RELATIONAL MODEL
UNIT-III

Introduction to Relational Model, Basic Structure, Database Schema, Relational Algebra, Relational Calculus. File Organization and Indexing: Introduction, Types of File Organizations, Overview of Indexes, Types of Indexes, Index Data Structures, Tree structured Indexing, Hash based Indexing.

INTRODUCTION TO RELATIONAL MODEL

Relational data model is the most popular data model used widely around the world for data storage. In this model data is stored in the form of tables.

Relational Model Concepts

Table is also called Relation. Let the below table name be SUDENT_DATA

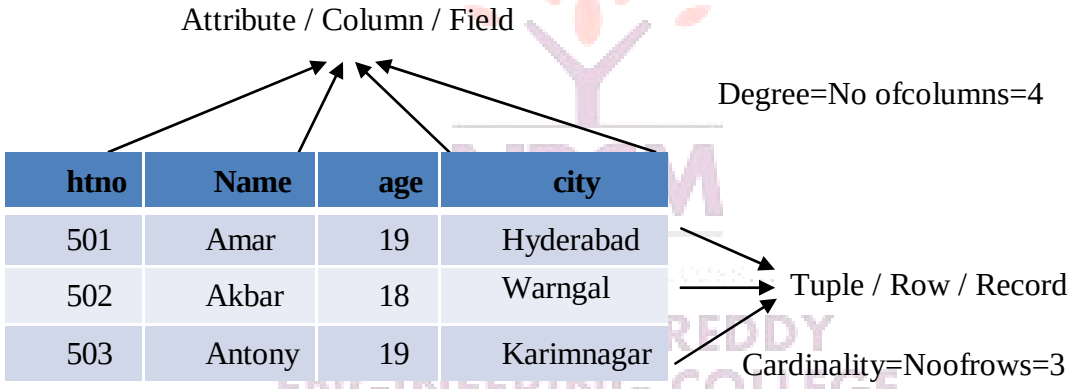


Table: In relational model the data is saved in the form of tables. A table has two properties rows and columns. Rows represent records and columns represent attributes.

Attribute: Each column in a Table is an attribute. Attributes are the properties that define a relation. e.g., HTNO, NAME, AGE, CITY in the above relation.

Tuple: Every single row of a table is called record or tuple.

Relation Schema: It represents the name of the relation (Table) with its attributes.

Eg., STUDENT_DATA(htno, name, age, city)

Degree: The total number of attributes in the relation is called the degree of the relation.

Cardinality: Total number of rows present intheTable.

RELATIONAL ALGEBRA

Relational Algebra is procedural query language, which takes Relation as input and generates relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

Operator Symbol	Operator Name	Explanation
π	Projection	Select column names
σ	Selection	Select row values
ρ	Renaming	Rename a table name or expression results
$\cup$	Union	Perform union operation
$\cap$	Intersection	Perform inter section operation
-	Setdeference	Perform set difference operation
$\times$	Cartesianproduct	Every row of first table is joined with every row of second table
$\bowtie$	Join	Join two tables based on some condition

i. **Select Operation(σ):** It select stuples that satisfy the given predicate from a relation.
Notation: $\sigma_p(r)$

where σ stands for selecting tuples (rows) and r stands for relation (table) name. *p* is prepositional logic formula which may use connectors like ***and, or***, and ***not***. These termsmay use relational operators like $=, \neq, \geq, <, >, \leq$.

Example1: $\sigma_{subject="database"}$(Books)
Output: Selects rowswhosesubjectis'database'frombooks table.

Example2: $\sigma_{subject="database" and price="450"}$(Books)
Output: Selects rowsfrombookswheresubjectis'database'and'price'is450.

Example3: $\sigma_{subject="database" and price="450" or year>"2010"}$(Books)
Output: Selects rows from books where subject is'database'and'price'is450orthose books published after 2010.

ii. **Project Operation (∏):**It projects column(s)thatsatisfyagiven predicate.
Notation: $\prod_{A1,A2,\ldots An}(r)$

whereA_1, A_2,$A_n$are column(attribute)namesofrelation**r**.Duplicaterowsare automatically eliminated in the output.

Example: $\prod_{subject, author}$ (Books)

Displayvalues fromcolumnssubjectand authorfrom therelation Books.

iii. **Union Operation (∪):**It performs union operation between two given relations.Itcombines rows from two given relations.

Notation: rUS

Where **r** and **s** are either database relations or relation result set (temporary relation). **r**U **S** returns a relation instance containing all tuples that occur in either relation instance **r** or relation instance **S**(or both). For a union operation to be valid, the following conditions must hold:

- **R**and**s** must have the same number of attributes.
- Attribute domains must be compatible in **r**and**s**.

Example: $\prod_{author}(Books) \cup \prod_{author}(Articles)$

Output: Projects the names of the authors who have either written a book or an article or both.

iv. **Intersection Operation (∩):** It performs intersection operation between two given relations . It collect only rows which are common in the two given relations.

Notation: $R \cap S$

$R \cap S$ returns a relation instance containing all tuples that occur in *both R* and *S*. The relations *R* and *S* must be union-compatible, and the schema of the result is defined to be identical to the schema of *R*.

$\prod_{author}(Books) \cap \prod_{author}(Articles)$

Output: Projects the names of the authors who have written bothbook and an article.

v. **Set Difference (−):** It finds tuples which are present in one relation but not in the second relation.

Notation: r−s

Finds all the tuples that are present in **r** but not in**s**.

Example: $\prod_{author}(Books) - \prod_{author}(Articles)$

Output− Provides the name of authors who have written books but not articles.

vi. **Cartesian Product (X):** It returns a relation instance whose schema contains all the fields of table-1 (in the same order as they appear in table-1) followed by all the fields of table-2. It combines every row in first table with every row in the second table.

Notation: r X **s**

Where **r**and **s**are relations and their output will be defined as :r Xs={ qt|q∈r and t ∈s}

vii. **Natural join(⋈):**The most 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 form. The operation is defined as follows:

$R \bowtie_c S = \sigma_c(R \times S)$

Thus $\bowtie$ is defined to be a cross-product followed by a selection.Note that thec ondition c can refer to attributes of both R and *S.*

Note: *If thecondition c in* $R \bowtie_c S$ *contain equal operator, then itis called equi-join*

viii. Natural Join($\bowtie$): In this case, we can simply omit the join condition; the default is that the join condition is a collection of equalities on all common fields. We call this special case as natural join, and it has the nice propertythat the result is guaranteed not to have two fields with the same name.

ix. Rename Operation (ρ): The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ.

Notation: ρ(temp,E)

Where the result of expression **E** is saved with name of ***temp***.

x. Division (/): Consider two relation instances *A* and *B* in which *A* has (exactly) two fields *x* and *y* and *B* has just one field *y*, with the same domain as in *A*. We define the *division* operation *A / B* as the set of all *x* values (in the form of unary tuples) such that for *every y* value in (a tuple of) *B*, there is a tuple *(x,y)* in *A*.

Example:

A

SNO	PNO
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

PNO
P2

B2

PNO
P2
P4

B3

PNO
P2
P4

A/B1

SNO
S1
S2
S3
S4

A/B2

SNO
S1
S4

A/B3

SNO
S1

SampleQueries: We present a number of sample queries using the following schema:

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

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

(*sid:* integer, *bid:* integer, *day:* date)

The key fields are underlined, and the domain of each field is listed after the field name. Thus *sid* is the key for Sailors, *bid* is the key for Boats, and all three fields together form thekeyforReserves. Fields in an instanceofoneoftheserelations will bereferred to by name, or positionally, using the order in which they are listed above.

(Q1) Find the names of sailors who have reserved boat 103.

This query can be written as follows:

$\pi_{sname}((\sigma_{bid=103}Reserves)\bowtie Sailors)$

We first compute the set of tuples in Reserves with *bid* = 103 and then take the natural join of this set with Sailors. This expression can be evaluated on instances of Reservesand Sailors. Evaluated on the instances *R*2 and *S*3, it yields a relation

(Q2) Find the names of sailors who have reserved are d boat.

$$\pi_{sname}((\sigma_{color='red'}Boats)\bowtie Reserves\bowtie Sailors$$

This query involves a series of two joins. First we choose (tuples describing) red boats. Then, we join this set with Reserves (natural join, with equality specified on the **bid** column) to identify reservations of red boats. Next, we join the resulting intermediate relation with Sailors (natural join, with equality specified on the **sid** column) to retrieve the names of sailors who have rnade reservations for red boats. Finally, we project the sailors' names.

(Q3) Find the colors of boats reserved by Lubber.

$$\pi_{color}((\sigma_{sname='Lubber'}Sailors)\bowtie Reserves\bowtie Boats)$$

(Q4) Find thenames ofsailors whohavereserved atleast oneboat.

$$\pi_{sname}(Sailors\bowtie Reserves)$$

The join of Sailors and Reserves creates an intermediate relation in which tuples consist of a Sailors tuple 'attached to' a Reserves tuple. A Sailors tuple appears in (some tuple of) this intermediate relation only if at least one Reserves tuple has the same sid value, that is, the sailor has made some reservation.

(Q5) Find the names of sailors who have reserved a red or a green boat.

$$\rho(Tempboats,(\sigma_{color='red'}Boats)U(\sigma_{color='green'}Boats))$$

$$\pi_{sname}(Tempboats\bowtie Reserves\bowtie Sailors)$$

We identify the set of all the rows that are either red or green from boats table. We rename this resultas Tempboats.Then we joinTemp boats with Reservestoidentifysid'sofsailors.Finally, we join with Sailors to find the names of Sailors with those sids.

(Q6) Find the names of sailors who have reserved a red and a green boat

$$\rho(Tempboats2,(\sigma_{color='red'}Boats)\cap(\sigma_{color='green'}Boats))$$

$$\pi_{sname}(Tempboats2\bowtie Reserves\bowtie Sailors)$$

However, this solution is incorrect-it instead tries to compute sailors who have reserved a boat that Is both red and green. A boat can be only one color;this query will always return an empty answer set. The right answer is

$$\rho(T\ empred,\ \pi_{sid}((\sigma_{color='red'}Boats)\bowtie Reserves))$$

$$\rho(Tempgreen,\pi_{sid}((\sigma_{color='green'}Boats)\bowtie Reserves))$$

$$\pi_{sname}((Tempred\cap Tempgreen)\bowtie Sailors)$$

The two temporary relations compute the sids of sailors, and their intersection identifiessailors who have reserved both red and green boats.

(Q7) Find the names of sailors who have reserved at least two boats.

$$\rho(Reservations,\ \pi_{sid,sname,bid}(Sailors\bowtie Reserves))$$

$$\rho(Reservationpairs(1\rightarrow sid1,2\rightarrow sname1,3\rightarrow bid1,4\rightarrow$$

$$sid2,5\rightarrow sname2,6\rightarrow bid2),Reservations\times Reservations)$$

$$\pi_{sname1}\sigma_{(sid1=sid2)\cap(bid1=bid2)}Reservationpairs$$

First, we compute tuples of the form (sid, sname, bid), where sailor sid has made a reservationfor boat bid; this set of tuples is the temporary relation Reservations. Next we find all pairs of Reservations tuples where the same sailor has made both reservations and the boats involved are distinct. Here is the central idea: To show that a sailor has reserved two boats, we must find two Reservations tuples involving the same sailor but distinct boats. Finally, we project the names of such sailors.

(Q8) Find the sids of sailors with age over 20 who have not reserved ared boat.

$$\pi_{sid}(\sigma_{age>20}Sailors)-\pi_{sid}((\sigma_{color='red'}Boats)\bowtie Reserves\bowtie Sailors)$$

This query illustrates the use of the set-difference operator. Again, we use the fact that *sid* is

The key for Sailors.We first identify sailors aged over 20 instances and then discard those who have reserved a red boat to obtain the answer.

(Q9) Find the names of sailors who have reserved all boats.

The use of the word *all*(or*every*)is a good indication that the division operation might be applicable:

$$\rho(Tempsids, (\pi_{sid,bid}Reserves)/(\pi_{bid}Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

(Q10) Find the names of sailors who have reserved all boats called Interlake.

$$\rho(Tempsids, (\pi_{sid,bid}\ Reserves)/(\pi_{bid}(\sigma_{bname='Interlake'}\ Boats)))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

RELATIONAL CALCULUS

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

TupleRelationalCalculus

Tuple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do.

In Tuple Relational Calculus, a query is expressed as `{t|P(t)}`

Where t=resulting tuples,P(t)=known as Predicate and these are the conditions that are used to fetch t. Thus, it generates set of all tuples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR(∨),AND(∧),NOT(¬). It also uses quantifiers:

∃t ∈r(Q(t))=”there exists”a tuple int in relationr such that predicate Q(t) is true.

∀t∈r(Q(t))=Q(t)is true“for all”tuples in relation r.

(Q12) Find the names and ages of sailors with a rating above7.

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

This queryillustrates a useful convention: *P* is considered to be a tuple variable with exactly two fields, which are called *name* and *age*.

(Q13) Find the sailorname, boatid, and reservation date for each reservation

$\{P \mid \exists R \in Reserves\ \ \exists S \in Sailors$

$(R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname)\}$

(Q1) Find the names of sailors who have reserved boat103. *(similarquestionQ1from relational algebra)*

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

This query can be read as follows: "Retrieve all sailor tuples for which there exists a tuple in Reserves, having the same value in the *sid* field, and with *bid* = 103."

(Q2) Find the names of sailors who have reserved a red boat. *(similarquestionQ2from relational algebra)*

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

$\wedge \exists B \in Boats(B.bid = R.bid \wedge B.color = 'red'))\}$

This query can be read as follows:"Retrieveallsailortuples *S*forwhichthereexisttuples *R*in Reserves and *B* in Boats such that *S.sid = R.sid*, *R.bid = B.bid*, and *B.color =* $'red'$."

(Q7) Find the names of sailors who have reserved atleast twoboats. *(similarquestionQ7 from relational algebra)*

$\{P \mid \exists S \in Sailors \exists R1 \in Reserves \exists R2 \in Reserves\ (S.sid = R1.sid$

$\exists R1.sid = R2.sid \wedge R1.bid \neq R2.bid \wedge P.sname = S.sname)\}$

(Q9) Find the names of sailors who have reserved all boats. *(similarquestionQ9from relational algebra)*

$\{P \mid \exists S \in Sailors\ \ \forall B \in Boats$

$(\exists R \in Reserves\ (S.sid = R.sid \wedge R.bid = B.bid \wedge P.sname = S.sname))\}$

(Q14) Find sailorswhohavereserved allred boats.

$\{S \mid S \exists Sailors \in \forall B \in Boats$

$(B.color='red' => (\exists R \in Reserves(S.sid=R.sid \wedge R.bid=B.bid)))\}$

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 whosedomain is the setofintegers).

ADRC query has the form $\{ \langle x_1, x_2, \ldots, x_n \rangle \mid p(\langle x_1,x_2,.. ., x_n \rangle)\}$

where each$x_i$is either a *domain variable*or a constant and $p(\langle x_1,x_2,.. ., x_n \rangle)$ denotes a DRC formula whose only free variables are the variablesamong the x_i, $1 \leq i \leq n$. The result of this query is the set of all tuples $\langle x_1, x_2,.. .,x_n \rangle$ for which the formula evaluates to true.

A DRC formula is defined in a manner very similar to the definition of a TRC formula. The main difference is that the variables are now domain variables. Let op denote an operatorin theset $\{<, >, =, \leq, \geq, \neq\}$ and let X and Ybedomain variables. Anatomicformulain DRC is one of the following:

- $(x_1, x_2, \ldots, x_n) \in$Rel, where Rel is a relation with n attributes; each x_i, $1 \leq i \leq$ n is either a variable or a constant
- X op Y
- X op*constant*, orconstantop X

A formula is recursively defined to be one of the following, where P and q are themselves formulas and p(X) denotes a formula in which the variable X appears:

- anyatomic formula
- ┐p,P/\q, P V q, orp=>q
- ∃X(p(X)), whereX isadomain variable
- ∀X(p(X)), whereX isa domain variable

(Q1) Find the names of sailors who have reserved boat 103.

$\{ (N) \mid \exists I,T,A (\langle I,N, T,A \rangle \in Sailors$

$\wedge \exists Ir,Br,D(\langle Ir,Br,D \rangle \in Reserves \wedge Ir=I \wedge Br= 103))\}$

(Q2) Find the names of sailors who have reserved a red boat.

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

(Q7) Find the names of sailors who have reserved at least two boats.

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

(Q9) Find thenames of sailors who have reserved all boats.

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

File Organization and Indexing:

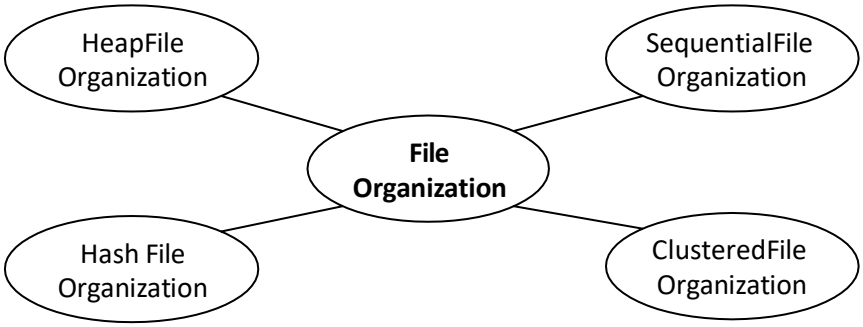
FILE ORGANIZATION

The database is stored as a collection of files. Each file contains a set of records. Each record is a collection of fields. For example, a student table (or file) contains many records and each record belongs to one student with fields (attributes) such as Name, Date of birth, class, department, address, etc.

File organization defines how file records are mapped onto disk blocks.

The records of a file are stored in the disk blocks because a block is the unit of data transfer between disk and memory. When the block size is larger than the record size, each block will contain more than one record. Sometimes, some of the files may have large records that cannot fit in one block. In this case,we can store part of a record on one block and the rest on another. A pointer at the end of the first block points to the block containing the remainder of the record.

The different types of file organization are given below:



Heap File Organization: When a file is created using Heap File Organization mechanism, the records are stored in the file in the order in which they are inserted. So the new records are inserted at the end of the file. In this type of organization inserting new records is more efficient. It uses linear search to search records.

Sequential File Organization: When a file is created using Sequential File Organization mechanism, all the records are ordered (sorted) as per the primary key value and placed in the file. In this type of organization inserting new records is more difficult because the records need to be sorted after inserting every new record. It uses binary search to search records.

Hash File Organization: When a file is created using Hash File Organization mechanism, ahash function is applied on some field of the records to calculate hash value. Based on the hash value, the corresponding record is placed in the file.

Clustered File Organization: Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in a same disk block, that is, the ordering of records is not based on primary key or search key.

OVER VIEW OF INDEXES:

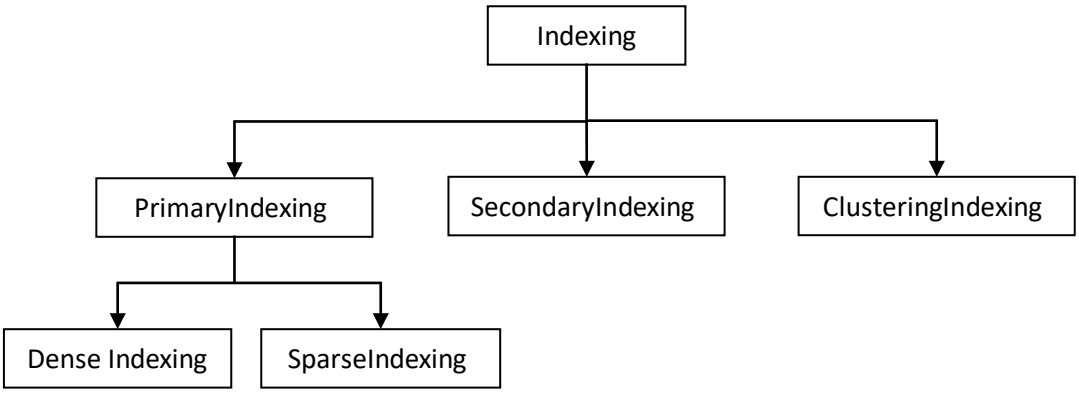
INDEXING

If the records in the file are in sorted order, then searching will become very fast. But, in most of the cases, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. It indicates, the records are not in sorted order. In order to make searching faster in the files with unsorted records, indexing is used.

Indexing is a data structure technique which allows you to quickly retrieve records from a database file. An Index is a small table having only two columns. The first column contains a copy of the primary or candidate key of a table. The second column contains a set of disk block addresses

where the record with that specific key value stored.

Indexing in DBMScan be of the following types:



i. Primary Index

- If the index is created by using the primary key of the table, then it is known as primary indexing.
- As primary keys are unique and are stored ina sorted manner, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: dense index and sparse index.

Dense index

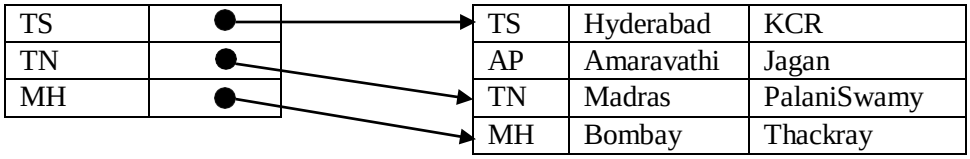
- If every record in the table has one index entry in the index table, then it is called denseindex.
- In this, the number of records (rows) in the index table is same as the number of records (rows) in the main table.
- Aseveryrecord hasoneindexentry, searchingbecomes faster.

TS	●	TS	Hyderabad	KCR
AP	●	AP	Amaravathi	Jagan
TN	●	TN	Madras	PalaniSwamy
MH	●	MH	Bombay	Thackray

Sparse index

- If only few records in the table have index entries in the index table, then it is called sparse index.
- In this, the number of records (rows) in the index table is less than the number of records (rows) in the main table.
- Asnotalltherecordhaveindex entries, searching becomes slowforrecordsthatdoesnot have

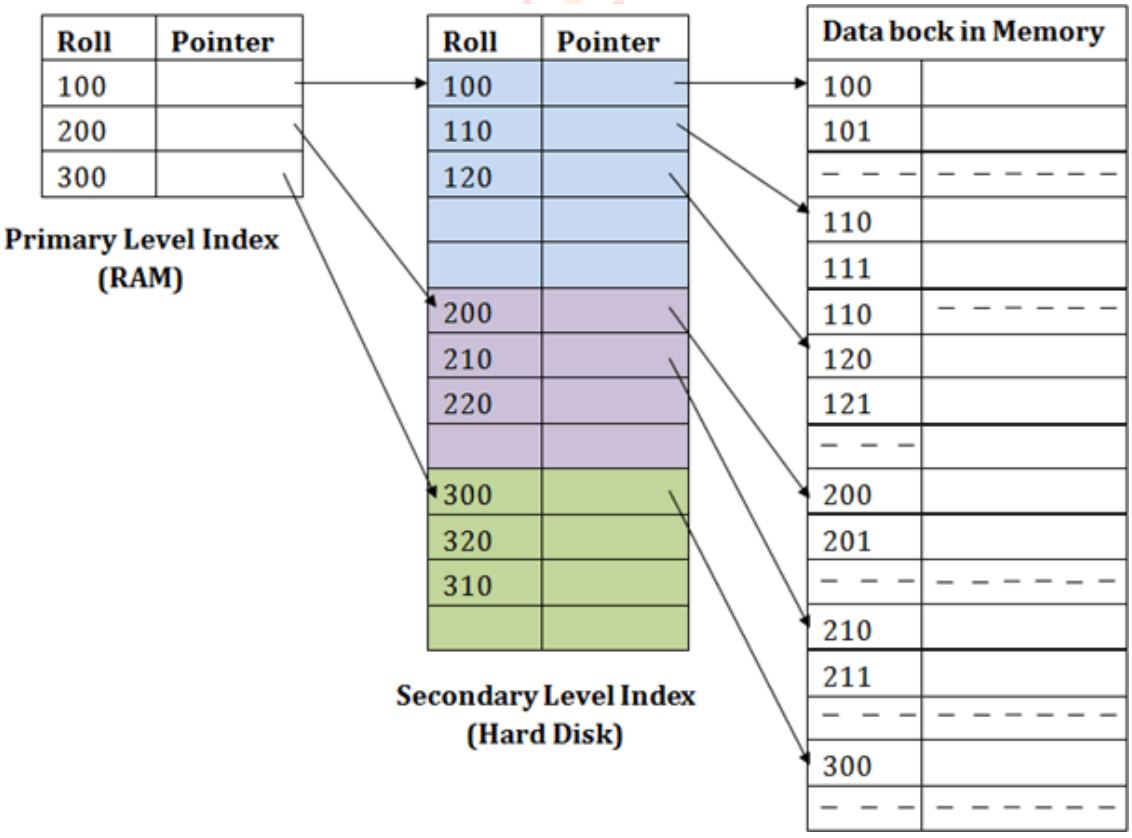
index entries.



ii. Secondary Index

When the sizeofthe main table grows, then sizeofindex tablealso grows. If theindex tablesize grows then fetching the address itself becomes slower. To overcome this problem, secondary indexing is introduced.

- Insecondaryindexing, toreducethesizeofmapping, another levelof indexingis introduced.
- Itcontainstwolevels. Inthefirstleveleachrecordinthemaintablehasoneentryinthefirst- level index table.
- The index entries in the fisrt level index table are divided into different groups. For each group, one index entry is created and added in the second level index table.

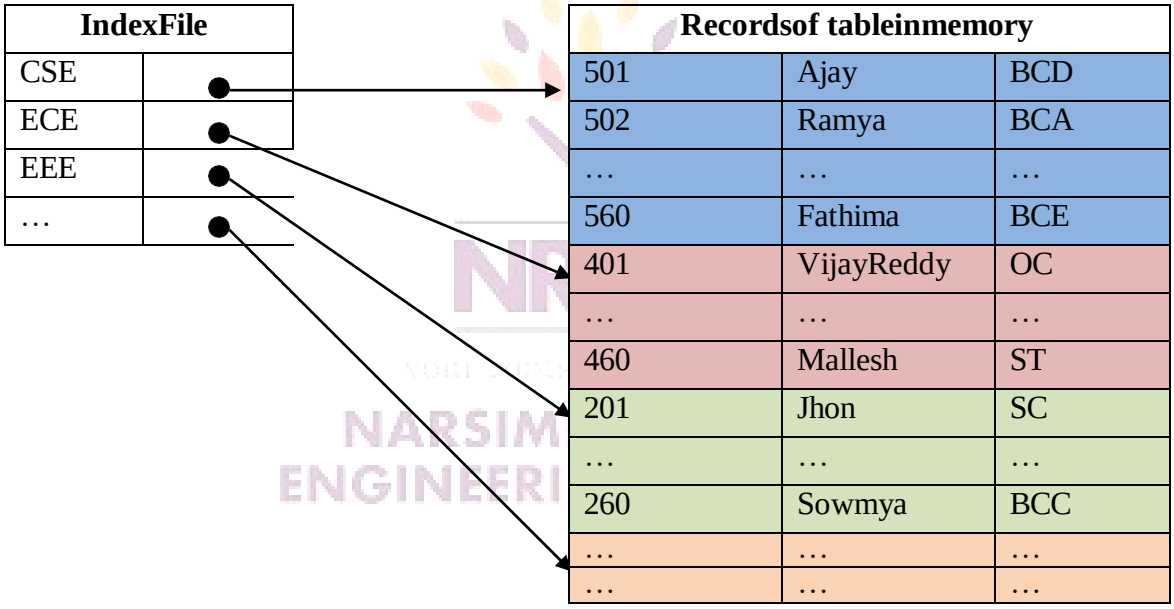


Multi-level Index: When the maintable size becomes too large, creating secondary level index improves the search process.Even if the search process is slow; we can add one more level of indexing and so on. This type of indexing is called multi-level index.

iii. Clustering Index

- Sometimestheindexiscreatedonnon-primarykeycolumnswhichmaynotbeunique for each record.
- In this case, to identify their cord faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are createdfor these group.

Example: Consider acollegecontainsmanystudentsineachdepartment. Allthestudentsbelong to the same Dept_ID are grouped together and treated as a single cluster. One index pointers point to the one cluster as a whole. The idex pointer points to the first record in each cluster. Here Dept_ID is a non-unique key.

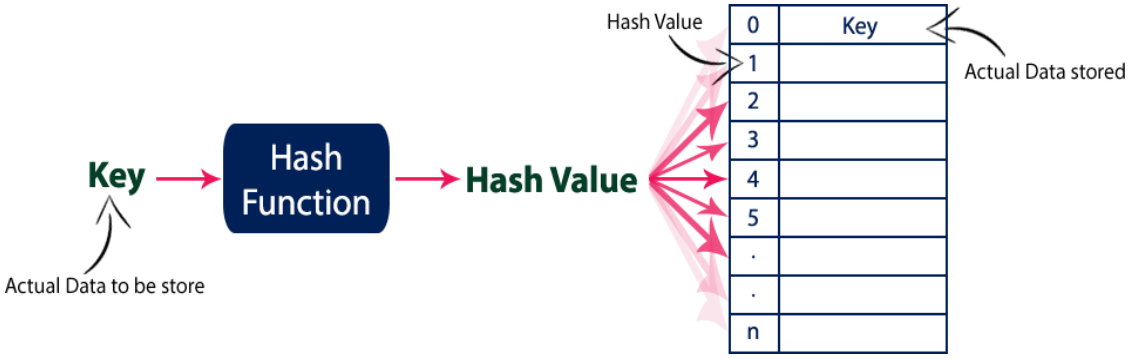


In above diagram we can see that, indexes are created for each department in the index file. In the data block, the students of each department are grouped together to form the cluster. The address in the index file points to the beginning of each cluster.

2. HASHBASED INDEXING

Hashing is a technique to directly search the location of desired data on the disk without using index structure. Hash function is a function which takes a piece of data (key) as input and produces a hash valueas output which maps the data to a particular location in the hash table.

Theconcept of hashingandhash table is shownin thebelow figure



There aremainlytwo typesof hashing methods:

i. Static Hashing

ii. DynamicHashing

- Extendedhashing
- Linearhashing

3. STATIC HASHING

In static hashing, the hash function produces only fixed number of hash values. For example, consider the hash function

f(x)= x mod 7

Foranyvalueof***x***, theabovefunctionproducesoneof the hash value from {0,1,2, 3,4, 5,6}. It means static hashing maps search-key values to a fixed set of bucket addresses.

Example: Inserting10,21,16and12inhash table.

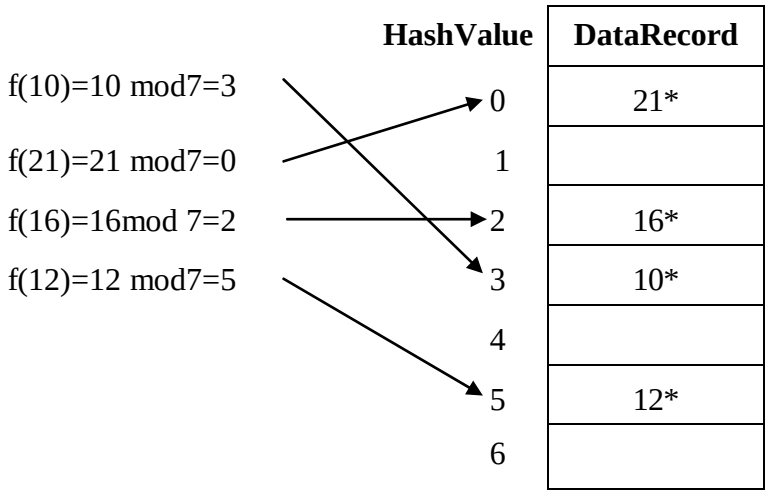


Figure5.1: *Static hashing*

Suppose, latter if we want to insert 23, it produce hash value as 2 (23 mod 7 = 2). But, in the above hash table, the location with hash value 2 is not empty (it contains 16*). So, a collision occurs. To resolve this collision, the following techniques are used.

- Open addressing
- SeparateChainingorClosed addressing

i. Open Addressing:

Openaddressingisacollisionresolvingtechniquewhichstoresallthekeysinsidethe hash table. No key is stored outside the hash table. Techniques used for open addressing are:

- LinearProbing
- QuadraticProbing
- DoubleHashing

➢ LinearProbing:

In linear probing, when there is a collision, we scan forwards for the next empty slot to fill the key's record. If you reach last slot, then start from beginning.

Example: Consider figure 1. When we try to insert 23, its hash value is 2. But the slotwith 2 is not empty. Then move to next slot (hash value 3), even it is also full, then move once again to next slot with hash value 4. As it is empty store 23 there. This is shown in the below diagram.

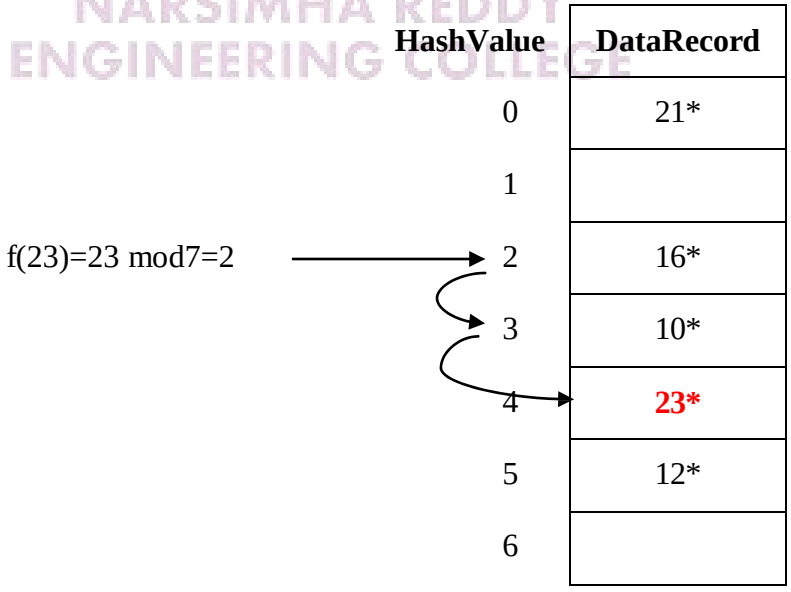


Figure5.2: *Linear Probing*

- **QuadraticProbing:**

In quadratic probing, when collision occurs, it computer new hash value by taking the original hash value and adding successive values of quadratic polynomial until an open slot is found. If here is a collision, it use the following hash function: $h(x) = (f(x) + i^2) \bmod n$,where I = 1, 2, 3, 4,….. and f(x) is initial hash value.

Example: Consider figure 1. When we try to insert 23, its hash value is 2. But the slot with hash value 2 is not empty. Then compute new hash value as $(2 + 1^2) \bmod 7 = 3$, even it is also full, and then once again compute new hash value as $(2 + 2^2) \bmod 7 = 6$. As it is empty store 23 there. This is shown in the below diagram.

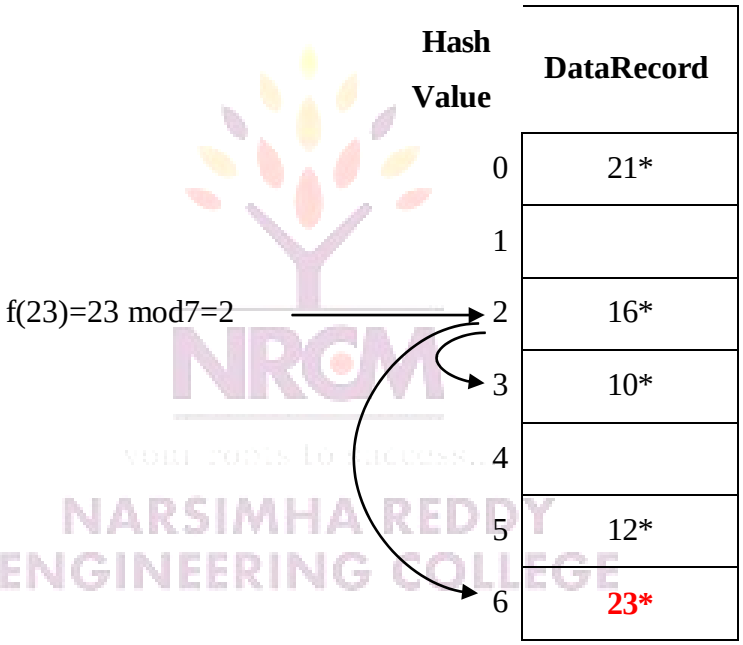


Figure5.3: *Quadratic Probing*

- **DoubleHashing**

In double hashing, there are two hash functions. The second hash function is used to provide an offset value in case the first function causes a collision. The followingfunction is an example of double hashing: (firstHash(key) + i * secondHash(key)) % tableSize. Use i = 1, 2, 3, …

Apopular secondhash function is : secondHash**(key)=PRIME–(key%PRIME)**

wherePRIMEisaprimesmallerthantheTABLE_SIZE.

Example: Consider figure1.Whenwetry toinsert23, its hash value is 2. But the slot with hash value 2 is not empty. Then compute double hashing value as

secondHash (key)=PRIME–(key%PRIME) → secondHash (23)=5–(23 %5)=2

Doublehashing:(firstHash(key)+i*secondHash(key)) %tableSize→ (2+1*2)) %7 =4

Astheslotwithhashvalue4 is empty, store 23 there. This is shown in the below diagram.

f(23)=23 mod7=2 →

HashValue	DataRecord
0	21*
1	
2	16*
3	10*
4	23*
5	12*
6	

Figure5.4: *Double Probing*

ii. Separate Chaining or Closed addressing:

To handle the collision, This technique creates a linked list to the slot for which collision occurs. The new key is then inserted in the linked list. These linked lists to the slots appear like chains. So, this technique is called as *separate chaining*. It is also called as closed addressing.

Example: Inserting10, 21,16, 12, 23,19, 28, 30in hash table.

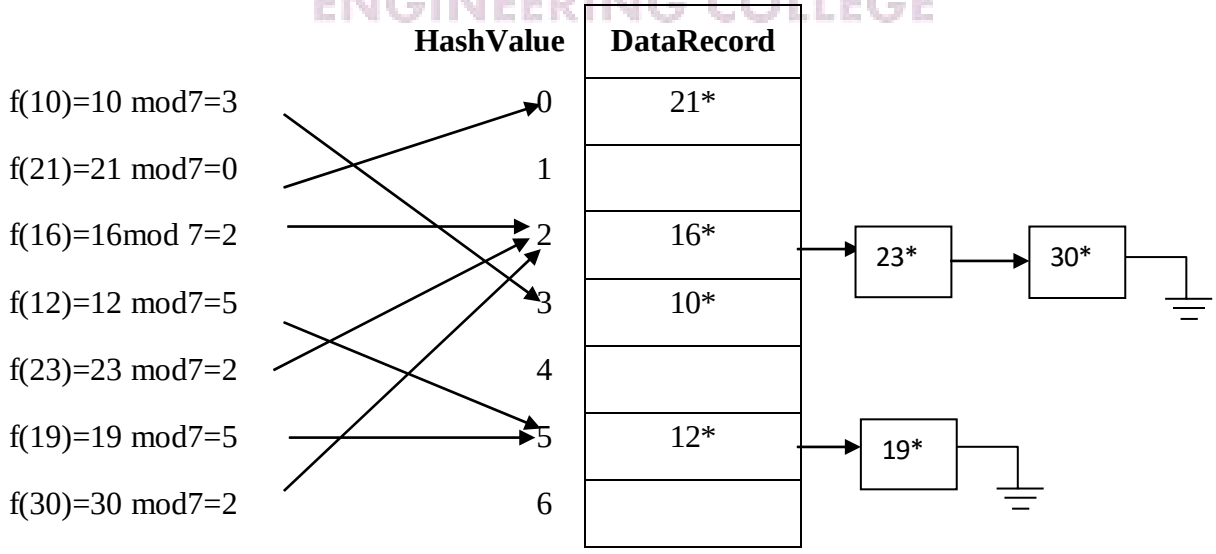


Figure5.5: *Separate chaining example*

4. DYNAMIC HASHING

The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing can be implemented using two techniques. They are:

- Extendedhashing
- LinearHashing

i. Extendable hashing

In extendable hashing, a separate *directory* of pointers to buckets is used. The number bitsusedindirectoryiscalledglobaldepth(gd)andnumberentriesindirectory=2^{gd}.Numberof bits used for locating the record in the buckets is called *local depth*(ld) and each bucket canstores up to 2^{ld} entries. The hash function use last few binarybits of the keyto find the bucket. If a bucket over flows, it splits, and if local depth greater than global depth, then the table double sin size. It is one form of dynamic hashing.

Example: Let global depth (gd) = 2. It means the directory contains four entries. Let the local depth (ld) of each bucket = 2. It means each bucket need two bits to perform search operation. Let each Bucket capacity is four. Let us insert 21, 15, 28, 17, 16, 13, 19, 12, 10, 24, 25 and 11.

21 = 10101
15 = 01111
28 = 11100
17 = 10001
16 = 10000
13 = 01101
19 = 10011
12 = 01100
10 = 01010
24 = 11000
25 = 11101
11 = 01011

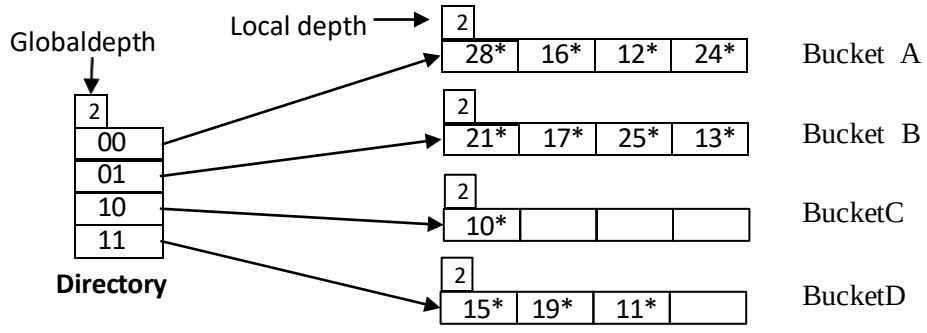


Figure6.1: *Extendible hashing example*

Now, let us consider insertion of data entry 20* (binary 101**00**). Looking at directory element00, we are led to bucketA,whichisalreadyfull.So,wehavetos**plit**thebucketby

allocating a new bucket and redistributing the contents (including the new entry to be inserted) across the old bucket and its 'split image (new bucket). To redistribute entries across the old bucket and its split image, we consider the last *three bits;* the last two bits are 00, indicating a data entrythat belongs to one of these two buckets, and the third bit discriminates between these buckets. That is if a key's last three bits are 000, then it belongs to bucket A and if the last three bits are 100, then the key belongs to Bucket A2.As we are using three bits for A and A2, so the local depth of these buckets becomes 3. This is illustrated in the below Figure 6.2.

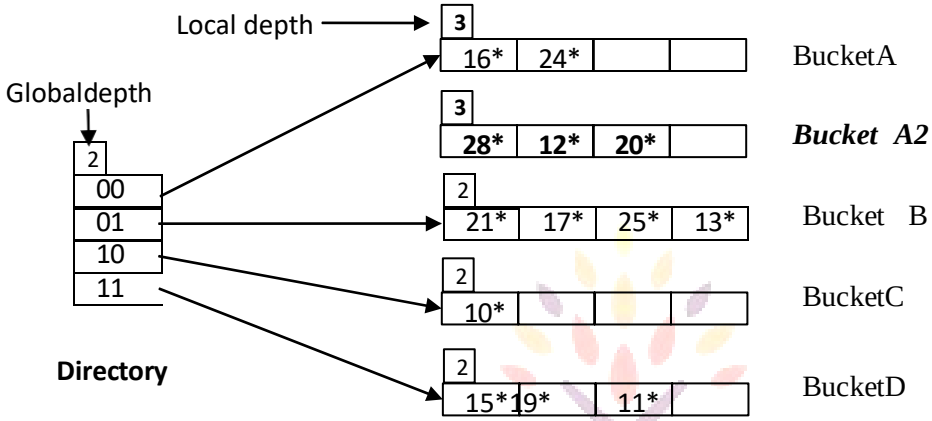


Figure6.2: *After inserting20andsplittingBucket A*

Aftersplit, BucketAandA2have local depth greater than global depth(3>2),so double the directory and use three bits as global depth. As Bucket A and A2 has local depth 3, so they have separate pointers from the directory. But, Buckets B, C and D use only local depth 2, so they have two pointers each. This is shown in the below diagram.

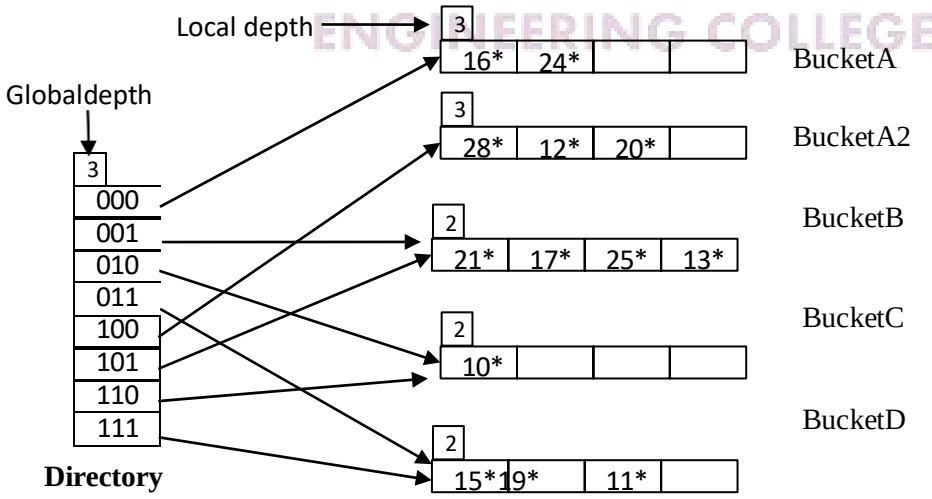


Figure6.3: *After inserting 20and doubling the directory*

An important point that arises is whether splitting a bucket necessitates a directory doubling. Consider our example, as shown in Figure 6.3. If we now insert 9* (01001), it belongs in bucket B;this bucket is already full.Wecan deal with this situation by splitting the bucket and using directory elements 001 and 101to point to the bucket and its split image. This is shown in the below diagram. As Bucket B and its split image now have local depth three and it is not greaterthan global depth. Hence, abucket split does not necessarilyrequire adirectorydoubling. However, if either bucket A or A2 grows full and an insert then forces a bucket split, we are forced to double the directory once again.

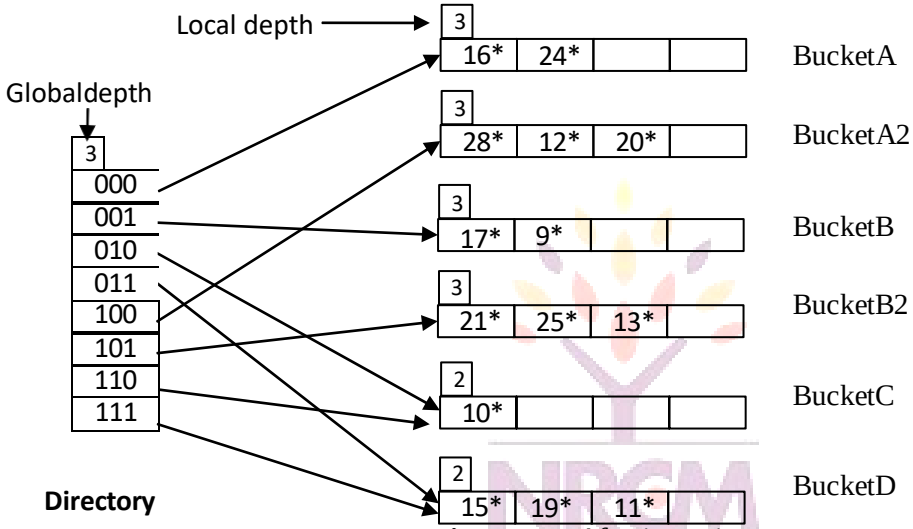


Figure6.4: *Afterinserting 9*

KeyObservations:

- A Bucket will have more than one pointers pointing to it if its local depth is less than the global depth.
- When over flow condition occurs in a bucket, all the entries in the bucket are hashed with a new local depth.
- If new Local Depth of the over flowing bucket is equal to the global depth, only then the directories are doubled and the global depth is incremented by 1.
- Thesizeof abucketcannot bechangedafterthedata insertionprocess begins.

ii. LinearHashing

Linear hashing is a dynamic hashing technique that linearly grows or shrinks number of buckets in a hash file without a directory as used in *Extendible Hashing*. It uses a family of hash functions instead of single hash function.

This scheme utilizes a *family* of hash functions h_0, h_1, h_2, ... , with the property that each function's range is twice that of its predecessor. That is, if h_i maps a data entry into one of N buckets, h_{i+1} maps a data entry into one of $2N$ buckets. One example of such hash functionfamily can be obtained by: $h_{i+1}(key) = key \bmod (2^i N)$ where N is the initial number of buckets and i = 0,1,2,….

Initiallyit use N buckets labelled 0 through N–1 and an initial hashingfunction $h_0(key)$ = key% N is used to map anykeyinto one of the N buckets. For each overflow bucket, one of the buckets in serial order will be splited and its content is redistributed between it and its split image. That is, for first time overflow in any bucket, bucket 0 will be splited, for second time overflow in any bucket; bucket 1 will be splited and so on. When number of buckets becomes 2N, then this marks the end of splitting round 0. Hashing function h_0 is no longer needed as all 2N buckets can be addressed by hashing function h_1. In new round namely splitting-round 1, bucketsplitonceagainstartsfrombucket0.Anewhashfunction$h_2$willbeused.Thisprocessis repeated when the hash file grows.

Example: Let N = 4, so we use 4 buckets and hash function $h_0(key)$ = key % 4 is used to map any key into one of the four buckets. Let us initially insert 4, 13, 19, 25, 14, 24, 15, 18, 23, 11, 16, 12 and 10. This is shown in the below figure.

Bucket#	h_1h_0		*Primarypages*				*Overflowpages*
0	000	00	4*	24*	16*	12*	
1	001	01	13*	25*			
2	010	10	14*	18*	10*		
3	011	11	19*	15*	23*	11*	

Next, when 27is inserted, an over flow occurs in bucket3.So, bucket 0 (firstbucket) is splited and its content is distributed between bucket 0and new bucket. This is shown in below figure.

Bucket#	h_1h_0		*Primarypages*				*Overflowpages*			
0	000	00	24*	16*						
1	001	01	13*	25*						
2	010	10	14*	18*	10*					
3	011	11	19*	15*	23*	11*	27*			
4	100	00	4*	12*						

Next, when 30,31and 34 is inserted, an over flow occurs in bucket 2. So, bucket 1issplited and its content is distributed between bucket 1 and new bucket. This is shown in below figure.

Bucket#	*h1*	*h0*	*Primarypages*	*Overflowpages*
0	000	00	24* 16*	
1	001	01	13*	
2	010	10	14* 18* 10* 30*	34*
3	011	11	19* 15* 23* 11*	27* 31*
4	100	00	4* 12*	
5	101	01	25*	

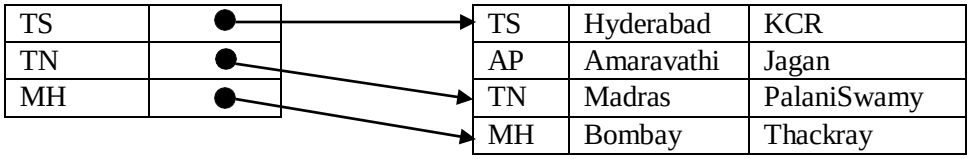
When32,35,40 and 48 is inserted, an over flow occurs in bucket0.So, bucket 2is splited and its content is distributed between bucket 2 and new bucket. This is shown in below figure.

Bucket#	h_1	h_0	*Primarypages*	*Overflowpages*
0	000	00	24* 16* 32* 40*	48*
1	001	01	13*	
2	010	10	18* 10* 34*	
3	011	11	19* 15* 23* 11*	27* 31* 35*
4	100	00	4* 12*	
5	101	01	25*	
6	110	10	14* 30*	

When26,20 and 42is inserted, an over flow occurs in bucket 0.So, bucket3issplited andits content is distributed between bucket 3 and new bucket.This is shown in below figure.

Bucket#	*h1*	*h0*	*Primarypages*	*Overflowpages*
0	000	00	24* 16* 32* 40*	48*
1	001	01	13*	
2	010	10	18* 10* 34* 26*	42
3	011	11	19* 11* 27* 35*	
4	100	00	4* 12* 20*	
5	101	01	25*	
6	110	10	14* 30*	
7	111	11	15* 23* 31*	

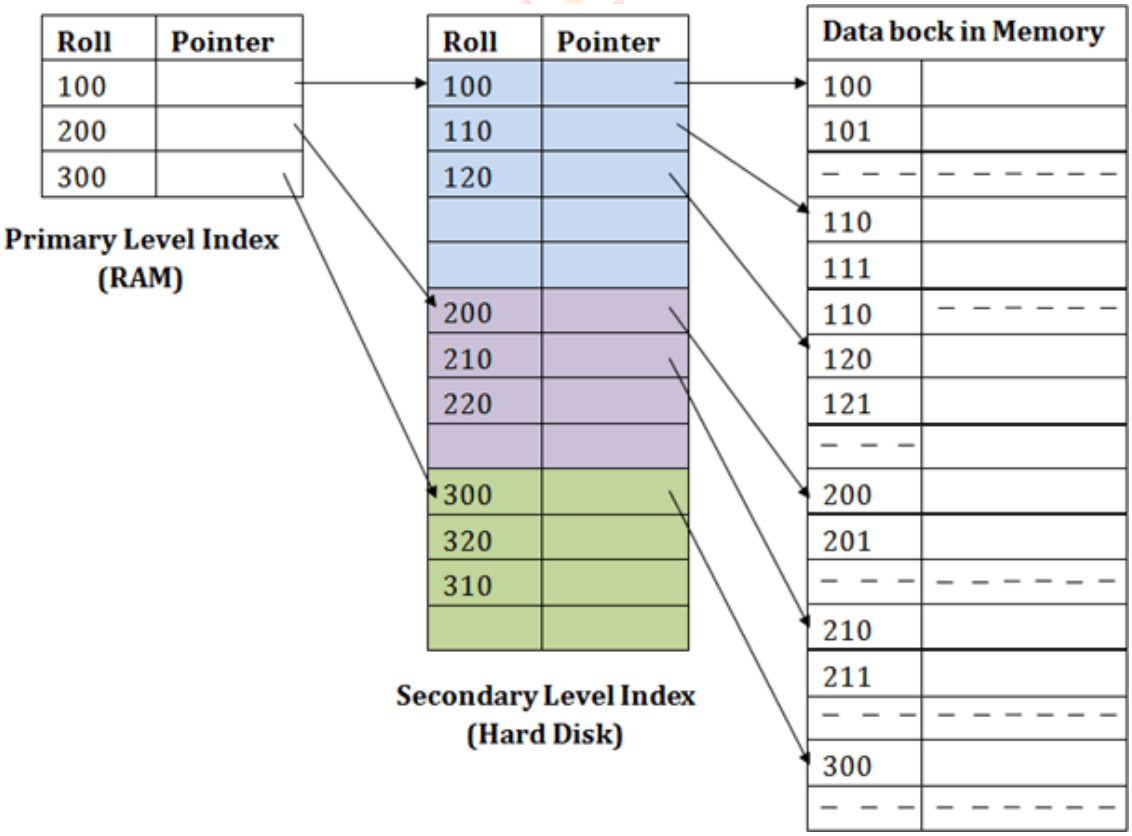
This marks the end of splitting round. Hashing function $h_0$is no longer needed as all 2N buckets can be addressed by hashing function h_1. In new round namely splitting-round 1, bucket split once again starts from bucket 0. A new hash function $h_2$will be used. This process is repeated.



i. Secondary Index

When the sizeofthe main table grows, then sizeofindex tablealso grows. If theindex tablesize grows then fetching the address itself becomes slower. To overcome this problem, secondary indexing is introduced.

- Insecondaryindexing, toreducethesizeofmapping, another levelof indexingis introduced.
- Itcontainstwolevels. Inthefirstleveleachrecordinthemaintablehasoneentryinthefirst- level index table.
- The index entries in the fisrt level index table are divided into different groups. For each group, one index entry is created and added in the second level index table.

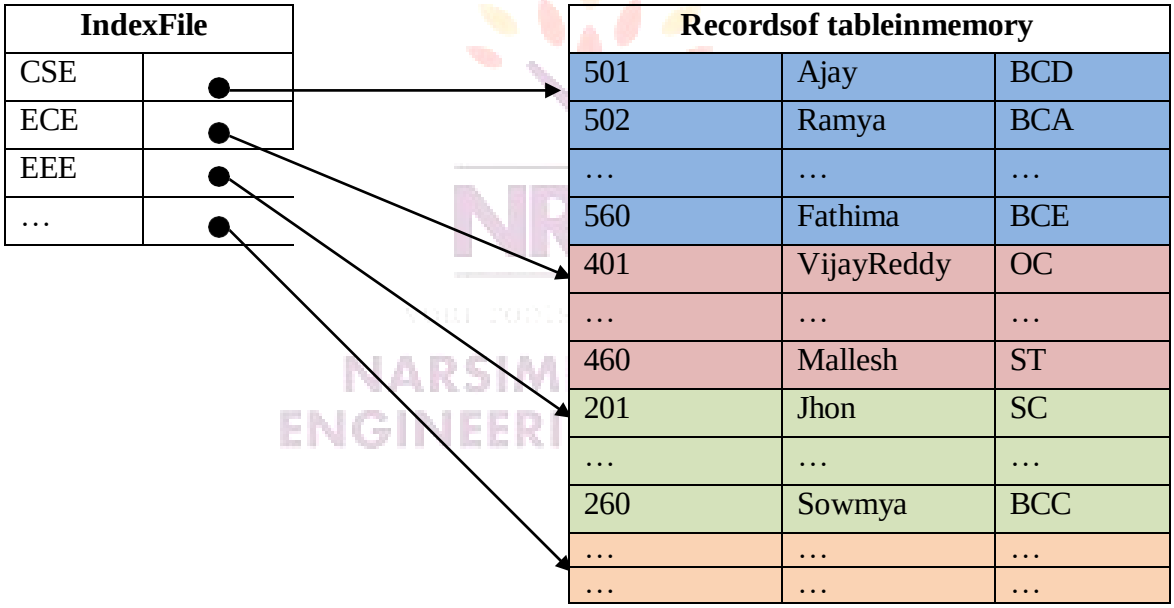


Multi-level Index: When the maintable size becomes too large, creating secondarylevelindex improves the search process.Even if the search process is slow; we can add one more level of indexing and so on. This type of indexing is called multi-level index.

ii. Clustering Index

- Some times the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Example: Consider a college contains many students in each department. All the students belong to the same Dept_ID are grouped together and treated as a single cluster. One index pointers point to the one cluster as a whole. The idex pointer points to the first record in each cluster. Here Dept_ID is a non-unique key.



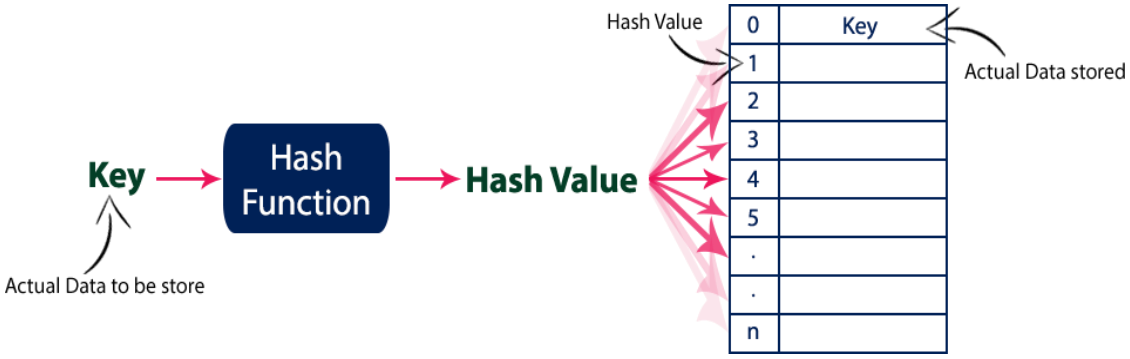
In above diagram we can see that, indexes are created for each department in the index file. In the data block, the students of each department are grouped together to form the cluster. The address in the index file points to the beginning of each cluster.

5. HASHBASED INDEXING

Hashing is a technique to directly search the location of desired data on the disk without using index structure. Hash function is a function which takes a piece of data (key) as input and

produces a hash valueas output which maps the data to a particular location in the hash table.

Theconcept of hashingandhash table is shownin thebelow figure



There aremainlytwo typesof hashing methods:

i. Static Hashing

ii. DynamicHashing

- Extendedhashing
- Linearhashing

6. STATIC HASHING

In static hashing, the hash function produces only fixed number of hash values. For example consider the hash function

f(x)= x mod 7

Foranyvalueof***x***, theabovefunctionproducesoneof the hash value from {0,1,2, 3,4, 5,6}. It means static hashing maps search-key values to a fixed set of bucket addresses.

Example: Inserting10,21,16and12inhash table.

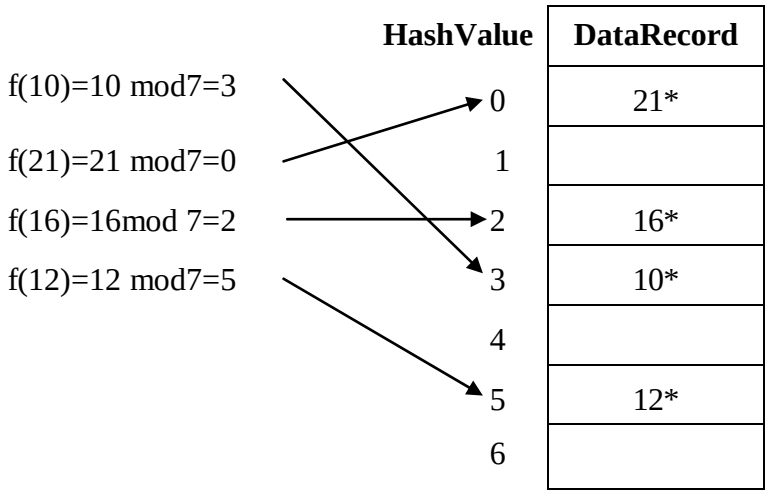


Figure5.1: *Statichashing*

Suppose, latter if we want to insert 23, it produce hash value as 2 (23 mod 7 = 2). But, in the above hash table, the location with hash value 2 is not empty (it contains 16*). So, a collision occurs. To resolve this collision, the following techniques are used.

- Open addressing
- SeparateChainingorClosed addressing

iii. Open Addressing:

Open addressing is a collision resolving technique which stores all the keys inside the hash table. No key is stored outside the hash table. Techniques used for open addressing are:

- Linear Probing
- Quadratic Probing
- Double Hashing

➢ LinearProbing:

In linear probing, when there is a collision, we scan forwards for the next empty slot to fill the key's record. If you reach last slot, then start from beginning.

Example: Consider figure 1. When we try to insert 23, its hash value is 2. But the slotwith 2 is not empty. Then move to next slot (hash value 3), even it is also full, then move once again to next slot with hash value 4. As it is empty store 23 there. This is shown in the below diagram.

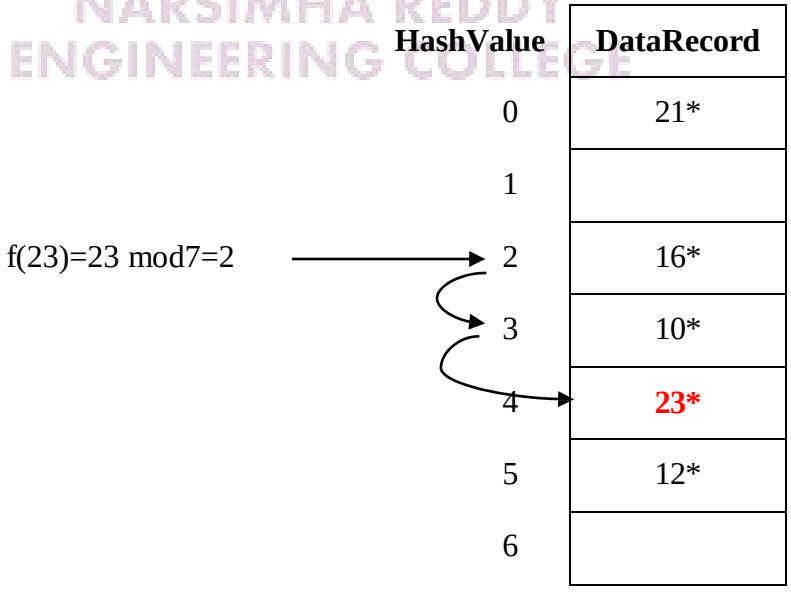


Figure5.2: *Linear Probing*

- **QuadraticProbing:**

In quadratic probing, when collision occurs, it compute new hash value by taking the original hash value and adding successive values of quadratic polynomial until an open slot is found. If here is a collision, it use the following hash function: $h(x) = (f(x) + i^2) \bmod n$,where I = 1, 2, 3, 4,….. and f(x) is initial hash value.

Example: Consider figure 1. When we try to insert 23, its hash value is 2. But the slot with hash value 2 is not empty. Then compute new hash value as $(2 + 1^2) \bmod 7 = 3$, even it is also full, and then once again compute new hash value as $(2 + 2^2) \bmod 7 = 6$. As it is empty store 23 there. This is shown in the below diagram.

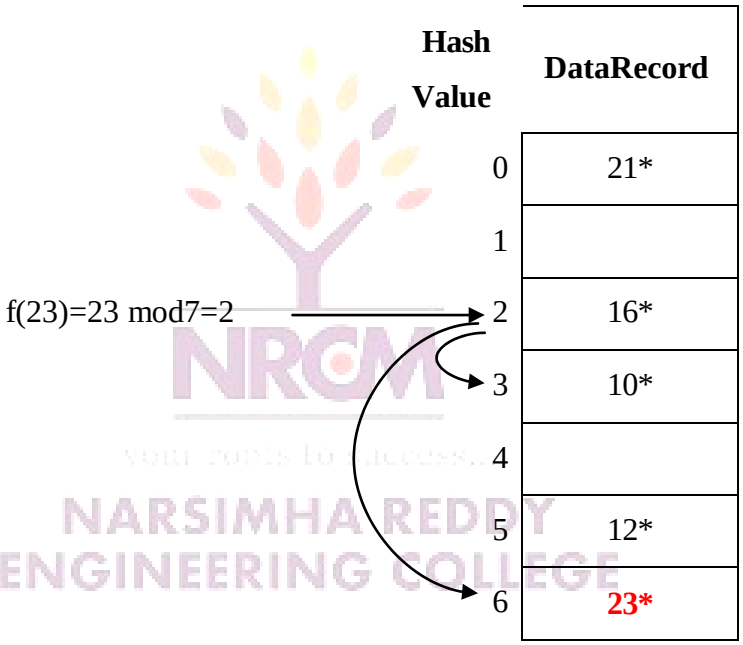


Figure5.3:*Quadratic Probing*

- **DoubleHashing**

In double hashing, there are two hash functions. The second hash function is used to provide an offset value in case the first function causes a collision. The followingfunction is an example of double hashing: (firstHash(key) + i * secondHash(key)) % tableSize. Use i = 1, 2, 3, …

Apopular secondhash functionis : secondHash**(key)=PRIME–(key%PRIME)**

wherePRIMEisaprimesmallerthantheTABLE_SIZE.

Example: Consider figure1.When we try toinsert 23, its hash valueis2.Buttheslot with hash value 2 is not empty. Then compute double hashing value as

secondHash (key)=PRIME–(key%PRIME) → second Hash (23)=5–(23 %5)=2

Doublehashing:(firstHash(key)+i*secondHash(key)) %tableSize→ (2+1*2)) %7 =4

Astheslotwithhashvalue4 is empty, store 23 there. This is shown in the below diagram.

f(23)=23 mod7=2 →

HashValue	DataRecord
0	21*
1	
2	16*
3	10*
4	23*
5	12*
6	

Figure5.4:*Double Probing*

iv. Separate Chaining or Closed addressing:

To handle the collision, This technique creates a linked list to the slot for which collision occurs. The new key is then inserted in the linked list. These linked lists to the slots appear like chains. So, this technique is called as *separate chaining*. It is also called as closed addressing.

Example: Inserting10, 21,16, 12, 23,19, 28, 30in hash table.

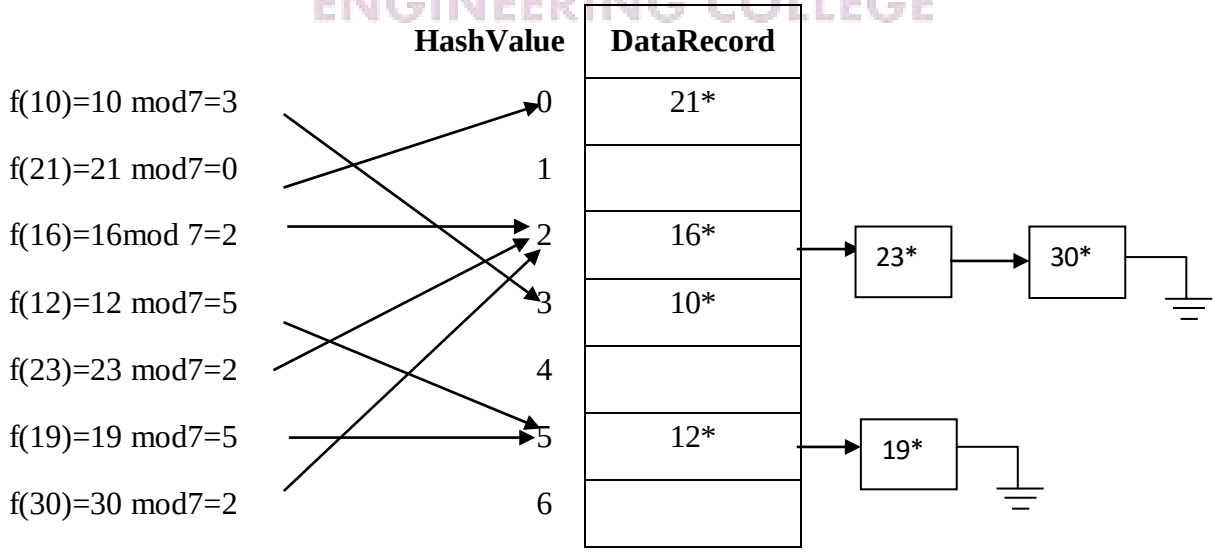


Figure5.5: *Separate chaining example*

7. DYNAMIC HASHING

The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing can be implemented using two techniques. They are:

- Extendedhashing
- LinearHashing

iii. Extendable hashing

In extendable hashing, a separate *directory* of pointers to buckets is used. The number bitsusedindirectoryiscalledglobaldepth(gd)andnumberentriesindirectory=2^{gd}.Numberof bits used for locating the record in the buckets is called *local depth*(ld) and each bucket canstores up to 2^{ld} entries. The hash function use lasts few binarybits of the keyto find the bucket. If a bucket over flows, it splits, and if local depth greater than global depth, then the table doubles in size. It is one form of dynamic hashing.

Example: Let global depth (gd) = 2. It means the directory contains four entries. Let the local depth (ld) of each bucket = 2. It means each bucket need two bits to perform search operation. Let each Bucket capacity is four. Let us insert 21, 15, 28, 17, 16, 13, 19, 12, 10, 24, 25 and 11.

21	=	10101	19	=	10011
15	=	01111	12	=	01100
28	=	11100	10	=	01010
17	=	10001	24	=	11000
16	=	10000	25	=	11101
13	=	01101	11	=	01011

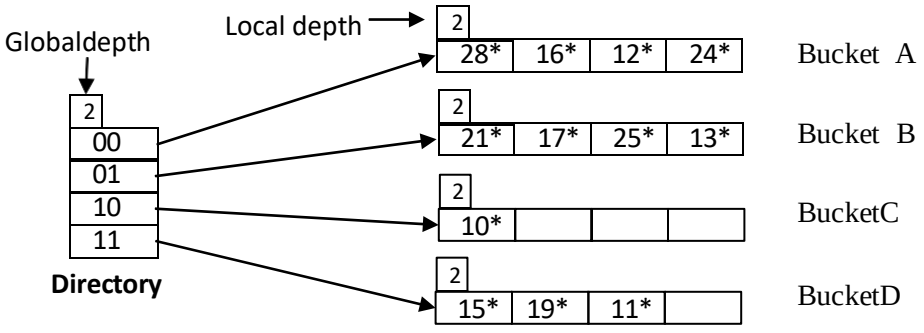


Figure6.1: *Extendible hashing example*

Now, let us consider insertion of data entry 20* (binary 101**00**). Looking at directory element00, we are led to bucket A, which is already full. So,we have to **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 (new bucket). To redistribute entries across the old bucket and its split image, we consider the last *three bits;* the last two bits are 00, indicating a data entrythat belongs to one of these two buckets, and the third bit discriminates between these buckets. That is if a key's last three bits are 000, then it belongs to bucket A and if the last three bits are 100, then the key belongs to Bucket A2.As we are using three bits for A and A2, so the local depth of these buckets becomes 3. This is illustrated in the below Figure 6.2.

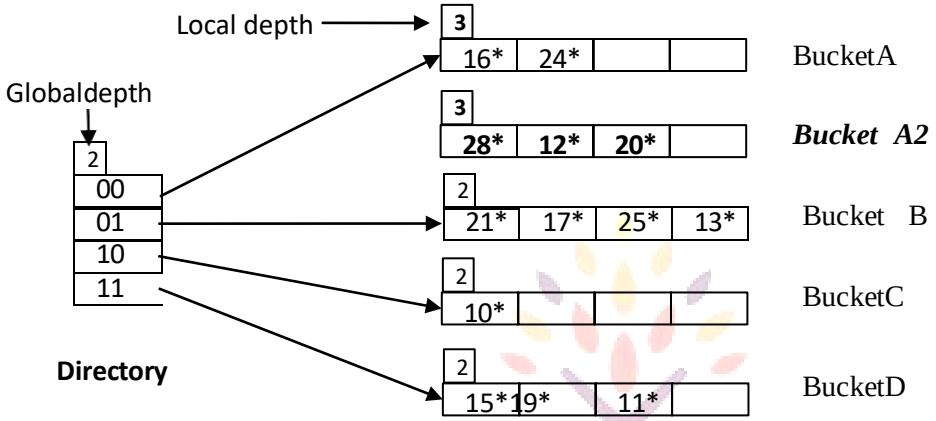


Figure6.2: *After inserting 20andsplittingBucket A*

Aftersplit, Bucket Aand A2 have local depth greater than global depth(3>2),so double the directory and use three bits as global depth. As Bucket A and A2 has local depth 3, so they have separate pointers from the directory. But, Buckets B, C and D use only local depth 2, so they have two pointers each. This is shown in the below diagram.

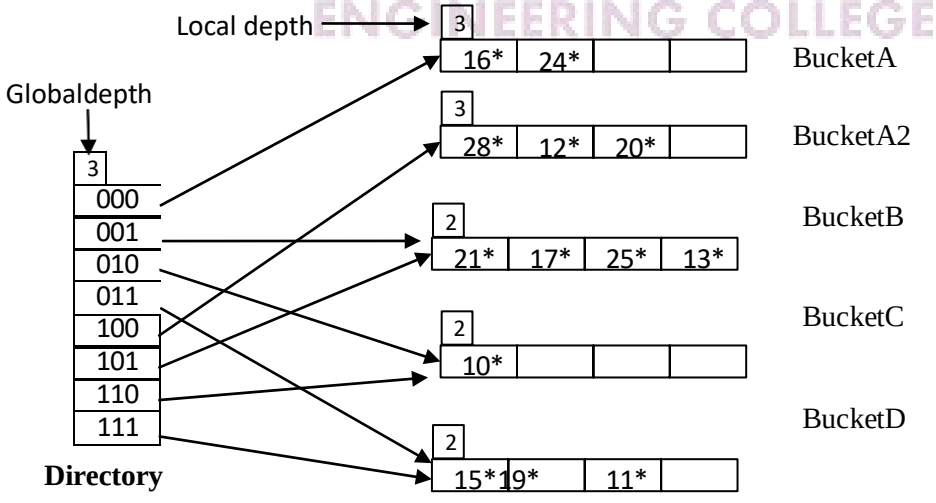


Figure6.3: *After inserting 20and doubling the directory*

An important point that arises is whether splitting a bucket necessitates a directory doubling. Consider our example, as shown in Figure 6.3. If we now insert 9* (01001), it belongs in bucket B;this bucket is already full.Wecan dealwiththissituationbysplittingthebucket and using directory elements 001 and 101to point to the bucket and its split image. This is shown in the below diagram. As Bucket B and its split image now have local depth three and it is not greaterthan global depth. Hence, abucket split does not necessarilyrequire adirectorydoubling. However, if either bucket A or A2 grows full and an insert then forces a bucket split, we are forced to double the directory once again.

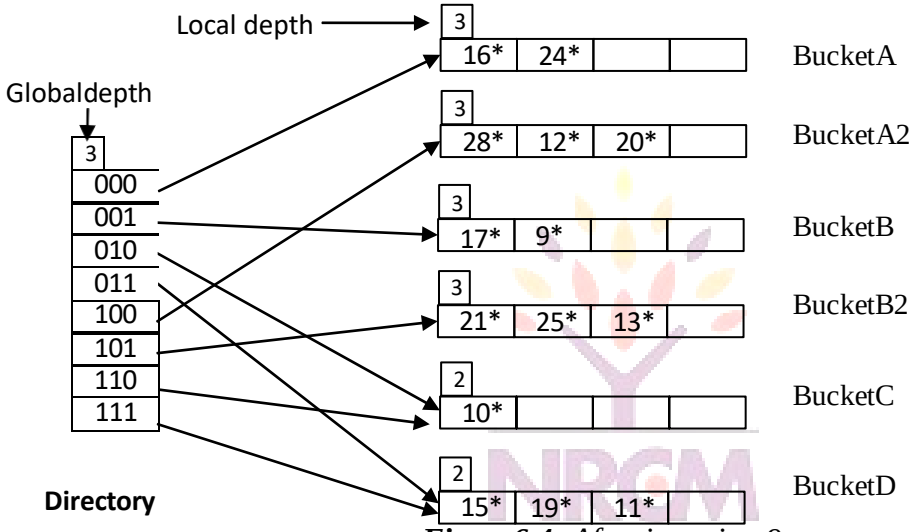


Figure6.4: *After inserting 9*

KeyObservations:

- A Bucket will have more than one pointers pointing to it if its local depth is less than the global depth.
- When over flow condition occurs in a bucket, all the entries in the bucket arere hashed with a new local depth.
- If new Loca lDepth of the over flowing bucket is equal to the global depth, only then the directories are doubled and the global depth is incremented by 1.
- The size of a bucket cannot be changed after the data insertion process begins.

iv. LinearHashing

Linear hashing is a dynamic hashing technique that linearly grows or shrinks number of buckets in a hash file without a directory as used in *Extendible Hashing*. It uses a family of hash functions instead of single hash function.

This scheme utilizes a *family* of hash functions h_0, h_1,h_2, ... , with the property that each function's range is twice that of its predecessor. That is, if $h_i$maps a data entry into one of *N* buckets, h_{i+1} maps a data entry into one of *2N* buckets. One example of such hash functionfamily can be obtained by: $h_{i+1}(key)=keymod\ (2^iN)$ where N is the initial number of buckets and i = 0,1,2,….

Initiallyit use N buckets labelled 0 through N–1 and an initial hashingfunction $h_0(key)$ = key% N is used to map anykeyinto one of the N buckets. For each overflow bucket, one of the buckets in serial order will be splited and its content is redistributed between it and its split image. That is, for first time overflow in any bucket, bucket 0 will be splited, for second time overflow in any bucket; bucket 1 will be splited and so on. When number of buckets becomes 2N, then this marks the end of splitting round 0.Hashing function h_0 is no longer needed as all 2N buckets can be addressed by hashing function h_1. In new round namely splitting-round 1, bucketsplitonceagainstartsfrombucket0.Anewhashfunction$h_2$willbeused.Thisprocessis repeated when the hash file grows.

Example: Let N = 4, so we use 4 buckets and hash function $h_0(key)$ = key % 4 is used to map any key into one of the four buckets. Let us initially insert 4, 13, 19, 25, 14, 24, 15, 18, 23, 11, 16, 12 and 10. This is shown in the below figure.

Bucket#	h_1h_0		*Primarypages*				*Overflowpages*
0	000	00	4*	24*	16*	12*	
1	001	01	13*	25*			
2	010	10	14*	18*	10*		
3	011	11	19*	15*	23*	11*	

Next, when27is inserted, an over flow occurs in bucket3. So, bucket0 (firstbucket)issplited and its content is distributed between bucket 0and new bucket. This isshown inbelow figure.

Bucket#	h_1h_0		*Primarypages*				*Overflowpages*			
0	000	00	24*	16*						
1	001	01	13*	25*						
2	010	10	14*	18*	10*					
3	011	11	19*	15*	23*	11*	27*			
4	100	00	4*	12*						

Next, when30,31and34is inserted, an over flow occurs in bucket 2.So, bucket1is splited and its content is distributed between bucket 1 and new bucket. This is shown in below figure.

Bucket#	*h1*	*h0*	*Primarypages*	*Overflowpages*
0	000	00	24* 16*	
1	001	01	13*	
2	010	10	14* 18* 10* 30*	34*
3	011	11	19* 15* 23* 11*	27* 31*
4	100	00	4* 12*	
5	101	01	25*	

When32,35,40 and 48 is inserted, an over flow occurs in bucket0. So, bucket 2is splited and its content is distributed between bucket 2 and new bucket. This is shown in below figure.

Bucket#	h_1	h_0	*Primarypages*	*Overflowpages*
0	000	00	24* 16* 32* 40*	48*
1	001	01	13*	
2	010	10	18* 10* 34*	
3	011	11	19* 15* 23* 11*	27* 31* 35*
4	100	00	4* 12*	
5	101	01	25*	
6	110	10	14* 30*	

When26,20 and 42is inserted, an over flow occurs in bucket0.So, bucket 3 is splited and its content is distributed between bucket 3 and new bucket.This is shown in below figure.

Bucket#	*h1*	*h0*	*Primarypages*	*Overflowpages*
0	000	00	24* 16* 32* 40*	48*
1	001	01	13*	
2	010	10	18* 10* 34* 26*	42
3	011	11	19* 11* 27* 35*	
4	100	00	4* 12* 20*	
5	101	01	25*	
6	110	10	14* 30*	
7	111	11	15* 23* 31*	

This marks the end of splitting round. Hashing function $h_0$is no longer needed as all 2N buckets can be addressed by hashing function h_1. In new round namely splitting-round 1, bucket split once again starts from bucket 0. A new hash function $h_2$will be used. This process is repeated.