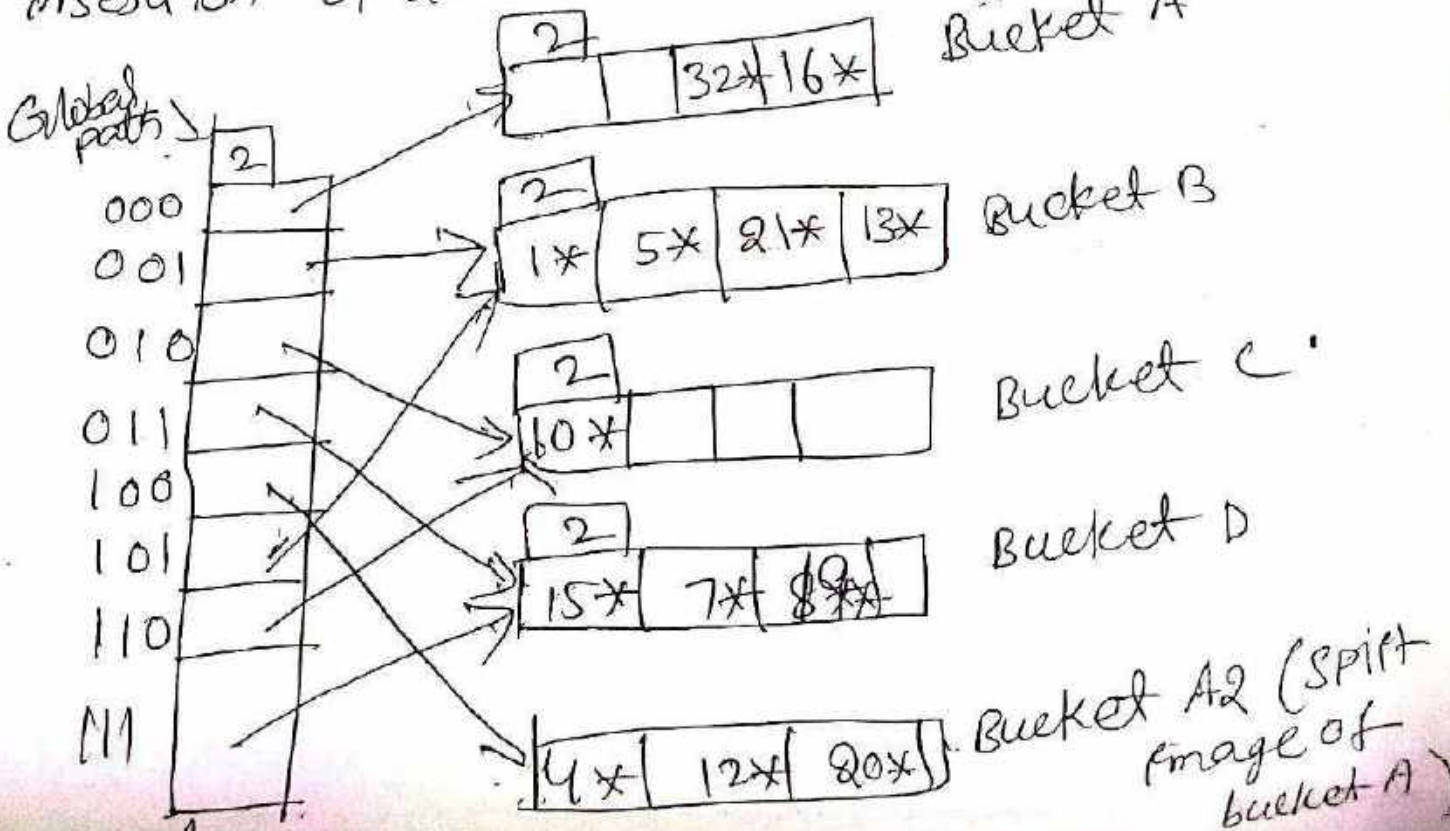
(20) The solution is to double the directory.

(21) Elements that differ only in the 3rd bit from the end are said to be 'corresponding'.

(22) Corresponding elements of the directory point to the same bucket with the exception of the elements corresponding to the split bucket.

(23) In our ex:- , bucket A was split; so new directory element 000 points to one of the split versions and new element 100 points to the other.

(24) The sample file after completing all steps in the insertion of 20* is shown in Fig 5.



Scanned by CamScanner

a new bucket page, writing both this page and the old bucket page that is being split, and doubling the directory array.

(26) The directory is likely to be much smaller than the file itself because each element is just a page-id, and can be doubled by ~~directory array~~ simply copying it over (and adjusting the elements for the split buckets). The cost of doubling is now quite acceptable.

(27) We observe that basic technique used in Extendible Hashing is to treat the result of applying a hash function h as a binary number and interpret the last d bits, where d depends on the size of the directory, as an offset into the directory. (28) In our example, d is originally 2 because we only have 4 buckets.

(29) A corollary, logical conclusion, is that when distributing entries across a bucket and its split image. (30) we should do so on the basis of the dth bit. (Note how entries are redistributed in our example; see Fig 5) (31) The number d called the global depth of the hashed file, is kept as part of the header of the file. (32) It is used every time we used to locate a data entry.

Scanned by CamScanner

(33) An insert point that causes [illegible] bucket necessitates a directory doubling.

(34) Consider our example, as shown in Fig 5.

(35) If we now insert 9*, it belongs in bucket B; this bucket is already full. We can deal with this situation by splitting the bucket and using directory element 001 and 101 to point to the bucket and its split image, as shown in Fig 6. (below)

001 = 1001

101

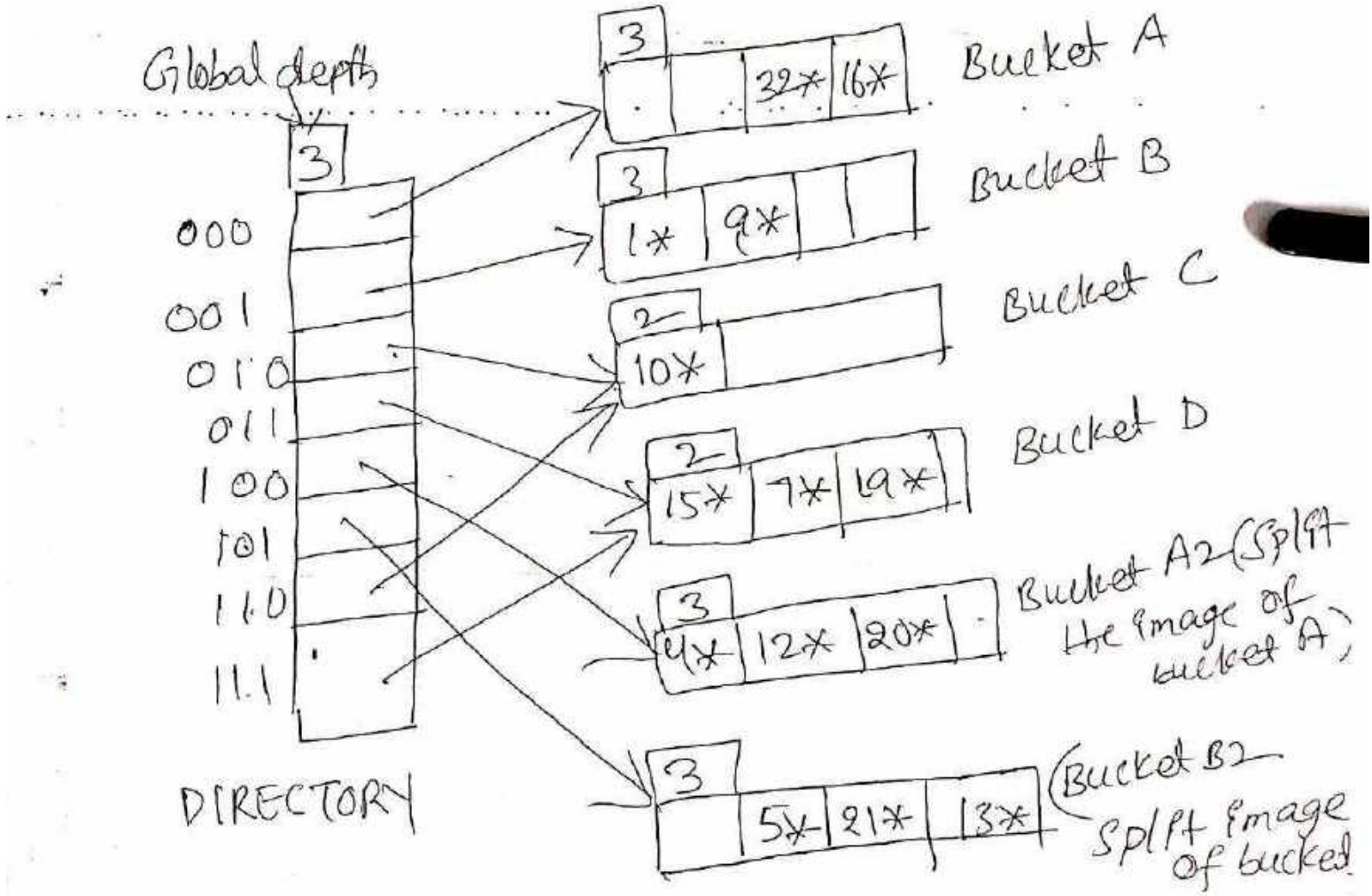


Fig 6 After inserting Entry r with h(r) = 9

36) Hence, a bucket split does not necessarily require a directory doubling. However, if either bucket A or A2 grows full and an insert then forces a bucket split, we are forced to double the directory again.

37) To differentiate b/w these cases and determine whether a directory doubling is needed, we maintain a local ~~path~~ depth for each bucket.

38) If a bucket whose local depth is equal to the global depth is split, the directory must be doubled. 39) Going back to the example, when we inserted 9* into the index shown in Fig 5, it belonging to bucket B with local depth 2, whereas the global depth was 3. 40) Even though the bucket was split, the directory did not doubled.

41) Buckets A and A2, on other hand, have local depth equal to the global depth, and, if they grow full and split, the directory must then be doubled. 42) Initially, all local depths are equal to the global depth (which is number of bits needed to express the total no of buckets).

43) We increment the global depth by 1 each time the directory doubles, of course, a

Scanned by CamScanner

(44) If not, split leads to a directory doubling, we increment by 1 the local depth of split bucket and assign this same (incremented) local depth to its (newly created) split image.

(45) Intuitively, if a bucket has local depth l, the hash values of data entries in it agree on the last l bits, further, no data entry in any other buckets of the file has a hash value with the same last l bits.

(46) A total of 2^{d+l} directory elements point to a bucket with local depth l. If $d=l$, exactly one directory element points to the bucket and splitting such a bucket requires directory doubling.

$2^{3-2} = 2$

$2^{3-3} = 1$

(47) A final point to note is that we can also use the 1st d bits (the most significant bits) instead of the last d (least significant bits), but in practice the last of d-bits are used the reason is that a directory can then be doubled simply by copying it.

(48) In summary, a data entry can be located by computing its hash value, taking the last d bits and looking in the bucket pointed to by this directory element.

Scanned by CamScanner

bucket to which it belongs and the bucket is split if necessary to make space.

50) A bucket split leads to increase in the local depth and, if the local depth becomes greater than the global depth as a result, to a directory doubling and in increase in the global depth) as well.

1) For deletes the data entry is located and removed. 52) If the delete leaves the bucket empty, it can be merged with its split image, although this step is often omitted in practice.

53) merging buckets decreases the local depth

54) If each directory element points to the same bucket as its split image (i.e, 0 and 2^{d-l} point to the same bucket, namely, A: 1 and $2^{d-l}+1$ point to same bucket, initially, B, which may or may not be identical to A; etc), we can halve the directory and reduce the global depth, although this step is not necessary for correctness.

55) The insertion examples can be worked out backwards as examples of deletion. (start with the structure shown after an insertion and delete the inserted elements) In each case the original structure should be restored.

Scanned by CamScanner

(56) If the directory [illegible] equality selection can be answered in a single disk acce as for Static Hashing (in the absence of overflow pages), but otherwise, two disk I/O's are needed.

(57) As a typical example, a 100MB file with 100b per data entry and a page size of 4KB contains 1 million data entries and only about 25,000 elements in the directory. (58) (Each page/bucket contains roughly 40 data entries, and we have one directory element per bucket). (59) Thus, although equality selections can be twice as slow for static hashing files, chances are high that the directory will fit in memory and performance is the same as for static hashing files. (60) On the other hand, the directory grows in spurts and can become large for skewed data distributions (where our assumption that the data pages contain roughly equal no' of data entries is not valid). (61) In the context of hashed files, In a skewed (suddenly change) data distribution the distribution of hash values of search field values (rather than the distribution of search field values themselves) is skewed (very 'bursty' or non uniform).

(62) Even if the distribution of search values is known, the choice of good hashing function

Scanned by CamScanner

of hash values; skew is therefore not a problem in practice.

(3) Further, collision, of data entries with the same hash value, cause a problem and must be handled specially;

(4) when more data entries than will fit on a page have the same hash value, we need overflow pages.

Linear Hashing :-

(1) Linear Hashing is a dynamic hashing technique, like Extendible Hashing, adjusting gracefully to inserts and deletes.

(2) In contrast to Extendible Hashing, ~~adjusting~~ it does not require a directory, deals normally with collisions and offers a lot of flexibility w.r.t the timing of bucket splits (allowing us to trade off slightly greater overflow chains for higher average space utilisation). (3) If the data distribution is very skewed, however, overflow chains could cause Linear Hashing performance to be worse than that of Extendible hashing.

(4) The scheme utilises a family of hash functions h_0, h_1, h_2 ---- with the property that each fun's range is twice that of its predecessor. That is, If h_i maps a data entry into one of M buckets,

Scanned by CamScanner

(4) h_{i+1} maps a data entry into one of 2M buckets. (5) Such a family is typically obtained by choosing a hash function h and an initial number N of buckets, and defining $h_i(value) = h(value) \bmod (2N)$. (6) If N is chosen to be a power of 2 then we apply h and look at the last d_0 bits; (7) d_0 is the number of bits needed to represent N, and $d_i = d_0 + i$. (8) Typically we choose h to be a function that maps a data entry to some integer. (9) Suppose that we get the initial number N of buckets to be 32. (10) In this case d_0 is 5, and d_1 is $d_0 + 1 = 6$; and h_1 is h mod (2*32), that is a number in the range 0 to 31. h_0 is therefore a number in the range 0 to 31. (11) The value of d_1 is $d_0 + 1 = 6$ and h_1 is h mod (2*32), that is, a number in the range 0 to 63. (12) Then h_2 yields a number in the range 0 to 127, and so on. (13) The idea is best understood in terms of rounds of splitting. (14) During round number Level, only hash functions h_{Level} and $h_{Level+1}$ are in use.

Scanned by CamScanner

~~the buckets in the file in the order~~ ~~are~~ split, one by one from the 1st to the last bucket, thereby doubling the no of buckets.

6) At any given point ~~the~~ last bucket, thereby doubling ~~the no of buckets~~ within a round, therefore, we have uckets that have been split, buckets that are yet to be plit, and buckets created by splits in this round, s illustrated in Fig 7. (below)

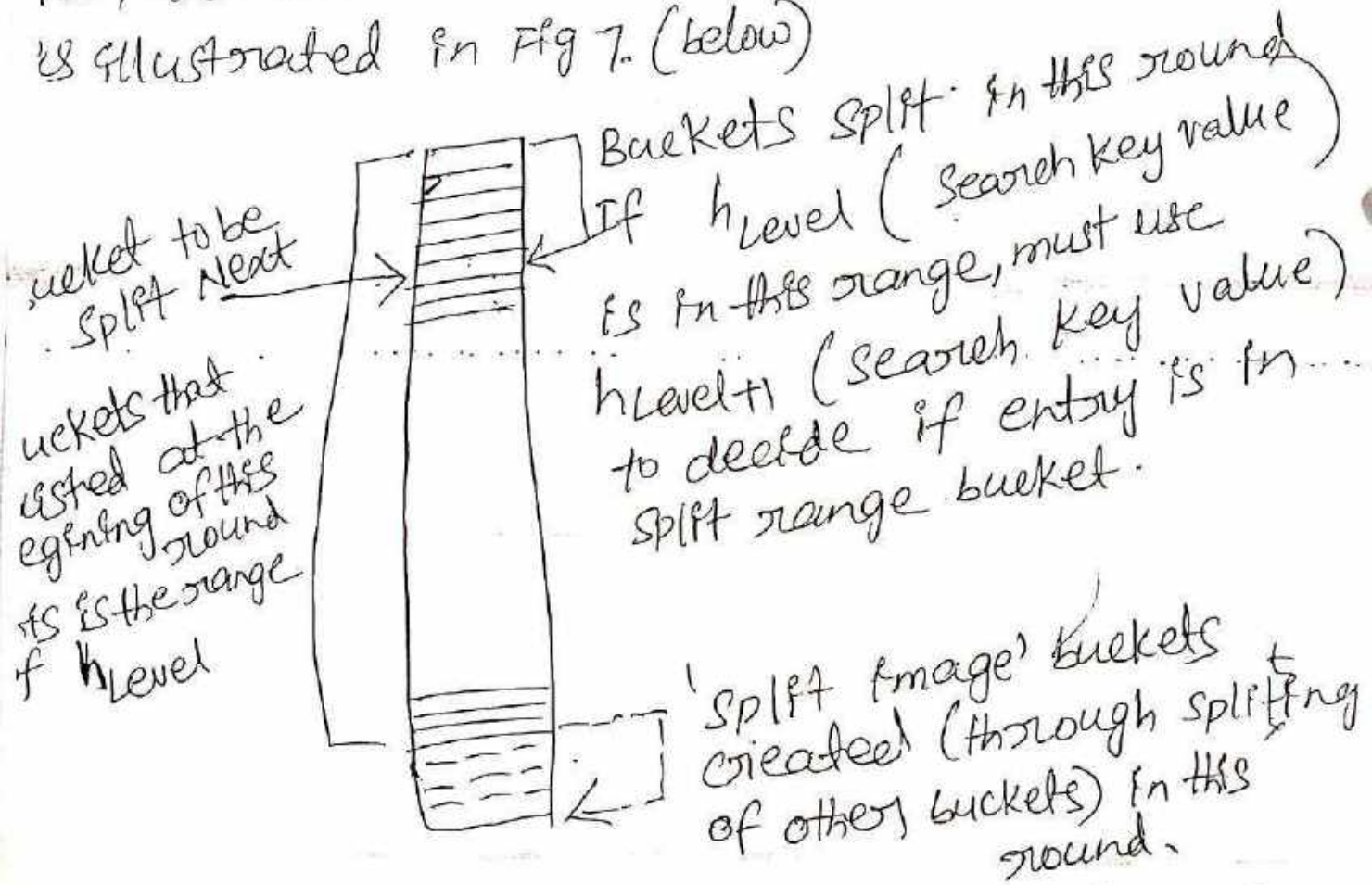


Fig 7 Buckets during a Round in Linear Hashing

(17) Consider how we search for data entry with a given search key value.

(18) we apply hash function h_{Level}, and if this

Scanned by CamScanner

needs us to one of the unsplit buckets,
(18) we simply look there. (19) If it leads us to one the split buckets, the entry may be there or it may have been moved to the new bucket created earlier in this round by splitting this bucket, to determine which of the two buckets contains the entry, we apply $h_{level+1}$.

page 11

(20) Unlike Extendible Hashing, when an Insert triggers a split, the bucket into which data entry is inserted is not necessarily the bucket that is split. (21) An overflow page is added to store the newly inserted data entry (which triggered the split), as in static Hashing. (22) However, since the bucket to split, thereby redistributing the data entries in overflow chains before the chains get to be more than one or two pages long.

page VI

(23) we now describe Linear Hashing in more detail.
(24) A counter Level is used to indicate the current round number and is initiated to 0.
(30) The bucket to split is denoted by Next and is initially bucket 0 (the 1st bucket).
(31) we denote the no' of buckets in the file at the beginning of Level by N_{Level} (32) we can easily verify that $N_{Level} = N * 2^{Level}$.

Scanned by CamScanner

Round 0, denoted by N_0 by N.

4) We show a small linear hashed file in Fig 8.

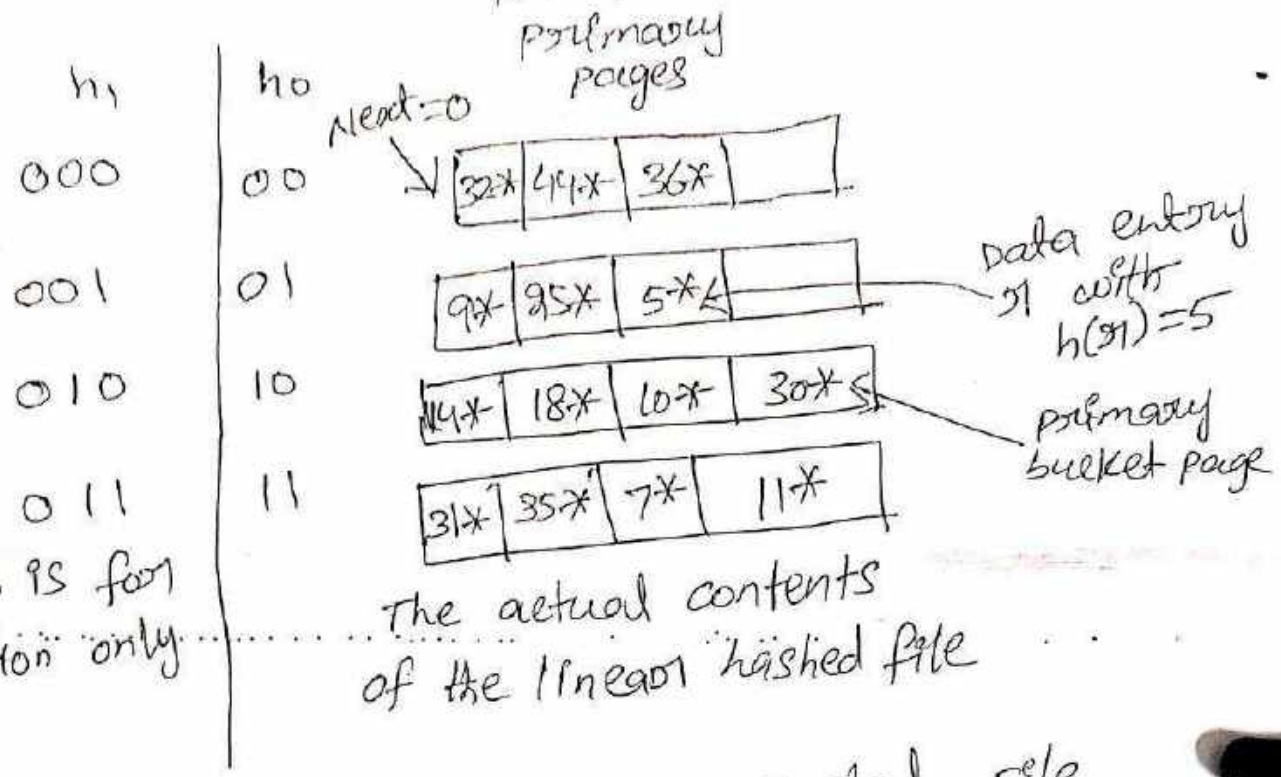


Fig 8 Example of Linear Hashed File

5) Each bucket can hold 4 data entries, and the file initially contains 4 buckets, as shown in Fig.

(36) We have considerable flexibility in how to trigger a split, thanks to the use of overflow pages. (37) We can split whenever a new overflow page is added, or we can impose additional conditions based on conditions such as space utilisation. For ours exists, a split is 'triggered' when inserting a new data entry causes the creation of an overflow page.

Scanned by CamScanner

(poor) (39) whenever a split is triggered, the file ... is Split, and hash function $h_{Level+1}$ redistributes entries b/w this bucket (say bucket number b) and its Split image; the split image is therefore bucket number $b + N_{Level}$. (40) After splitting a bucket, the value of Next is incremented by 1.

(41) In the example file, insertion of data entry 43 triggers a split. (42) The file after completing the insertion is shown in Fig 9.

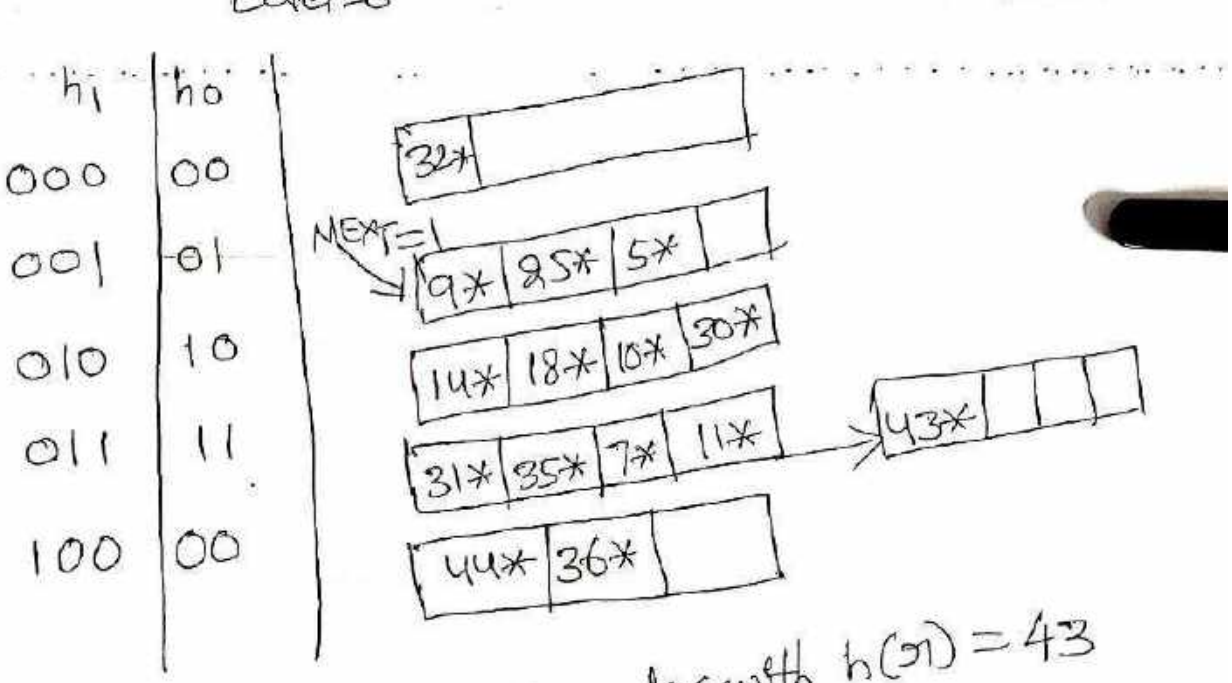


Fig 9 After Inserting Record with $h(r) = 43$

(poor) (43) At any time in the middle of a round Level all buckets above bucket Next have been split, and the file contains buckets that are their split images, as illustrated in Fig 7.

Scanned by CamScanner

Level ...
... split. from (46) If we use h_{Level} on a data entry and obtain a number b in the range Next through N_{Level}, the data entry belongs to bucket b.

(46) For ex:- $h_0(18)$ is 2 (binary 10). Since the ~~data entry belongs~~ value is b/w the current values of Next (=1) and N_1 (=4), this bucket has not been split.

(47) However, if we obtain a number b in the range 0 through Next, the data entry may be in this bucket >>) in its split image (which is bucket number 2+N_{Level}); (48) we have to use $h_{Level+1}$ to determine which of these two buckets the data entry belongs.

(49) In otherwords, we have to look at one more bit of the data entry's hash value.

(50) Ex:- $h_0(32)$ and $h_0(44)$ are both 0 (binary 00). Since Next is currently equal to 1, which indicates a bucket that has been split, we have to apply h_1.

(51) we have $h_1(32) = 0$ (binary 000) and $h_1(44) = 4$ (binary 100). Therefore, 32 belongs in bucket A and 44 belongs in its split image, bucket A2.

(52) Not all insertions triggers a split, of course

(53) If we insert 37* into the file shown in Fig 9, the appropriate

Scanned by CamScanner

bucket has space. This is how the new data entry is inserted. The file after the insertion is shown in Fig 10.

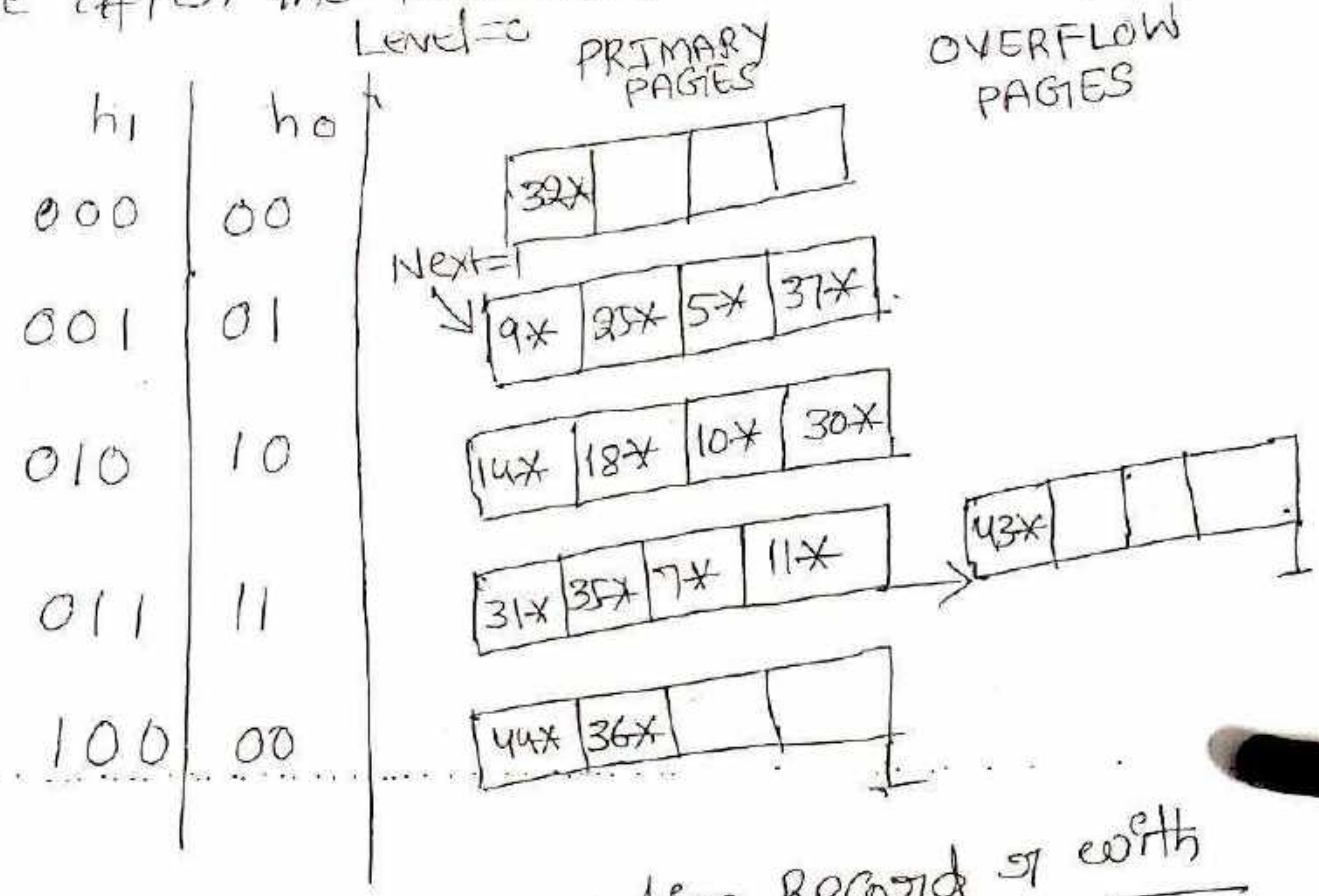
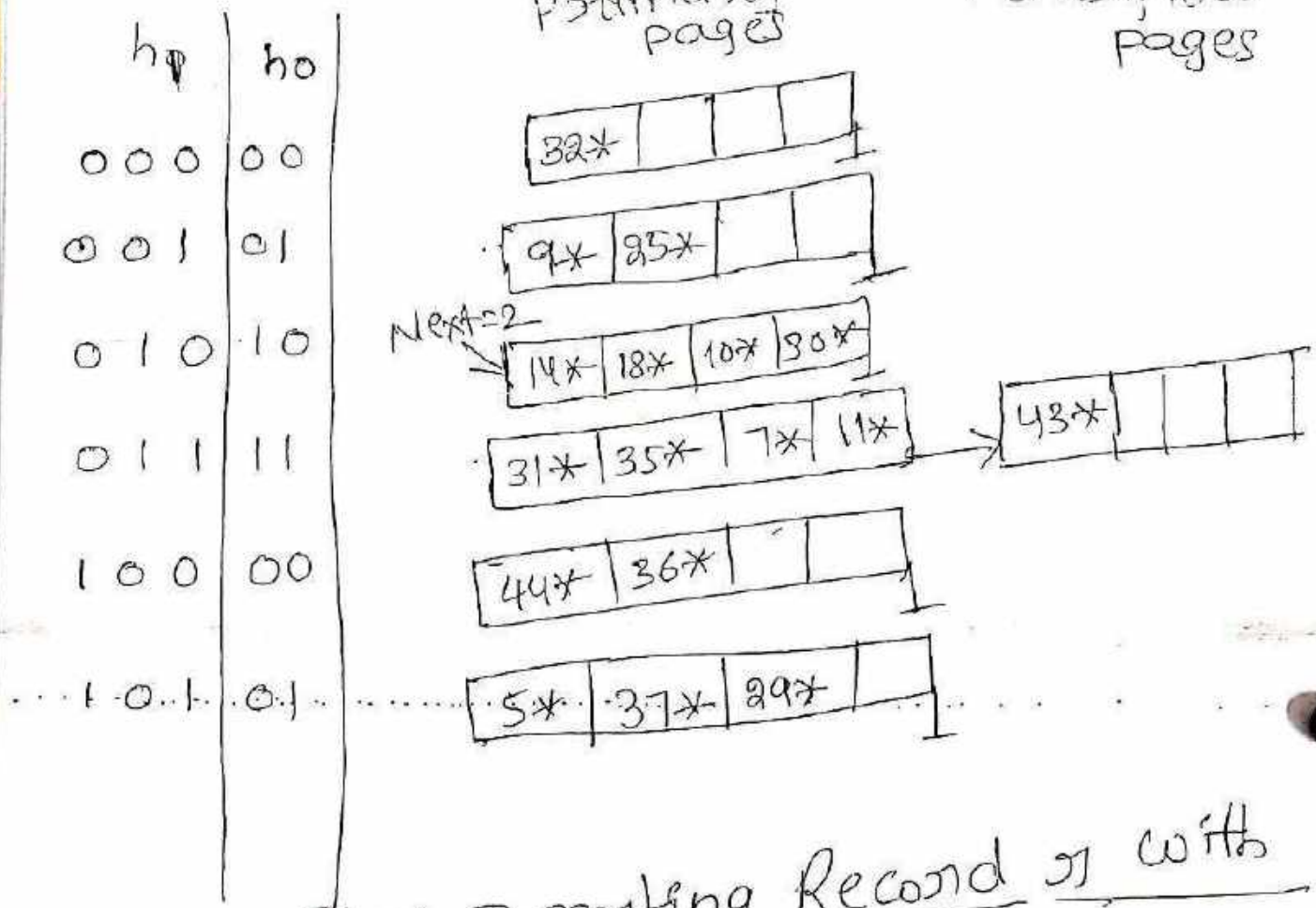


Fig 10 After Inserting Record with h(r) = 37

10th page (54) Sometimes the bucket pointed to by Next (the current candidate for splitting) is full, and a new data entry should be inserted in this bucket. (55) In this case, a split is triggered, of course, but we do not need a new overflow bucket, (56) this situation is illustrated by inserting 29* into the file shown in Fig 10. the result is shown in Fig 11.

Scanned by CamScanner

Level=0



After Inserting Record r with h(r)=29

(57) When Next is equal to $N_{Level}-1$ and a split is triggered, we split the last of the buckets present in the file at the begining of round Level. (58) The no' of buckets after the split is twice the number at the begining of the round, and we start a new round with Level incremented by 1 and Next reset to 0. (59) Incrementing Level amounts to doubling the effective range into which keys are hashed.

Scanned by CamScanner

(60) Consider the example file in Fig. 12,

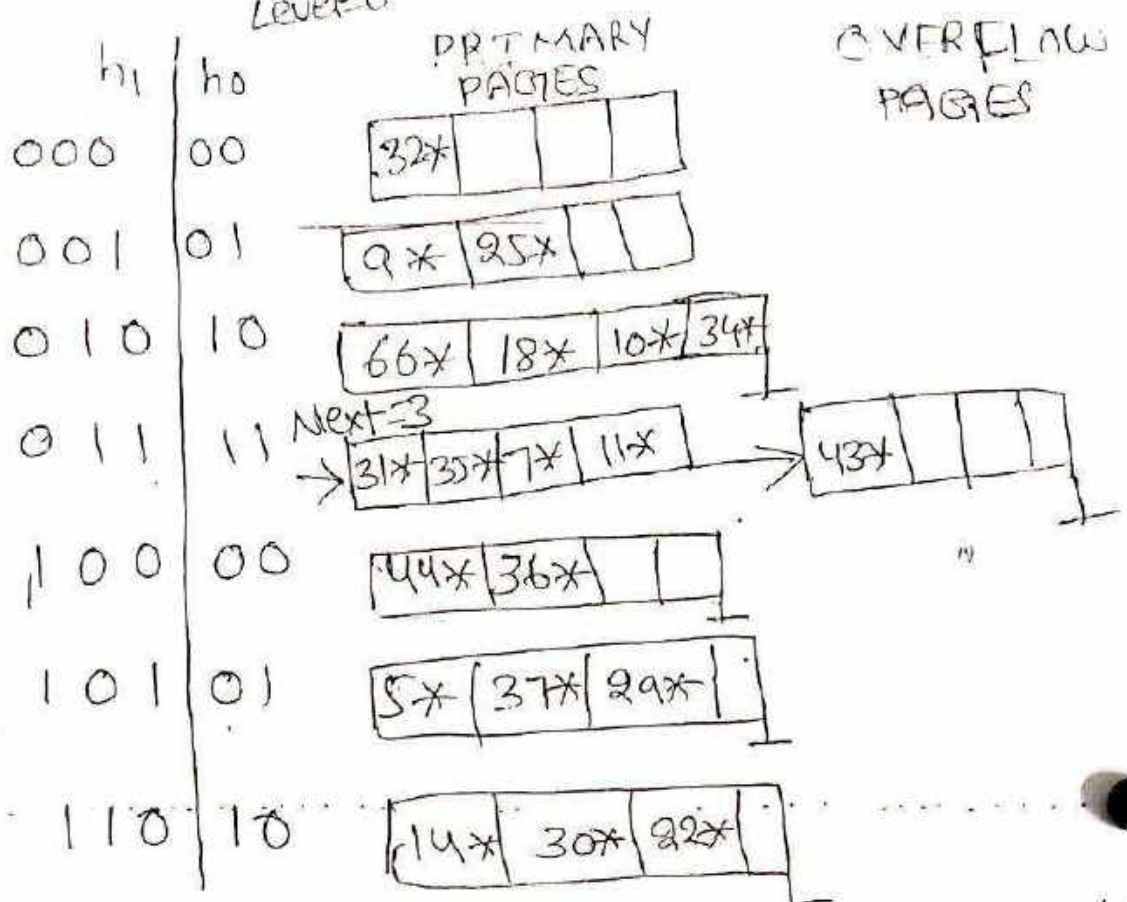
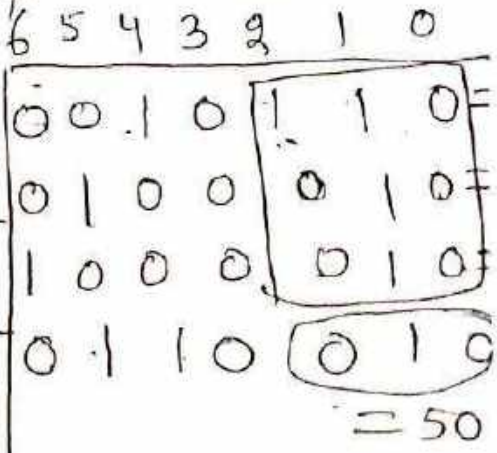


Fig 12

After Inserting with h(r) = 22, 11 and 34

which was obtained from the file of Fig 11 by inserting 22*, 66*, and 34*

(61) Inserting 50* causes a split that leads to incrementing Level, the file after this insertion is shown in Fig 13. (62) In summary, the equality selection costs just one disk I/O unless the bucket has overflow pages; (63) in practice, the cost



Scanned by CamScanner

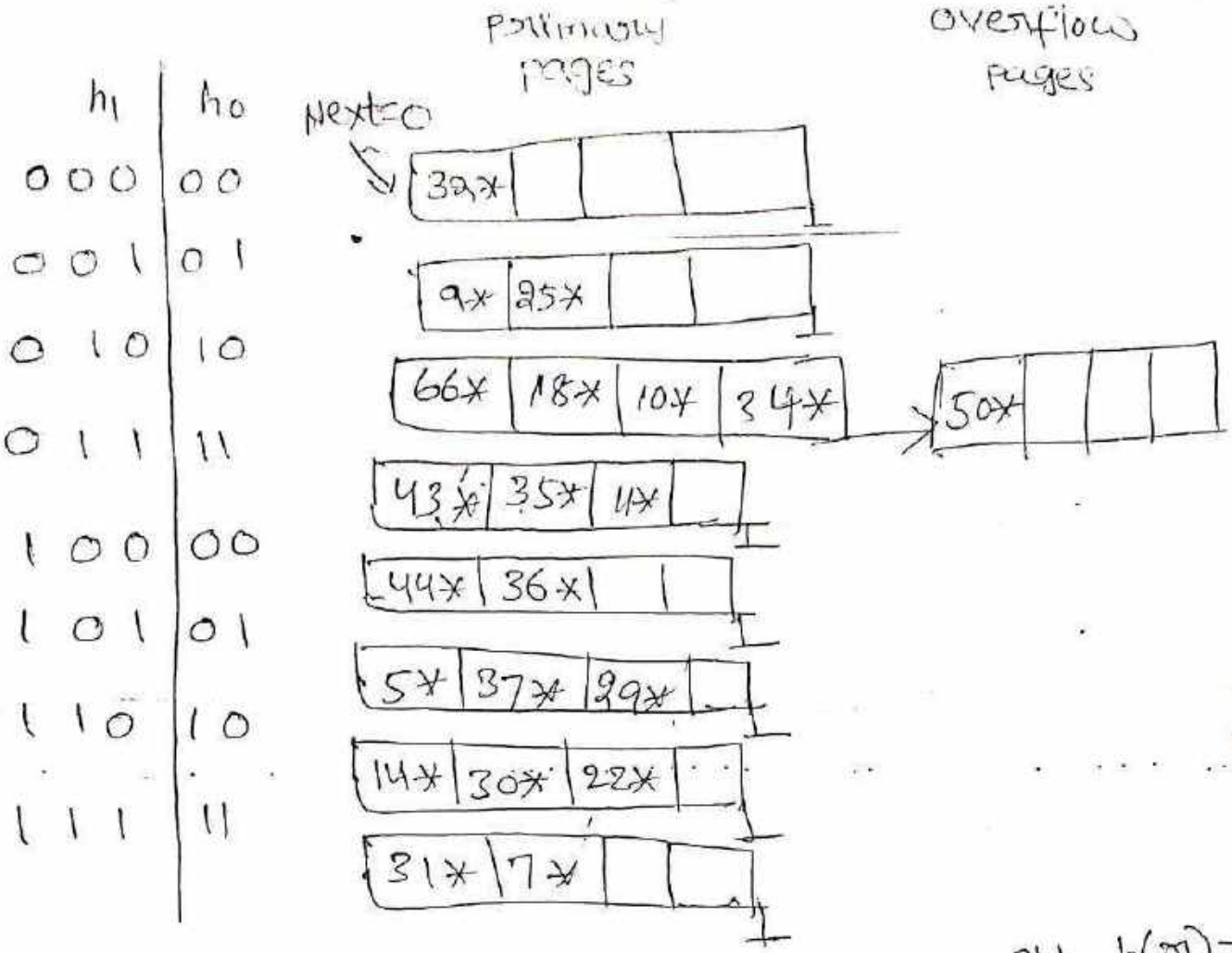


Fig 13 After Inserting Record r with $h(r)=50$

on average is about 1.2 disk accesses for reasonably uniform data distribution. (the cost can be considerably worse — linear in the no of data stores in the file — if the distribution is very skewed, the space utilisation is also very poor with skewed data distribution.)

4) Inserts require reading & writing a single page, unless split is triggered.

5) Deletion is essentially the inverse of insertion.

Scanned by CamScanner

removed and Next can be decremented.

(67) (If Next is 0 and last becomes empty, Next is made to point to bucket $(M/2)-1$, where M is the current no' of buckets, Level is decremented, and empty bucket is removed.)

(68) If we wish, we can combine the last bucket with its split image even when it is not empty, using some criterion to trigger this merging in essentially the same way. (69) The criterion is typically based on the occupancy of the file, and merging can be done to improve space utilisation.

Extendible VS Linear Hashing:-

(1) To understand the relationship b/w Linear Hashing and Extendible Hashing imagine that we also have a directory in Linear Hashing with elements 0 to N-1 (2) The 1st split is at bucket 0, and so we add directory element N, (3) In principle, we may imagine that the entire directory has been doubled at this point, because element 1 is the same as element N+1, element 2 is the same as N+2, and so on, we can avoid the actual copying for the rest of directory.

(4) The 2nd split occurs at bucket 1; now directory element N+1 becomes significant and is added. At the end of the round, all the original N buckets one split,

Scanned by CamScanner

... and the directory is doubled in size (because all elements point to distinct buckets).

(6) We observe that the choice of hashing function is actually very similar to what goes on in Extendible hashing - in effect, moving from h_i to h_{i+1} in linear hashing corresponds to doubling the directory in Extendible Hashing.

(7) Both operations double the effective range into which key values are hashed; but whereas the directory is doubled in a single step of Extendible Hashing, moving from h_i to h_{i+1}, along with a corresponding doubling in the no of buckets, occurs gradually over the course of a round in Linear Hashing.

(8) The new idea behind Linear hashing is that a directory can be avoided by a clever choice of the bucket to split. (9) On the other hand, by always splitting the appropriate bucket. (10) Extendible Hashing may lead to reduced number of splits and higher bucket occupancy.

(11) The directory analogy is useful for understanding the ideas behind Extendible and Linear Hashing.

(12) However, the directory structure can be avoided for Linear Hashing. (but not for Extendible Hashing)

Scanned by CamScanner

by allocating primary bucket pages consecutively, which would allow us to locate the page for bucket i by a simple offset calculation.

(13) For uniform distributions, this impl^n of Linear Hashing has a lower average cost for equality selections (because the directory level is eliminated)

(14) For skewed distributions, the impl^n could result in many empty or nearly empty buckets, each of which is allocated at least one page, leading to poor performance relative to Extendible Hashing, which is likely to have higher bucket occupancy.

(15) A different impl^n of Linear Hashing, in which a directory is actually maintained, offers the flexibility of not allocating one page per bucket, null directory elements can be used as in Extendible Hashing. However this implementation introduces the overhead of directory level and could prove costly for large, uniformly distributed files. (Also, although this impl^n alleviates the potential problem of low bucket occupancy by not allocating pages for empty buckets, it is not a complete solution because we can still have many pages with very few entries)

Scanned by CamScanner

for each data entry, fetch the corresponding data record from the underlying file, obtaining data records in the sort order ⟨age, sal⟩.

i) → We can read all data entries at a cost of 1.1B(D+6.7RC) I/Os.

→ Now comes the expensive part: we have to fetch the employee record for each data entry in the index.

⇒ The cost of fetching the employee records is one I/O per record, since the index is unclustered and each data entry on leaf page of the index could point to a different page in the employee file.

→ The cost of this step is BR(D+C), which is prohibitively high.

⇒ If we want the employee records is sorted order, we would be better off ignoring the index and scanning the employee file directly, and then sorting it.

→ A simple rule of thumb is that a file can be sorted by a two-pass algorithm in which each pass requires reading & writing the entire file. Thus, the I/O cost of sorting a file with B pages is 4B, which is much less than the cost of using an unclustered index.

Scanned by CamScanner

Search with Equality Selection:-

(1) We assume that equality selection matches the search order (age, sal).

→ We can locate the 1st page containing the desired data entry or entries, should any qualifying entries exist, in $\log_F 0.15B$ steps, that is, by fetching all pages from the root to the appropriate leaf.

→ Each step requires a disk I/O and two comparisons.

→ Once the page is known, the 1st qualifying data entry can again be located by a binary search of the page at a cost of $C \log_2 6.7R$.

→ The 1st qualifying data record can be fetched from the employee file with another I/O.

→ The cost is $D \log_F 0.15B + C \log_2 6.7R + D$, which is a significant improvement over searching sorted files.

(2) If several records qualify (Ex:- "Find all employees aged 18"), they are not guaranteed to be adjacent to each other.

The cost of retrieving all such records is the cost of locating the 1st qualifying data entry ($D \log_F 0.15B + C \log_2 6.7R$) plus one I/O per qualifying record. The cost of using an unclustered index is therefore very dependent on the no' of qualifying

Scanned by CamScanner

Search with Range Selection:-

> Again assuming that the range selection matches
e composite key, the 1st record that satisfies the
lection is located as it is for search with equality.
Subsequently, data entries are sequentially
etrieved (using the next and previous links at the
af level of the index) until a data entry is found
hat does not satisfy the range selection.
> For each qualifying data entry, we incur one
/o to fetch the corresponding employee records.
The cost can quickly become prohibitive as the
o' of records that satisfy the range selection
ncreases.
> As a rule of thumb, if 10 percent of data
ecords satisfy the selection condition, we are better
ff retrieving all employee records, sorting them,
nd then retaining those that satisfy the selection.

Insert:- we must 1st insert the record in the
mployee file, at a cost of 2D+c.
In addition, we must insert the corresponding
data entry in the index.
Finding the right leaf page costs $D \log_F 0.15B$ +
$C \log_2 6.7R$, and writing it out after adding
the new data entry costs another D.

Scanned by CamScanner

Delete:- we need to locate the data records in the employee file and the data entry in the index and this search step costs $D \log_F 0.15B + C \log_{6.7R} 1$
→ Now, we need to write out the modified pages in the index and the data file, at a cost of 2D.

Heap File with Unclustered Hash Index

(1) As for unclustered tree indexes, we assume that each data entry is one tenth the size of data record. we consider only static hashing in our analysis, and for simplicity we assume that there are no overflow chains.

(2) In static hashed file, pages are kept about 80 per occupancy (to leave space for future insertions and minimise overflows as the file expands).
→ This is achieved by adding a new page to a bucke when each existing page is 80 percent full, when records are initially loaded into a hashed file structure
→ The no' of pages required to store data entries is therefore 1.25 times the no' of pages when the entries are densely packed, that is, $1.25(0.10B) = 0.125B$.
→ the no' of data entries that fit on a page is $10(0.80R) = 8R$, taking into ale the relative size and occupancy.

Scanned by CamScanner

scan: As for an unclustered tree index, all data entries can be retrieved inexpensively, at a cost of 0.125 B(D+8RC) I/Os.

However, for each entry, we incur the additional cost of one I/O to fetch the corresponding data record; the cost of this step is BR(D+C). This is prohibitively expensive, and further, results are unordered. So no one even scans a hash index.

Search with Equality Selection:

1) This operation is supported very efficiently for matching selections, that is, equality conditions ove specified for each field in the composite search key (age, sal).

The cost of identifying the page that contains qualifying data entries is H.

Assuming that this bucket consists of just one age (i.e, no overflow pages), the cost of scanning the page is 0.5(8R) C = 4RC.

Finally, we have to fetch the data record from re employee file, which is another D.

The total cost is therefore H+2D+4RC, which is ven lower than the cost for a tree index.

) If several records qualify, they are not guaranteed to be adjacent to each other.

Scanned by CamScanner

→ The cost of retrieving all such records is the cost of locating the 1st qualifying data entry (H+D+4) plus one I/O per qualifying record.

→ The cost of unclustered index therefore depe heavily on the no of qualifying records.

Search with Range Selection:—

The hash structure offers no help, and the entire heap file of employee records must be scanned at a cost of B(D+RC).

Insert:— we must 1st insert the record in the employee heap file, at a cost of 2D+C.

In addition, the appropriate page in the index must be located, modified to insert a new data entry, and then written back. The additional cost is H+2D+C.

Delete:— We need to locate the data record in the employee file and the data entry in the index; this search step costs H+2D+4RC. Now, we need to write out the modified pages in the index and the data file, at a cost of 2D.

Scanned by CamScanner

Comparision of I/O costs:-

Below Fig 4 compares I/O costs for the various file organisations.

File Type	Scan	Equality Search	Range Search	Insert	Delete.
Heap	BD	0.5BD	BD	2D	Search + D
Sorted	BD	$D \log_2 B$	$D \log_2 B$ + # matching pages	Search + BD	Search + BD
Clustered	1.5BD	$D \log_F 1.5B$	$D \log_F 1.5B$ + # matching pages	Search + D	Search + D
Unclustered tree index	BD(R + 0.15)	$D(1 + \log_F 0.15B)$	$D(\log_F 0.15B$ + # matching records)	$D(3 + \log_F 0.15B)$	Search + 2D
Unclustered hash index	BD(R + 0.125)	2D	BD	4D	Search + 2D

A Comparision of I/O costs

1) A heap file has good storage efficiency and supports fast scanning and insertion of records. However, it is slow for searches and deletions.

2) A sorted file also offers good storage efficiency, but insertion and deletion of records are slow. Searches are faster than in heap files. It is worth noting that, in a real DBMS, a file is almost

never kept fully sorted.

(3) A clustered file offers all the advantages of a sorted file and supports inserts and deletes efficiently. (There is a space overhead for these benefits, relative to a sorted file, but the trade-off is well worth it.)

Searches are even faster than in sorted files, although a sorted file can be faster when a large no of records are retrieved sequentially; because of blocked I/O efficiencies.

(4) Unclustered tree and hash indexes offer fast searches, insertion, and deletion, but scans and range searches with many matches are slow.

Hash indexes are better than on equality searches, but they do not support range selection searches.

(5) In summary, Fig 4 demonstrates that no one file organisation is uniformly superior in all situations.

Scanned by CamScanner

Hash-Based Indexing

(1) In this Hash-based Indexing we consider file organisation that are excellent for equality selection.

(2) The basic idea is to use a hashing function, which maps values in a search field into range of bucket numbers to find the page on which a desired data entry belongs.

3) We use a simple scheme called Static Hashing to introduce ~~data~~ idea.

This scheme like ISAM, suffers from the problem of long overflow chains, which can affect performance.

4) Two solutions to the problem are presented

- Extendible hashing
- Linear hashing.

(5) Extendible hashing scheme uses a directory to support insert & delete efficiently with no overflow pages.

(6) Linear Hashing scheme uses a clever policy for creating new buckets and supports inserts and deletes efficiently without use of a directory

Scanned by CamScanner

Although overflow pages are used, the length of overflow chains is rarely more than two.

(7) Hash-based indexing techniques cannot support range searches, unfortunately.

(8) Tree-based indexing techniques, can support range searches efficiently and are almost as good as hash based indexing for equality selection.

(9) Thus, many commercial systems choose to support only tree based Indexes.

(10) Nonetheless, hashing techniques prove to be very useful in implementing relation operations such as joins. (11) In particular, the Index Nested Loops join method generates many equality selection queries, and the difference in cost b/w a hash-based index and a tree-based index can be significant in this context.

Static hashing:- (1) Static hashing is illustrated in Fig 1 (below)

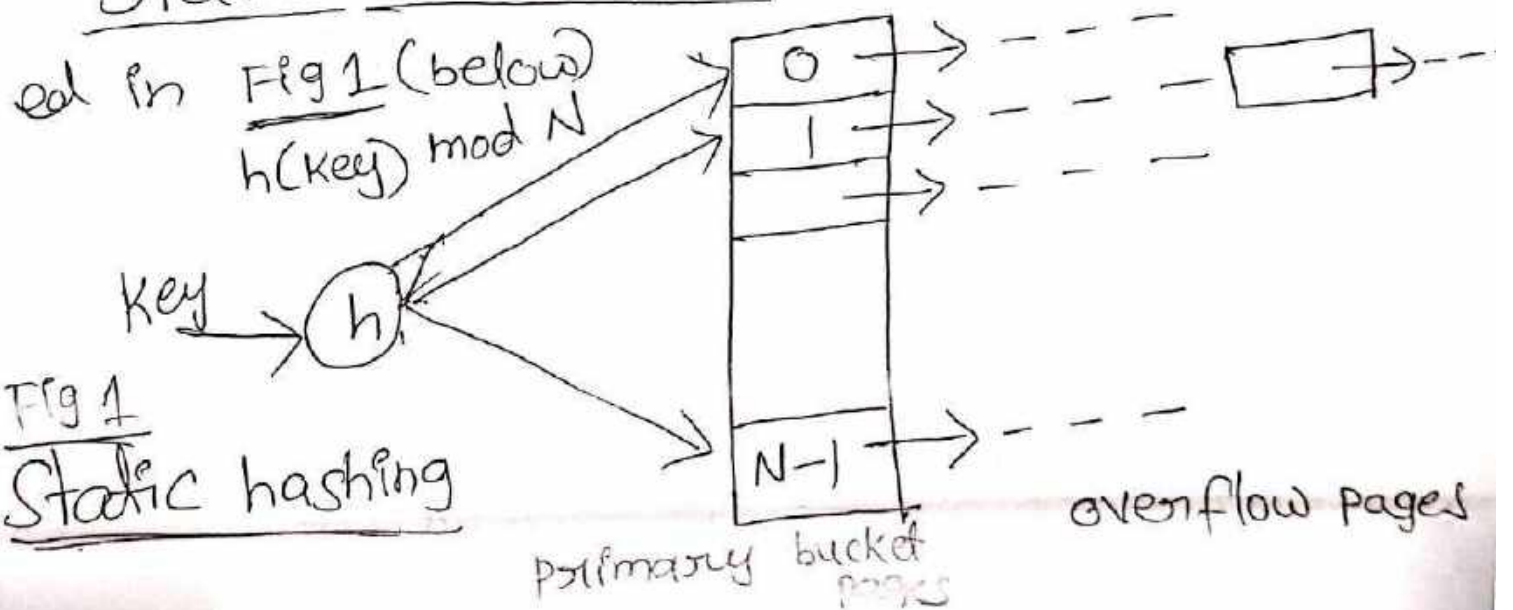


Fig 1
Static hashing

Scanned by CamScanner

The pages containing the data can be viewed as a collection of buckets, with one primary page and possibly additional overflow pages per bucket.
A file consists of buckets 0 through N-1, with one primary page per bucket initially.
Buckets contain data entries, which can be any of the three alternatives.
To search for data entry, we apply a hash function to identify the bucket to which it belongs and then search this bucket.
To speed up search of bucket, we can maintain data entries in the sorted order by search key value; (1) we do not sort entries, and the order of entries within a bucket has no significance.
To insert a data entry, we use the hash function to identify the correct bucket and then put the data entry there.

If there is no space for this data entry, we allocate a new overflow page, put the data entry on this page, and add the page to overflow chain of the bucket.
To delete a data entry, we use the hashing function to identify the correct bucket, locate the data entry by searching the bucket, and then remove it.

If the data entry is the overflow page, the overflow page is removed from the overflow

Scanned by CamScanner

chain of the bucket and added to a list of free pages.

(9) The hash function is an important component of the hashing approach. (10) It must distribute values in the domain of the search field uniformly over the collection of buckets.

(11) If we have N buckets, numbered 0 through N-1, a hash function h of the form h(value) = (a * value + b) works well in practice. (The bucket identified is h(value) mod N.) The constants a and b can be chosen to (time) the hash function.

(12) Since the no' of buckets in a static Hashing file is known when the file is created, the primary pages can be stored on successive disk pages.

(13) Hence, a search ideally requires just one disk I/O, and insert and delete operations require two IOs (read and write the page), although the cost could be higher in the presence of overflow pages.

(14) As the file grows, long overflow chains can develop. Since searching a bucket requires us to search (in general) all pages in it is easy to see how performance can deteriorate.

(15) By initially keeping pages 80 percent full, we can avoid overflow pages if the file does not grow too much, but in general the only way to get rid of overflow is to create a new file

Scanned by CamScanner

6) The main problem with static hashing is that the no of buckets fixed.

7) If a file shrinks greatly, a lot of space is wasted; more IMP, if a file grows a lot, long overflow chains develop, resulting in poor performance. Therefore, static hashing can be compared to ISAM structure, which can also develop long overflow chains in case of insertions to the same leaf. (18) Static hashing also has the same advantage as ISAM w.r.t concurrent access -

9) One simple alternative to static hashing is to periodically 'rehash' the file to restore the ideal situation (no overflow chains, about 80 percent occupancy). (20) However, rehashing takes time and the index cannot be used while rehashing is in progress.

21) Another alternative is to use dynamic hashing techniques such as Extendible and Linear hashing, which deal with inserts and deletes gracefully.

Notation and Conventions:- Record with search key k, we denote the index data entry by k*, For hash-based indexes, the 1st step in searching for, inserting or deleting a data entry with search key k is to apply a hash function h to k.

Scanned by CamScanner

we denote this operation by h(K), and the value h(K) and the value h(K) identifies the bucket for the data entry k*.

Note that a diff. search keys have the same hash value.

Extendible Hashing:-

① To understand extendible Hashing let us begin by considering a static Hashing file.

② If we have to insert a new data entry into a full bucket, we need to add an overflow page.

③ If we do not want to add overflow pages, one solution is reorganise the file at this point by doubling the no of buckets and redistributing the entries across the new set of buckets.

④ This solution suffers from one major defect — the entire file has to be read, and twice as many pages have to be written to achieve the reorganisation.

⑤ This problem, however, can be overcome by a simple idea: Use a directory of pointers to buckets. and double the size of the no of buckets by doubling just the directory and splitting only the bucket that overflowed.

⑥ To understand the idea, consider the sample file shown in Fig 2.

Scanned by CamScanner

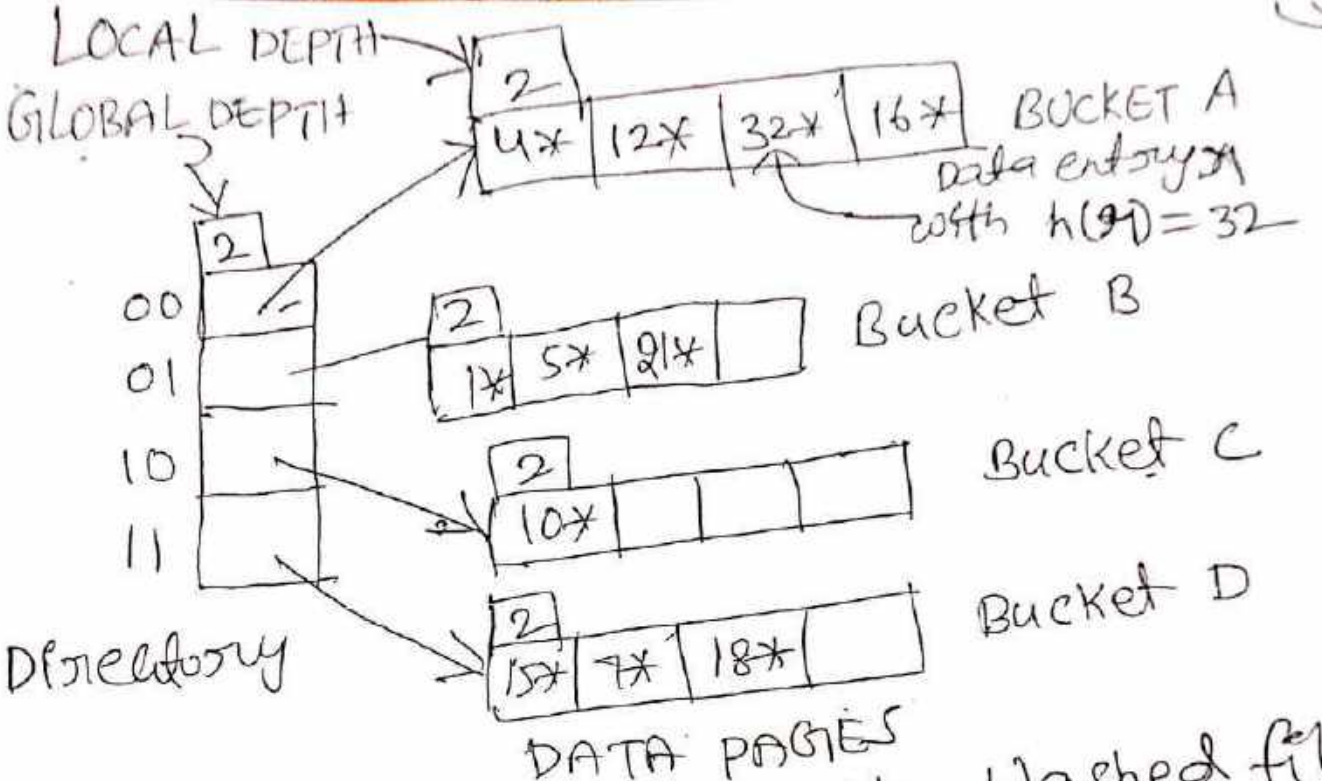


Fig 2 Example of Extendible Hashed file

1) The directory consists of an array of size 4, with each element being a pointer to a bucket.

(Ignore the global depth & local depth discussed shortly ignore them for now)

2) To locate a data entry, we apply a ~~Search~~ hash function to the search field and take the last 2 bits of its binary representation to get a number b/w 0 and 3. The pointer in this array position gives us the desired bucket. (10) We assume that each bucket can hold 4 data entries. (11) Therefore, to locate a data entry with hash value 5 (binary 101), we look at directory element 01 and follow the pointer to the data page (bucket B in the figure.)

Scanned by CamScanner

(12) To insert a data entry, we search to find the appropriate bucket.

#	3	2	1	0
B=	1	1	0	1

Ex:- To insert data entry with hash value 13 (denoted as 13*), we examine directory element 01 and go to the page containing data entries 1*, 3* and 21*. Since the page has space for an additional data entry, we are done after we insert the entry (Fig 3).

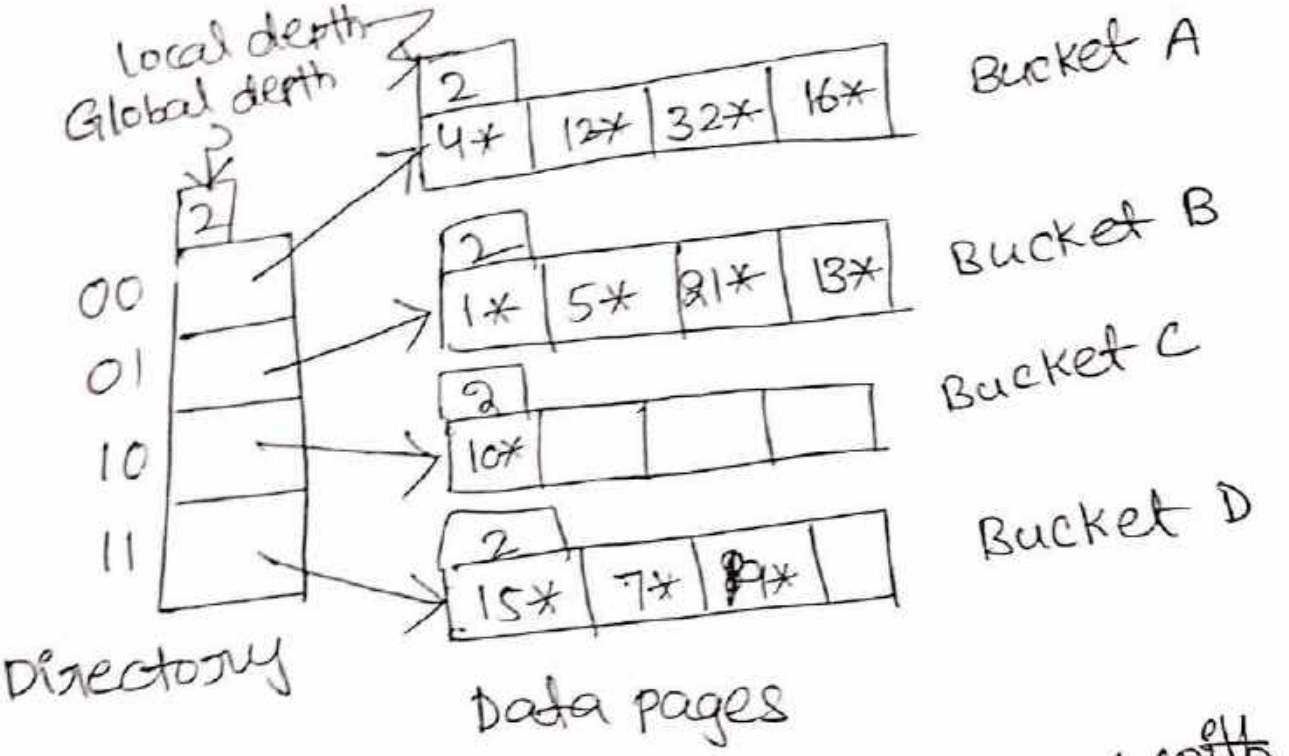


Fig 3 After Inserting Entry r with h(r)=13

(13) Next, let us consider Insertion of a data entry into a full bucket.

(14) The essence of the Extendible Hashing idea lies in how we deal with this case.

Scanned by CamScanner

5) Consider the insertion of data entry 20* (binary 10100).

6) Looking at directory element 00, we are led to bucket A, which is already full.

7) We must 1st split the bucket by allocating a new bucket and redistributing the contents (including the new entry to be inserted) across the old bucket and its 'split image'.

8) To redistribute entries across the old bucket and its split image, we consider the last 3 bits of h(r); the two bits are 00, indicating a data entry that belongs to one of these two buckets, and the 3rd bit discriminates between these buckets. The redistribution of entries is illustrated in Fig. 4

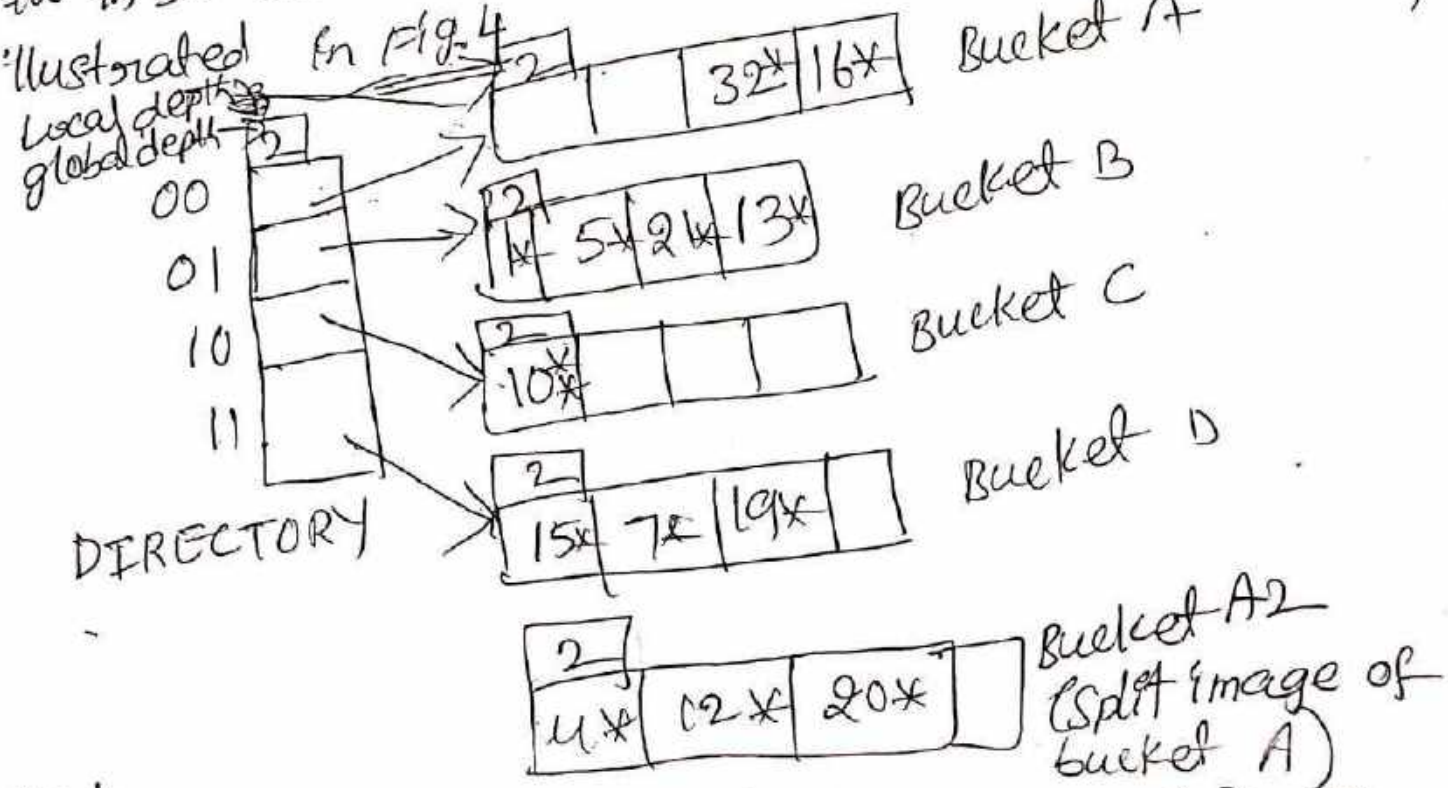


Fig 4 While Inserting Entry r with h(r)=20.

Scanned by CamScanner

(19) Note a problem that must now resolve — we need 3 bits to discriminate b/w two of our data pages (A and A2), but the directory has only enough slots to store all two-bit patterns.

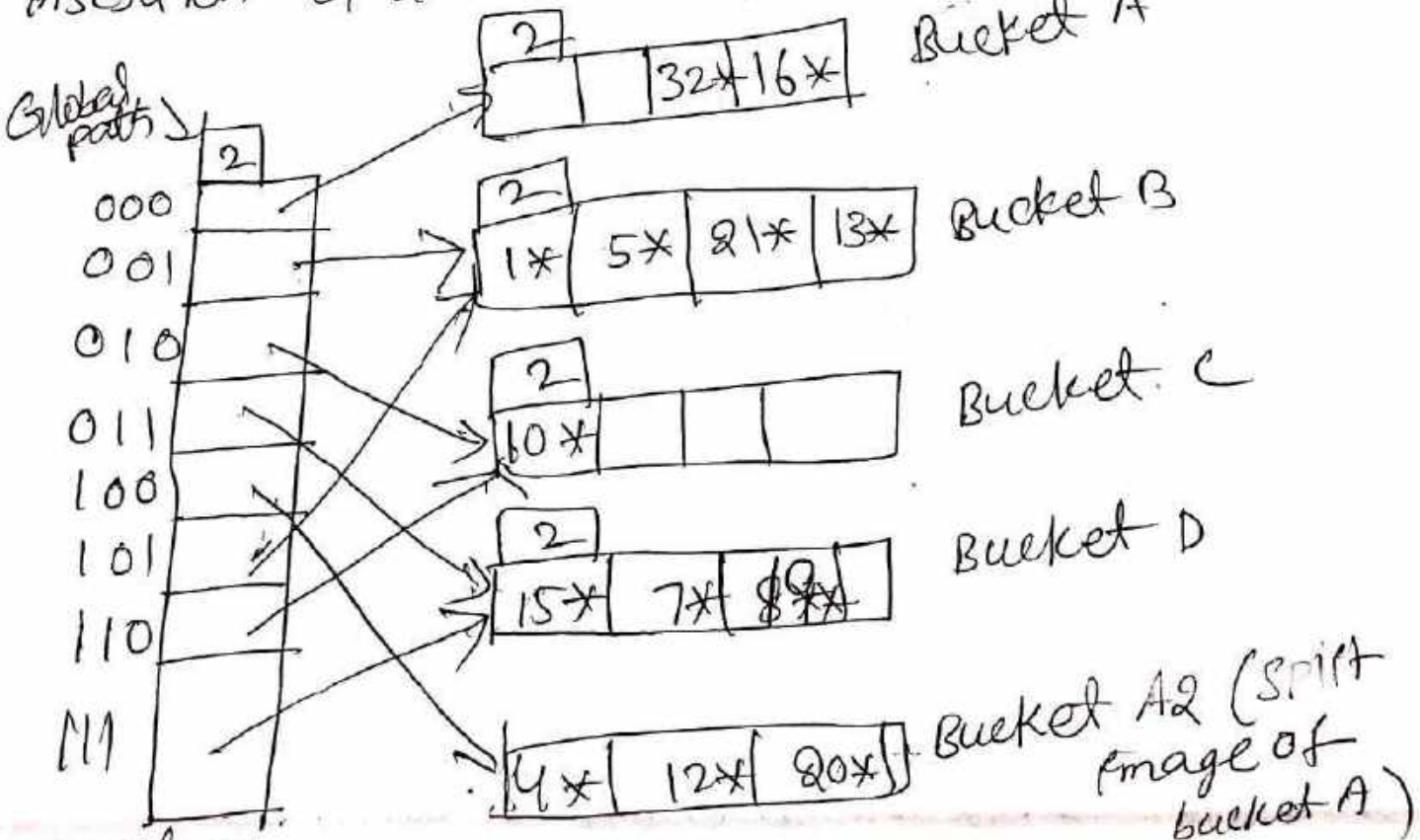
(20) The solution is to double the directory.

(21) Elements that differ only in the 3rd bit from the end are said to be 'correspond'.

(22) Corresponding elements of the directory point to the same bucket with the exception of the elements corresponding to the split bucket.

(23) In our ex:-, bucket 0 was split; so new directory element 000 points to one of the split versions and new element 100 points to the other.

(24) The sample file after completing all steps in the insertion of 20* is shown in Fig 5.



Entry 20 with h(20)=20

Scanned by CamScanner

(15) Therefore, doubling the file requires allocating (16) a new bucket page, writing both this page and the old bucket page that is being split, and doubling the directory array.

(16) The directory is likely to be much smaller than the file itself because each element is just a page-id, and can be doubled by ~~directory array~~ simply copying it over (and adjusting the elements for the split buckets). The cost of doubling is now quite acceptable.

(17) We observe that basic technique used in Extendible Hashing is to treat the result of applying a hash function h as a binary number and interpret the last d bits, where d depends on the size of the directory, as an offset into the directory. (28) In our example, d is originally 2 because we only have 4 buckets.

(29) A corollary, a logical conclusion, is that when distributing entries across a bucket and its split image. (30) we should do so on the basis of the dth bit. (Note how entries are redistributed in our example; see Fig 5) (31) The number d called the global depth of the hashed file, is kept as part of the header of the file. (32) It is used every time we used to locate a data entry.

Scanned by CamScanner

(33) An IMP point that arises is whether splitting a bucket necessitates a directory doubling.

(34) Consider our example, as shown in Fig 5.

(35) If we now insert 9*, it belongs in bucket B; this bucket is already full. We can deal with this situation by splitting the bucket and using directory element 001 and 101 to point to the bucket and its split image, as shown in Fig 6. (below)

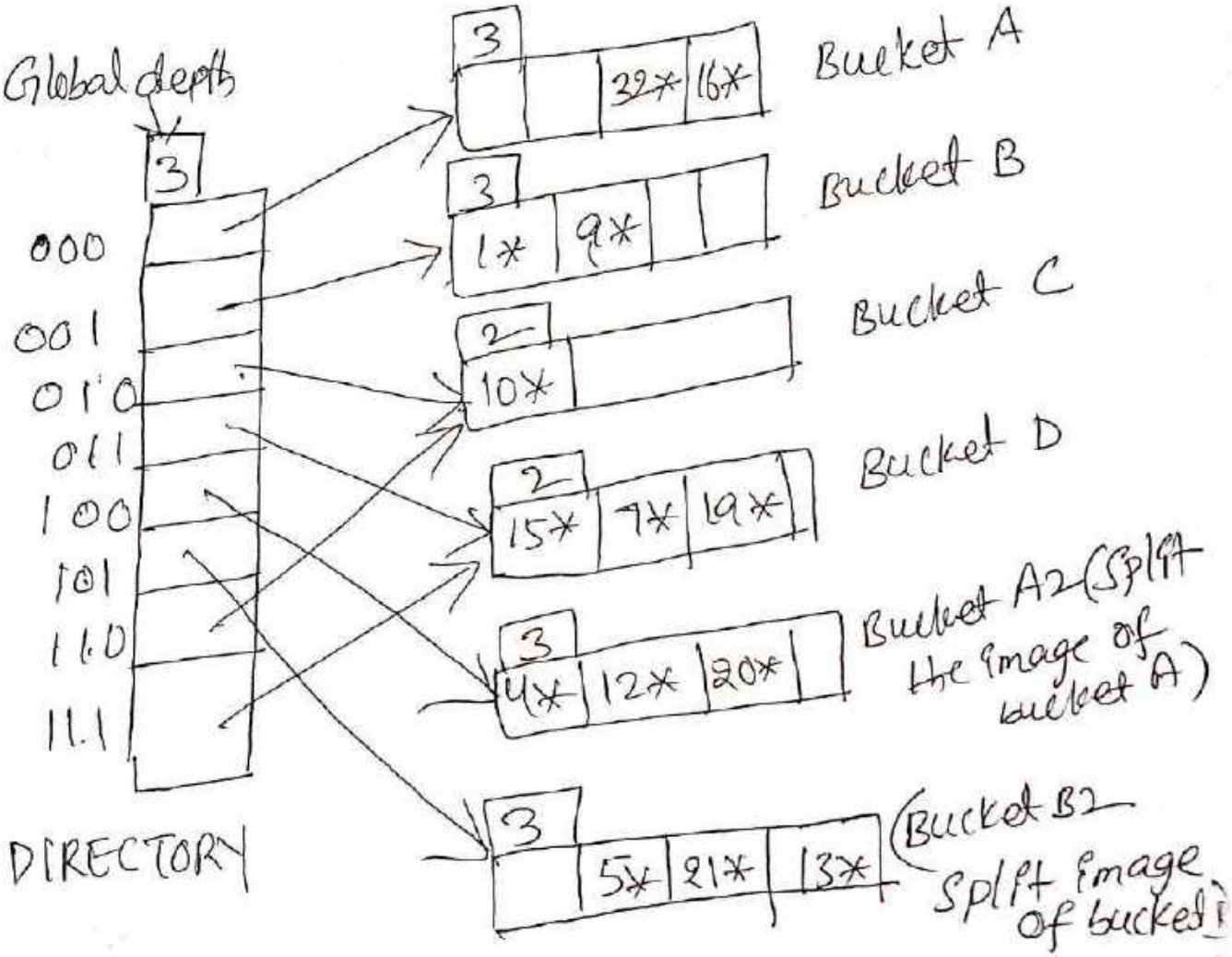


Fig 6 After inserting Entry r with h(r) = 9

Scanned by CamScanner

(36) Hence, a bucket split does not necessarily require a directory doubling. However, if either bucket A or A2 grows full and an insert then forces a bucket split, we are forced to double the directory again.

(37) To differentiate b/w these cases and determine whether a directory doubling is needed, we maintain a local depth for each bucket.

(38) If a bucket whose local depth is equal to the global depth is split, the directory must be doubled. (39) Going back to the example, when we inserted 9* into the index shown in Fig 5, it belonging to bucket B with local depth 2, whereas the global depth was 3. (40) Even though the bucket was split, the directory did not doubled.

(41) Buckets A and A2, on other hand, have local depth equal to the global depth, and, if they grow full and split, the directory must then be doubled. (42) Initially, all local depths are equal to the global depth (which is number of bits needed to express the total no of of buckets).

(43) We increment the global depth by 1 each time the directory doubles, of course.

Scanned by CamScanner

(44) Also, whenever a bucket is split (whether or not split leads to a directory doubling), we increment by 1 the local depth of split bucket and assign this same (incremented) local depth to its (newly created) split image.

(45) Intuitively, if a bucket has local depth l, the hash values of data entries in it agree on the last l bits, further, no data entry in any other buckets of the file has a hash value with the same last l bits.

(46) A total of 2^{d+l} directory elements point to a bucket with local depth l. If $d=l$, exactly one directory element points to the bucket and splitting such a bucket requires directory doubling.

$2^{3-2} = 2$

$2^{3-3} = 1$

(47) A final point to note is that we can also use the 1st d bits (the most significant bits) instead of the last d (least significant bits), but in practice the last of d-bits are used the reason is that a directory can then be doubled simply by copying it.

(48) In Summary, a data entry can be located by computing its hash value, taking the last d bits and looking in the bucket pointed to by this directory element.

Scanned by CamScanner

(49) For inserts, the data entry is placed in the bucket to which it belongs and the bucket is split if necessary to make space.

(50) A bucket split leads to increase in the local depth and, if the local depth becomes greater than the global depth as a result, to a directory doubling (and in increase in the global depth) as well.

(51) For deletes the data entry is located and removed. (52) If the delete leaves the bucket empty, it can be merged with its split image, although this step is often omitted in practice.

(53) merging buckets decreases the local depth

(54) If each directory element points to the same bucket as its split image (i.e, 0 and 2^{d-l} point to the same bucket, namely, A: 1 and $2^{d-l}+1$ point to same bucket, initially, B, which may or may not be identical to A; etc), we can halve the directory and reduce the global depth, although this step is not necessary for correctness.

(55) The insertion examples can be worked out backwards as examples of deletion. (start with the structure shown after an insertion and delete the inserted element.) In each case the original structure should be restored.

Scanned by CamScanner

(56) If the directory fits in memory, an equality selection can be answered in a single disk access, as for Static Hashing (in the absence of overflow pages), but otherwise, two disk I/O's are needed.

(57) As a typical example, a 100MB file with 100byte per data entry and a page size of 4KB contains 1 million data entries and only about 25,000 de's in the directory. (58) (Each page/bucket contains roughly 40 data entries, and we have one directory element per bucket). (59) Thus, although equality selections can be twice as slow for static Hashing files, chances are high that the directory will fit in memory and performance is the same as for static Hashing files. (60) On the other hand, the directory grows in spurts and can become large for skewed data distributions (where our assumption that the data pages contain roughly equal no' of data entries is not valid). (61) In the context of hashed files, (suddenly change)
In a skewed data distribution the distribution of hash values of search field values (rather than the distribution of search field values themselves) is skewed (very 'bursty' or nonuniform).

(62) Even if the distribution of search values is skewed, the choice of good hashing function

Scanned by CamScanner

(62) typically yields a fairly uniform distribution (4) of hash values; skew is therefore not a problem in practice.

(63) Further, collision, or data entries with the same hash value, cause a problem and must be handled specially;

(64) when more data entries than will fit on a page have the same hash value, we need overflow pages.

Linear Hashing :-

(1) Linear Hashing is a dynamic hashing technique, like Extendible Hashing, adjusting gracefully to inserts and deletes.

(2) In contrast to Extendible Hashing, ~~adjusting~~ it does not require a directory, deals normally with collisions, and offers a lot of flexibility w.r.t the timing of bucket splits (allowing us to trade off slightly greater overflow chains for higher average space utilisation). (3) If the data distribution is very skewed, however, overflow chains could cause Linear Hashing performance to be worse than that of Extendible hashing. The scheme utilises a family of hash functions h_0, h_1, h_2 ---- with the property that each fun's range is twice that of its predecessor. That is, if h_i maps a data entry into one of M buckets,

Scanned by CamScanner

(4) h_{i+1} maps a data entry into one of 2^{i+1} buckets. (5) Such a family is typically obtained by choosing a hash function h and an initial number N of buckets, and defining $h_i(value) = h(value) \bmod (2N)$. (6) If N is chosen to be power of 2, then we apply h and look at the last d_0 bits; (7) d_0 is the number of bits needed to represent N, and $d_0 = d_0 + 1$.

(8) typically we choose h to be a function that maps a data entry to some integer.

(9) Suppose that we get the initial number N of buckets to be 32. (10) In this case d_0 is 5, and d_1 is ~~$d_0+1=6$, and h_1 is $h \bmod (2*32)$, that is a number in the range 0 to 31.~~ h_0 is therefore

(11) $h \bmod 32$ that is, a number in the range 0 to 31.

(12) The value of d_1 is $d_0+1 = 6$ and h_1 is $h \bmod (2*32)$, that is, a number in the range 0 to 63. (13) Then h_2 yields a number in the range 0 to 127, and so on. The idea is best understood in terms of rounds of splitting. (14) During round number Level, only hash fun's h_{Level} and $h_{Level+1}$ are in use.

Scanned by CamScanner

5) The buckets in the file at the beginning of the round are split, one by one from the 1st to the last bucket, thereby doubling the no' of buckets.

6) At any given point ~~the~~ last bucket, ~~thereby doubling the no of buckets~~ within a round, therefore, we have buckets that have been split, buckets that are yet to be split, and buckets created by splits in this round, as illustrated in Fig 7. (below)

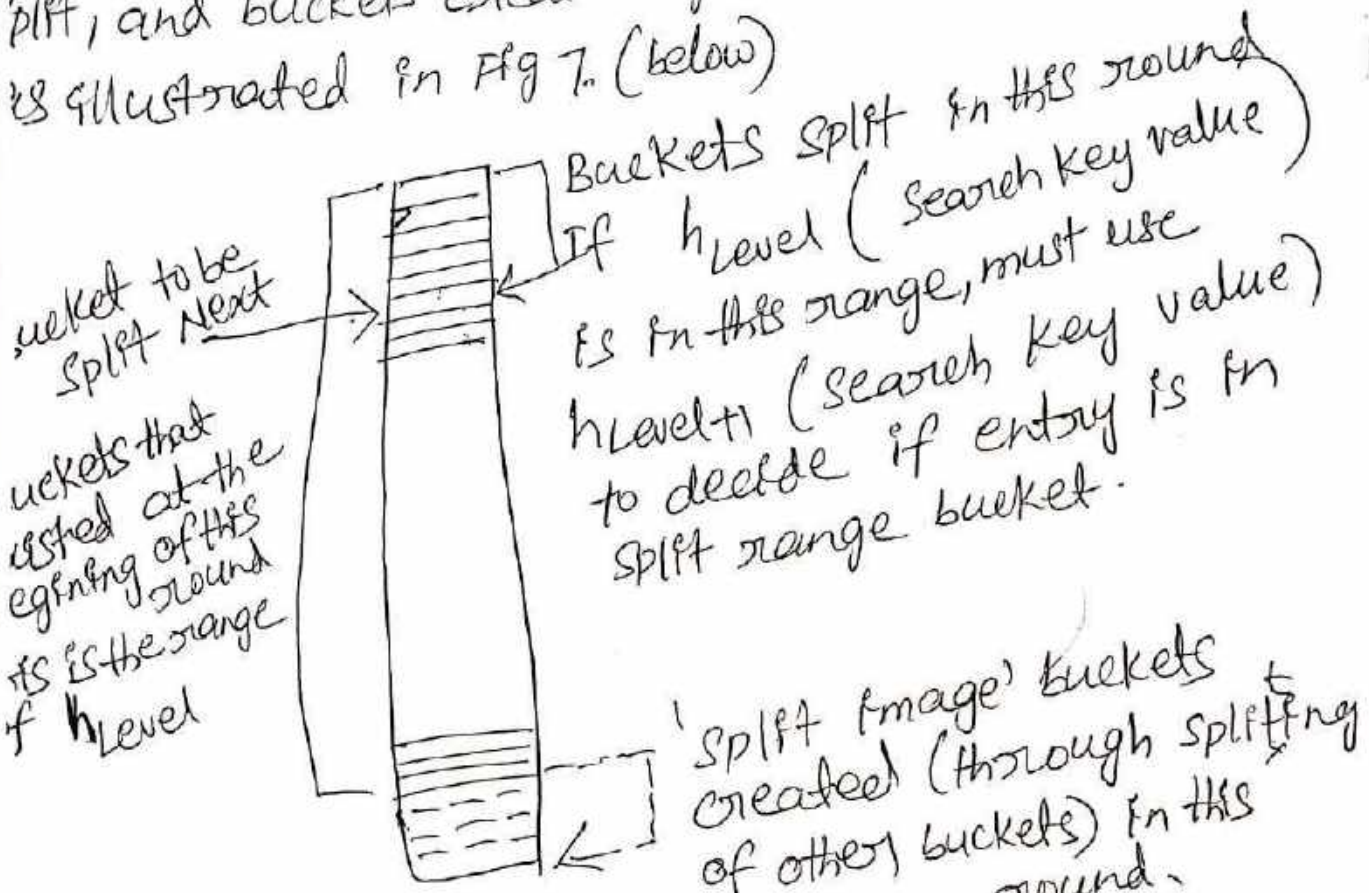


Fig 7 Buckets during a Round in Linear Hashing

(17) Consider how we search for data entry with a given search key value.

(18) we apply hash function h_{Level}, and if this

Scanned by CamScanner

reads us to one of the unsplit buckets, we simply look there. (19) If it leads us to one of the split buckets, the entry may be there or it may have been moved to the new bucket created earlier in this round by splitting this bucket, to determine which of the two buckets contains the entry, we apply $h_{level+1}$.

(20) Unlike Extendible Hashing, when an Insert triggers a split, the bucket into which data entry is inserted is not necessarily the bucket that is split. (21) An overflow page is added to store the newly inserted data entry (which triggered the split as in Static Hashing. (22) However, since the bucket to split, there by re distributing the data entries in overflow chains before the chains get to be more than one or two pages long.

(23) We now describe Linear Hashing in more detail

(24) A counter Level is used to indicate the current round number and is initiated to 0.

(30) The bucket to split is denoted by Next and is initially bucket 0 (the 1st bucket).

(31) We denote the no' of buckets in the file at the beginning of Level by N_{Level} (32) we can easily verify that $N_{Level} = N * 2^{Level}$.

Scanned by CamScanner

(33) Let the no of buckets at the beginning of round 0, denoted by N_0 by N.

(34) We show a small linear hashed file in Fig 8.

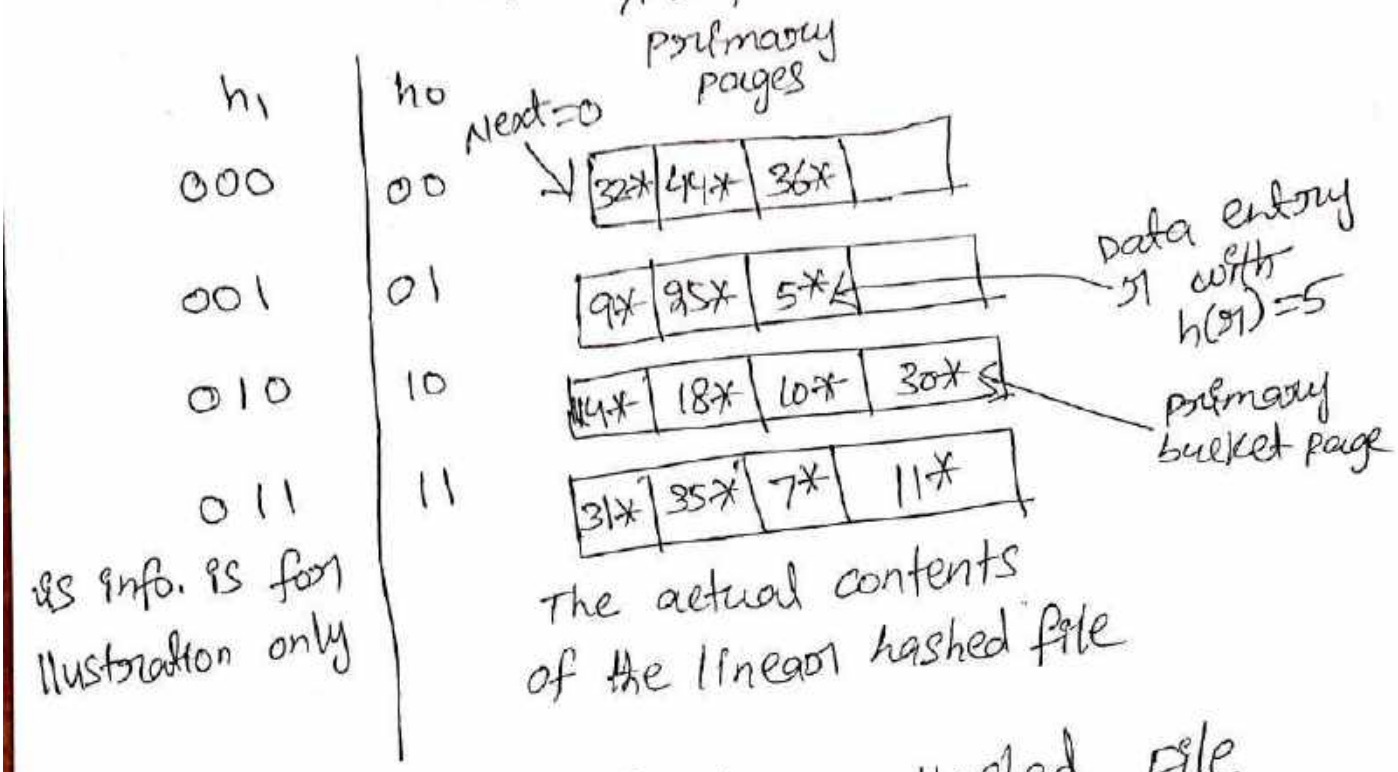


Fig 8 Example of Linear Hashed File

(35) Each bucket can hold 4 data entries, and the file initially contains 4 buckets, as shown in Fig.

(36) we have considerable flexibility in how to trigger a split, thanks to the use of overflow pages. (37) we can split whenever a new overflow page is added, and we can impose additional conditions based on conditions such as space utilisation.

(38) Post ours exists, a split is 'triggered' when inserting a new data entry causes the creation of an overflow page.

Scanned by CamScanner

still going (39) whenever a split is triggered the Next bucket is split, and hash function $h_{Level+1}$ redistributes entries b/w this bucket (say bucket number b) and its split image; the split image is therefore bucket number $b + N_{Level}$. (40) After splitting a bucket, the value of Next is incremented by 1.

(41) In the example file, insertion of data entry 43* triggers a split. (42) The file after completing the insertion is shown in Fig 9.

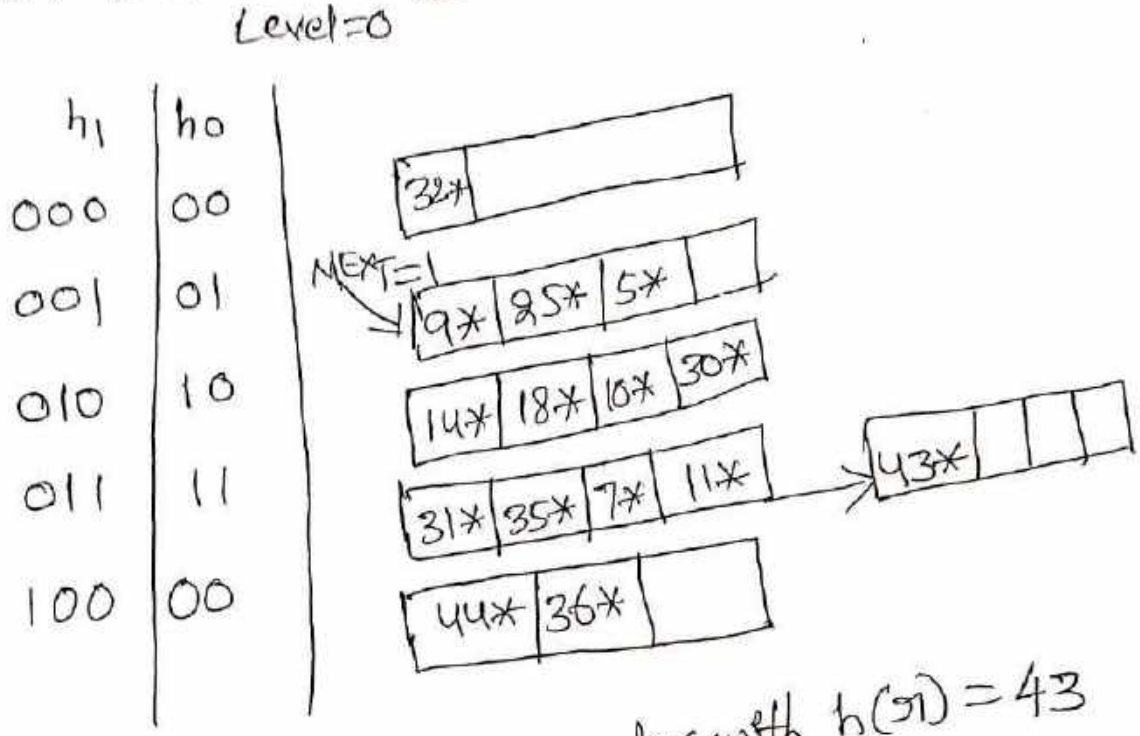


Fig 9 After Inserting Record with h(r) = 43

bth power (43) At any time in the middle of a round Level, all buckets above bucket Next have been split, and the file contains buckets that are their split images, as illustrated in Fig 7.

Scanned by CamScanner

4) Buckets Next through N_{Level} have not yet been split. (45) If we use h_{Level} on a data entry and obtain a number b in the range Next through N_{Level}, the data entry belongs to bucket b.

(46) For ex:- $h_0(18)$ is 2 (binary 10). Since the ~~data entry belongs~~ value is b/w the current values of Next (=1) and N_1 (=4), this bucket has not been split.

(47) However, if we obtain a number b in the range 0 through Next, the data entry may be in this bucket >>] in its split image (which is bucket number $b+N_{Level}$); (48) we have to use $h_{Level+1}$ to determine o which of these two buckets the data entry belongs. In otherwords, we have to look at one more bit of the data entry's hash value.

(49) Ex:- $h_0(32)$ and $h_0(44)$ are both 0 (binary 00). (50) Since Next is currently equal to 1, which indicates a bucket that has been split, we have to apply h_1. (51) we have $h_1(32)=0$ (binary 000) and $h_1(44)=4$ (binary 100). Therefore, 32 belongs in bucket A and 44 belongs in its split image, bucket A2.

(52) Not all insertions triggers a split, of course. (53) If we insert 37* into the file shown in Fig 9, the appropriate

Scanned by CamScanner

bucket has space for the new data entry. The file after the insertion is shown in Fig 10.

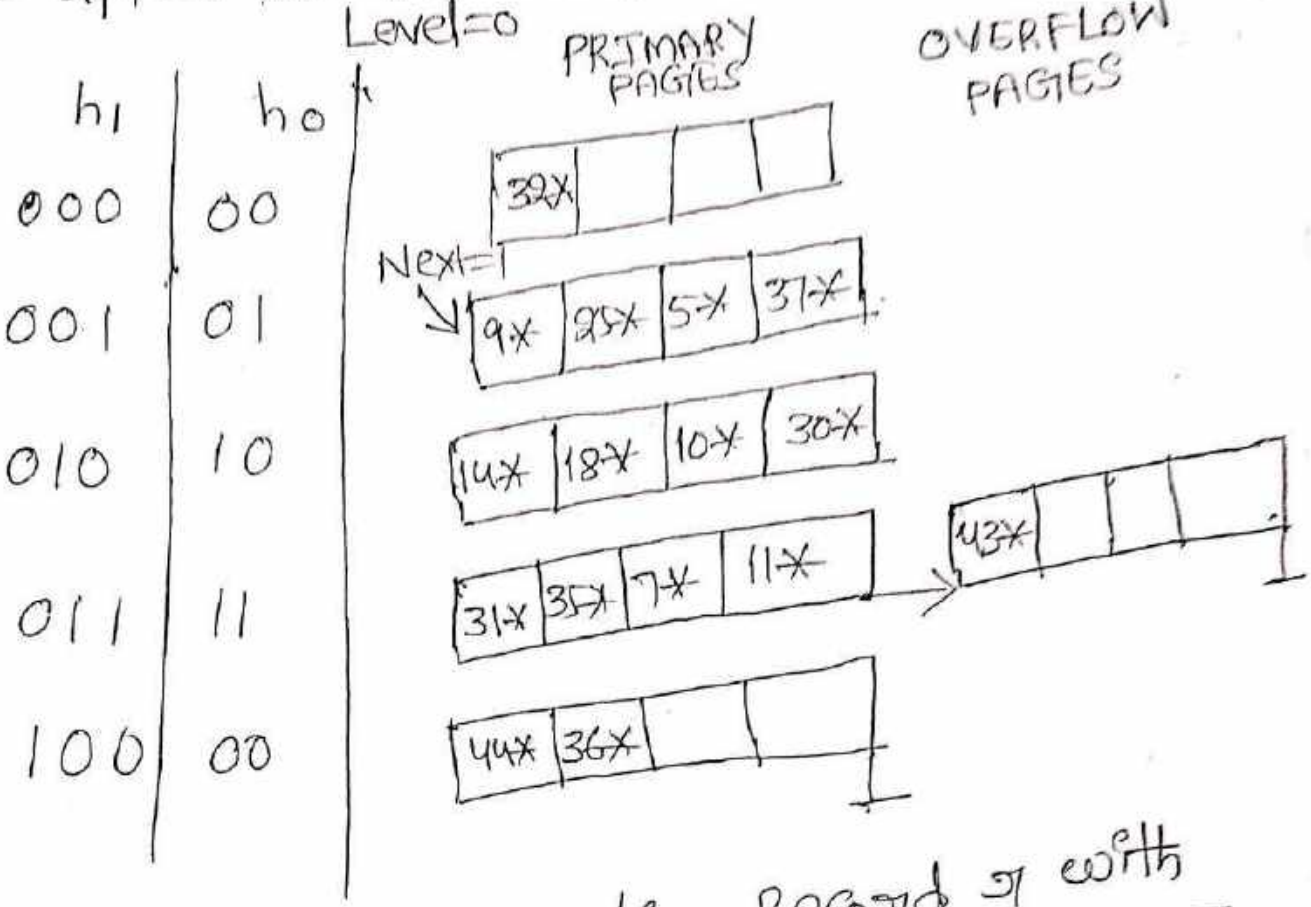
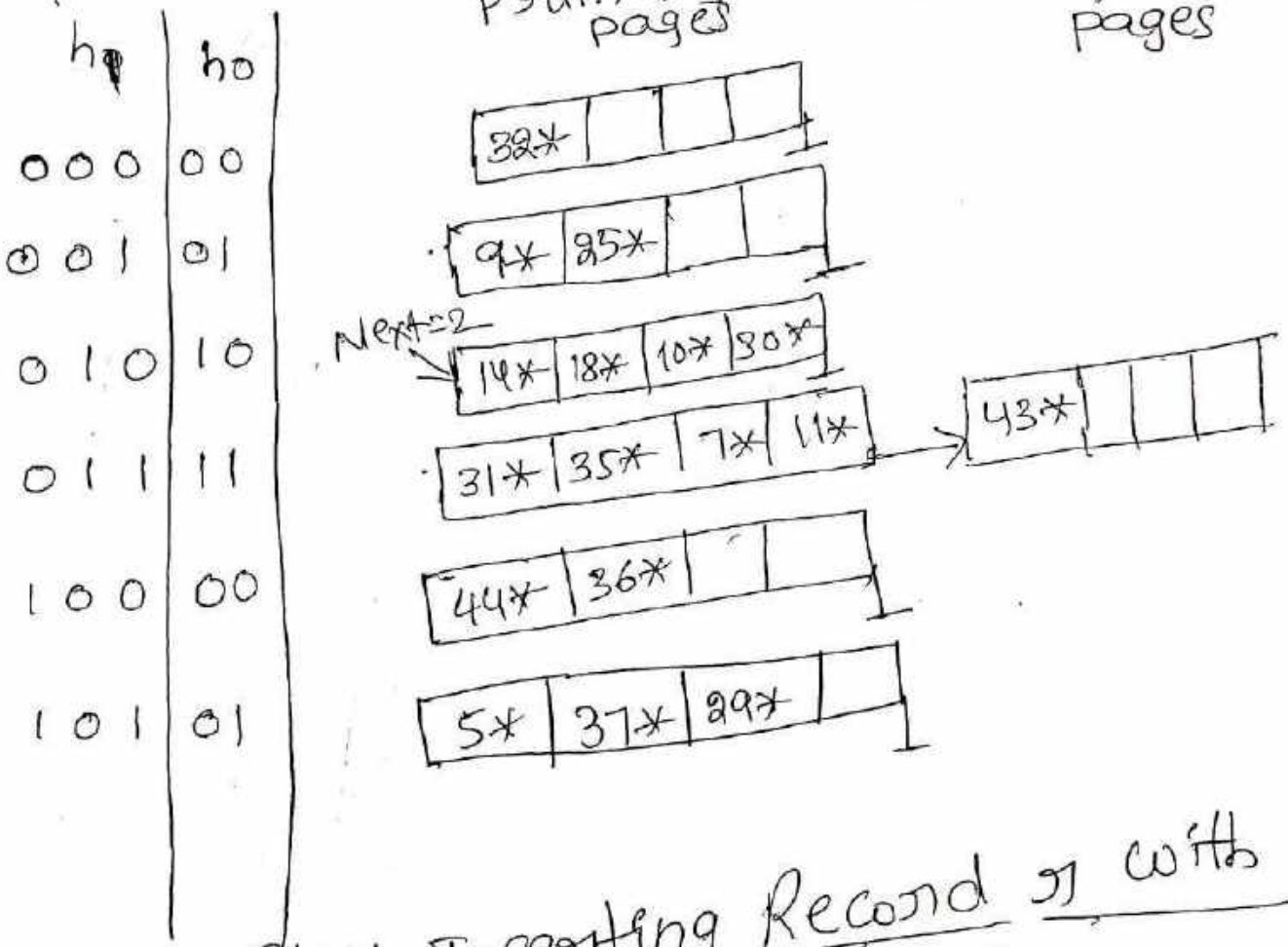


Fig 10 After Inserting Record r with h(r) = 37

10th para (54) Sometimes the bucket pointed to by next (the current candidate for splitting) is full, and a new data entry should be inserted in this bucket. (55) In this case, a split is triggered, of course, but we do not need a new overflow bucket. (56) This situation is illustrated by inserting 29* into the file shown in Fig 10. The result is shown in Fig 11.

Scanned by CamScanner

Level=0



After Inserting Record 37 with h(37)=29

para 57) When Next is equal to $N_{Level}-1$ and a split is triggered, we split the last of the buckets present in the file at the begining of round Level. 58) The no' of buckets after the split is twice the number at the begining of the round, and we start a new round with Level incremented by 1 and Next reset to 0. 59) Incrementing Level amounts to doubling the effective range into which keys are hashed.

Scanned by CamScanner

(60) Consider the example file in Fig 12,

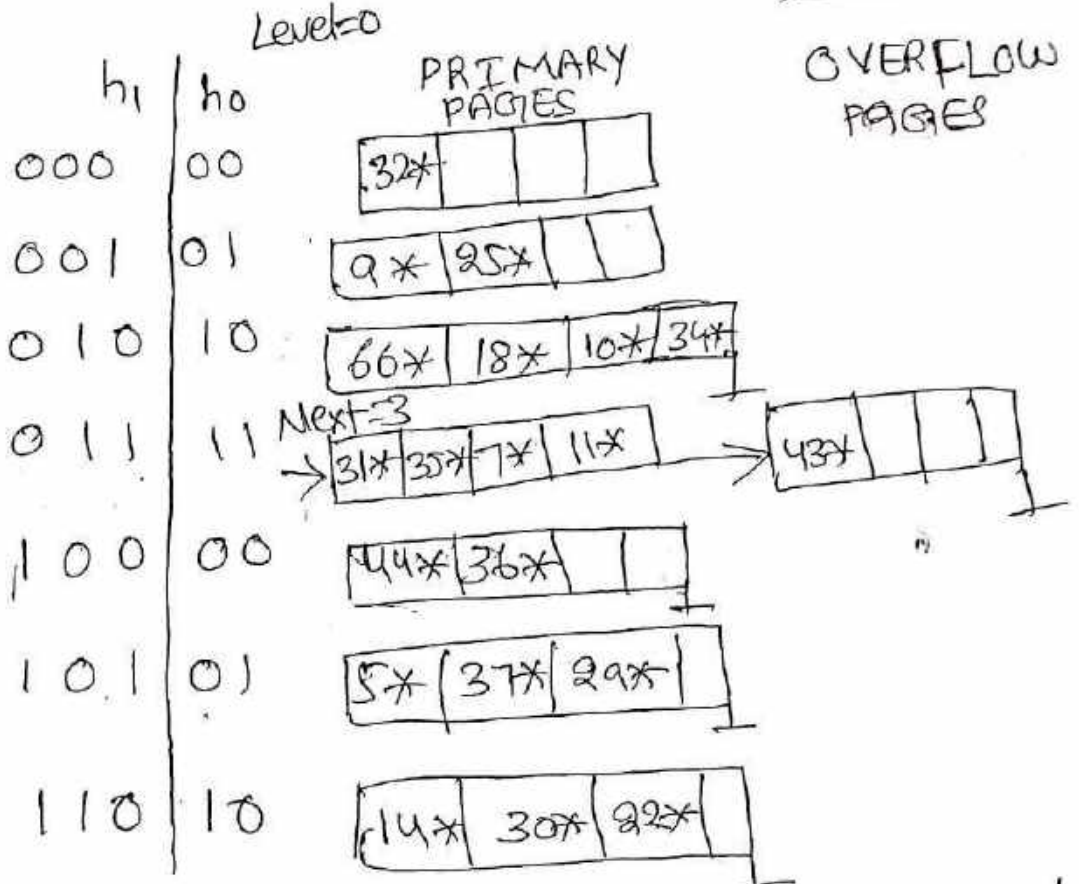
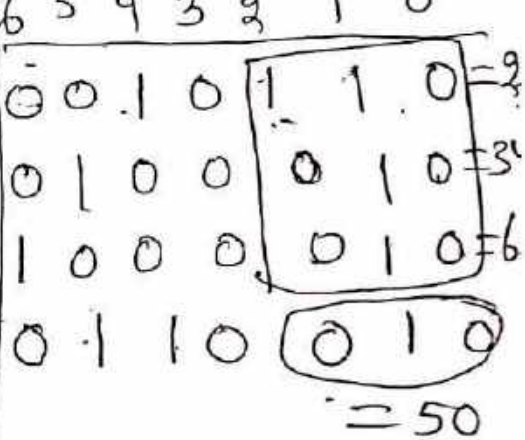


Fig 12

After Inserting with h(r) = 22, 11 and 34

which was obtained from the file of Fig 11 by inserting 22*, 66*, and 34*



(61) Inserting 50* causes a split that leads to incrementing Level, the file after this insertion is shown in Fig 13. (62) In summary, the equality selection costs just one disk I/O unless the bucket has overflow pages; (63) in practice, the cost

Scanned by CamScanner

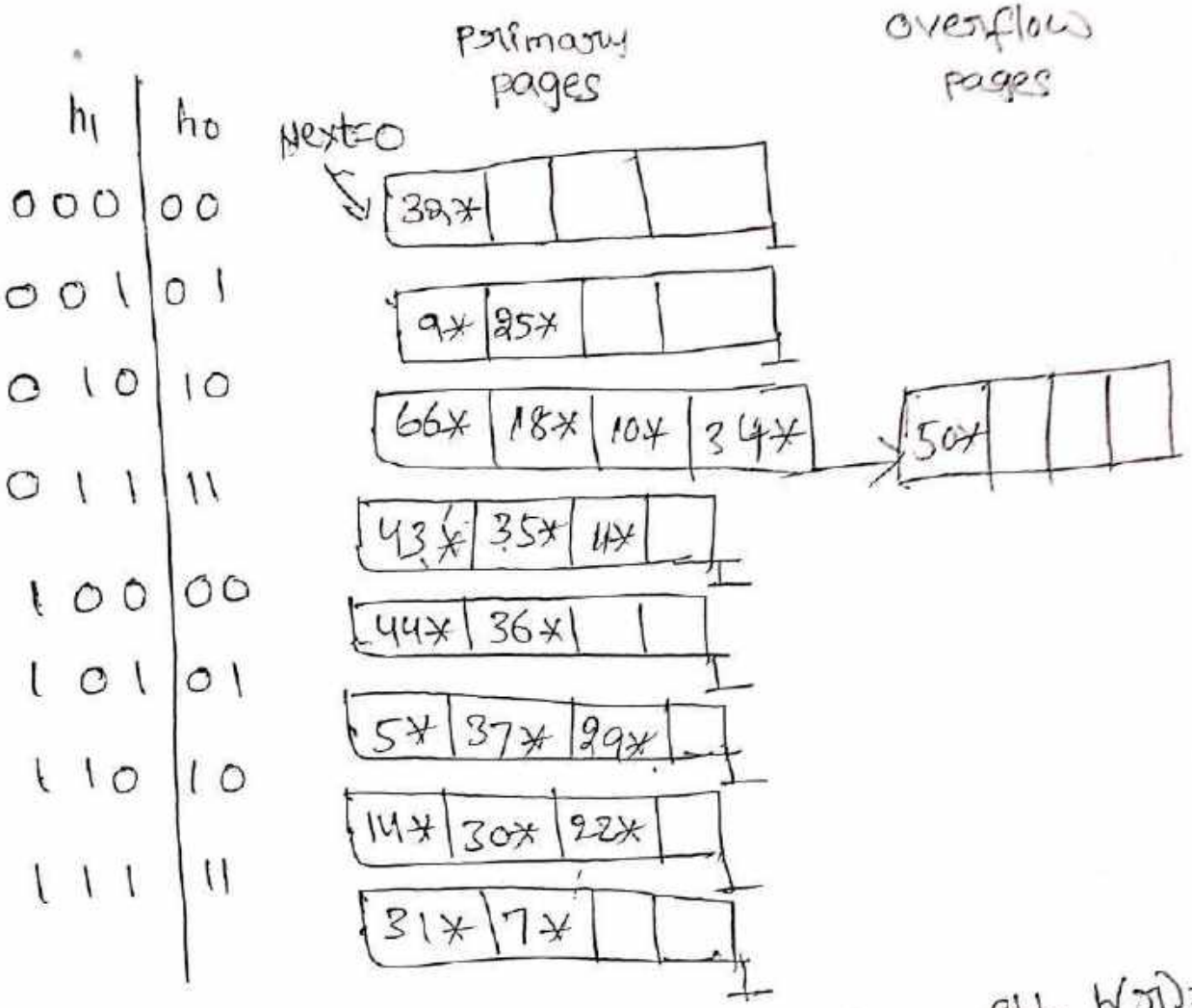


Fig 13 After Inserting Record r with h(r)=50

In average is about 1.2 disk accesses for reasonably uniform data distribution. (The cost can be considerably worse — linear in the no of data entries in the file — if the distribution is very skewed, the space utilisation is also very poor with skewed data distribution.)

34) Inserts require reading & writing a single page, unless split is triggered.

35) Deletion is essentially the inverse of insertion.

Scanned by CamScanner

(66) If the last bucket in the file is empty, it can be removed and Next can be decremented.

(67) (If Next is 0 and last becomes empty, Next is made to point to bucket $(M/2)-1$, where M is the current no' of buckets, Level is decremented, and empty bucket is removed.)

(68) If we wish, we can combine the last bucket with its split image even when it is not empty, using some criterion to trigger this merging in essentially the same way. (69) The criterion is typically based on the occupancy of the file, and merging can be done to improve space utilization.

Extendible VS Linear Hashing:—

(1) To understand the relationship b/w Linear Hashing and Extendible Hashing imagine that we also have a directory in Linear Hashing with elements 0 to N-1. (2) The 1st split is at bucket 0, and so we add director element N. (3) In principle, we may imagine that the entire directory has been denoted at this point, because element 1 is the same as element N+1, element 2 is the same as N+2, and so on, we can avoid the actual copying for the rest of directory.

(4) The 2nd split occurs at bucket 1; now directory element N+1 becomes significant and is added.

(5) At the end of the round, all the original N bucket one split,

Scanned by CamScanner

and the directory is doubled in size (because all elements point to distinct buckets).

(6) We observe that the choice of hashing functions is actually very similar to what goes on in Extendible hashing - in effect, moving from h_i to h_{i+1} in linear hashing corresponds to doubling the directory in Extendible Hashing.

(7) Both operations double the effective range into which key values are hashed; but where as the directory is doubled in a single step of Extendible Hashing, moving from h_i to h_{i+1}, along with a corresponding doubling in the no' of buckets, occurs gradually over the course of a round in Linear Hashing.

(8) The new idea behind Linear hashing is that a directory can be avoided by a clever choice of the bucket to split. (9) On the other hand, by always splitting the appropriate bucket. (10) Extendible Hashing may lead to reduced number of splits and higher bucket occupancy.

(11) The directory analogy is useful for understanding the ideas behind Extendible and Linear Hashing.

(12) However, the directory structure can be avoided for Linear Hashing. (but not for Extendible Hashing)

Scanned by CamScanner

by allocating primary bucket pages consecutively, which would allow us to locate the page for bucket i by a simple offset calculation.

(13) For uniform distributions, this impl^n of Linear Hashing has a lower average cost for equality selections (because the directory level is eliminated).

(14) For skewed distributions, the impl^n could result in any empty or nearly empty buckets, each of which is allocated at least one page, leading to poor performance relative to Extendible Hashing, which is likely to have higher bucket occupancy.

(15) A different impl^n of Linear Hashing, in which a directory is actually maintained, offers the flexibility of not allocating one page per bucket, null directory elements can be used as in Extendible Hashing. However this implementation introduces the overhead of directory level and could prove costly for large, uniformly distributed files. (Also, although this impl^n alleviates the potential problem of low bucket occupancy by not allocating pages for empty buckets, it is not a complete solution because we can still have many pages with very few entries.)

Scanned by CamScanner

for each data entry, fetch the corresponding data record from the underlying file, obtaining data records in the sort order ⟨age, sal⟩.

i) → We can read all data entries at a cost of 1.1·B(D+6.7RC) I/Os.

→ Now comes the expensive part: we have to fetch the employee record for each data entry in the index.

⇒ The cost of fetching the employee records is one I/O per record, since the index is unclustered and each data entry on leaf page of the index could point to a different page in the employee file.

→ The cost of this step is BR(D+C), which is prohibitively high.

⇒ If we want the employee records is sorted order, we would be better off ignoring the index and scanning the employee file directly, and then sorting it.

⇒ A simple rule of thumb is that a file can be sorted by a two-pass algorithm in which each pass requires reading & writing the entire file. Thus, the I/O cost of sorting a file with B pages is 4B, which is much less than the cost of using an unclustered index.

Scanned by CamScanner

Search with Equality Selection:-

(1) We assume that equality selection matches the search order (age, sal).

→ We can locate the 1st page containing the desired data entry or entries, should any qualifying entries exist, in $\log_F 0.15B$ steps, that is, by fetching all pages from the root to the appropriate leaf.

→ Each step requires a disk I/O and two comparisons.

→ Once the page is known, the 1st qualifying data entry can again be located by a binary search of the page at a cost of $C \log_2 6.7R$.

→ The 1st qualifying data record can be fetched from the employee file with another I/O.

→ The cost is $D \log_F 0.15B + C \log_2 6.7R + D$, which is a significant improvement over searching sorted files.

(2) If several records qualify (Ex- "Find all employees aged 18"), they are not guaranteed to be adjacent to each other.

The cost of retrieving all such records is the cost of locating the 1st qualifying data entry ($D \log_F 0.15B + C \log_2 6.7R$) plus one I/O per qualifying record. The cost of using an unclustered index is therefore very dependent on the no' of qualifying

Scanned by CamScanner

earch with Range Selection:—

>Again assuming that the range selection matches
ie composite key, the 1st record that satisfies the
lection is located as it is for search with equality.
Subsequently, data entries are sequentially
etrieved (using the next and previous links at the
af level of the index) until a data entry is found
hat does not satisfy the range selection.
>For each qualifying data entry, we incur one
lo to fetch the corresponding employee records.
The cost can quickly become prohibitive as the
o' of records that satisfy the range selection
ncreases.
>As a rule of thumb, if 10 percent of data
ecords satisfy the selection condition, we are better
ff retrieving all employee records, sorting them,
nd then retaining those that satisfy the selection.

Insert:— we must 1st insert the record in the
mployee file, at a cost of 2D+C.
In addition, we must insert the corresponding
data entry in the index.
Finding the right leaf page costs $D \log_F 0.15B +$
$C \log_2 6.7R$, and writing it out after adding
the new data entry costs another D.

Scanned by CamScanner

Delete:- we need to locate the data record for the employee file and the data entry in the index and this search step costs $D \log_F 0.15B + C \log_2 6.7R + D$.
→ Now, we need to write out the modified pages in the index and the data file, at a cost of 2D.

Heap File with Unclustered Hash Index:-

(1) As for unclustered tree indexes, we assume that each data entry is one tenth the size of data record. We consider only static hashing in our analysis, and for simplicity we assume that there are no overflow chains.

(2) In static hashed file, pages are kept about 80 percent occupancy (to leave space for future insertions and minimise overflows as the file expands).
→ This is achieved by adding a new page to a bucket when each existing page is 80 percent full, when records are initially loaded into a hashed file structure.
→ The no' of pages required to store data entries is therefore 1.25 times the no' of pages when the entries are densely packed, that is, $1.25(0.10B) = 0.125B$.
→ The no' of data entries that fit on a page is $10(0.80R) = 8R$, taking into account the relative size and occupancy—

Scanned by CamScanner

Scan:- As for an unclustered tree index, all data entries can be retrieved inexpensively, at a cost of 0.125 B(D+8RC) I/Os.

However, for each entry, we incur the additional cost of one I/O to fetch the corresponding data record; the cost of this step is BR(D+C). This is prohibitively expensive, and further, results are unordered. So no one ever scans a hash index.

Search with Equality Selection:-

1) This operation is supported very efficiently for matching selections, that is, equality conditions are specified for each field in the composite search key (age, sal).

⇒ The cost of identifying the page that contains qualifying data entries is H.

⇒ Assuming that this bucket consists of just one page (i.e, no overflow pages), the cost of scanning the page is 0.5(8R) C = 4RC.

⇒ Finally, we have to fetch the data record from the employee file, which is another D.

⇒ The total cost is therefore H+2D+4RC, which is even lower than the cost for a tree index.

2) If several records qualify, they are not guaranteed to be adjacent to each other.

Scanned by CamScanner

→ The cost of retrieving all such records is the cost of locating the 1st qualifying data entry (H+D+C), plus one I/O per qualifying record.
→ The cost of unclustered index therefore depends heavily on the no of qualifying records.

Search with Range Selection:-

The hash structure offers no help, and the entire heap file of employee records must be scanned at a cost of B(D+RC).

Insert:- we must 1st insert the record in the employee heap file, at a cost of 2D+C.
In addition, the appropriate page in the index must be located, modified to insert a new data entry, and then written back. The additional cost is H+2D+C.

Delete:- We need to locate the data record in the employee file and the data entry in the index; this search step costs H+2D+4RC. Now, we need to write out the modified pages in the index and the data file, at a cost of 2D.

Scanned by CamScanner

Comparision of I/O costs:-

Below Fig 4 compares I/O costs for the various file organisations.

File Type	Scan	Equality Search	Range Search	Insert	Delete.
Heap	BD	0.5BD	BD	2D	Search + D
Sorted	BD	$D \log_2 B$	$D \log_2 B$ + # matching pages	Search + BD	Search + BD
Clustered	1.5BD	$D \log_F 1.5B$	$D \log_F 1.5B$ + # matching pages	Search + D	Search + D
Unclustered tree index	BD(R + 0.15)	$D(1 + \log_F 0.15B)$	$D(\log_F 0.15B$ + # matching records)	$D(3 + \log_F 0.15B)$	Search + 2D
Unclustered hash index	BD(R + 0.125)	2D	BD	4D	Search + 2D

A Comparision of I/O costs

(1) A heap file has good storage efficiency and supports fast scanning and insertion of records. However, it is slow for searches and deletions.

(2) A sorted file also offers good storage efficiency, but insertion and deletion of records is slow. Searches are faster than in heap files. It is worth noting that, in a real DBMS, a file is almost

Scanned by CamScanner

never kept fully sorted.

(3) A clustered file offers all the advantages of a sorted file and supports inserts and deletes efficiently. (There is a space overhead for these benefits, relative to a sorted file, but the trade-off is well worth it.)

Searches are even faster than in sorted files, although a sorted file can be faster when a large no of records are retrieved sequentially, because of blocked I/O efficiencies.

(4) Unclustered tree and hash indexes offer fast searches, insertion, and deletion, but scans and range searches with many matches are slow.

Hash indexes are better on equality searches, but they do not support range searches.

(5) In summary, Fig 4 demonstrates that no one file organisation is uniformly superior in all situations.

Scanned by CamScanner