Section B: File Organization and Relational Model and Calculus

Sequential Files:

Sequential file techniques provide a straightforward way to read and write files. PowerBASIC's sequential file commands manipulate *text* files: files of ANSI or WIDE characters with carriage-return/linefeed pairs separating records.

Quite possibly, the best reason for using sequential files is their degree of portability to other programs, programming languages, and computers. Because of this, you can often look at sequential files as the common denominator of data processing, since they can be read by word-processing programs and editors (such as PowerBASIC's), absorbed by other applications (such as database managers), and sent over the Internet to other computers.

The idea behind sequential files is simplicity itself: write to them as though they were the screen and read from them as though they were the keyboard.

Create a sequential file using the following steps:

1. Open the file in sequential output mode. To create a file in PowerBASIC, you must use the OPEN statement. Sequential files have two options to prepare a file for output:

OUTPUT: If a file does not exist, a new file is created. If a file already exists, its contents are erased, and the file is then treated as a new file.

APPEND: If a file does not exist, a new file is created. If a file already exists, PowerBASIC appends (adds) data at the end of that file.

2. Output data to a file. Use WRITE# or PRINT# to write data to a sequential file.
3. Close the file. The CLOSE statement closes a file after the program has completed all I/O operations.

To read a sequential file:

1. First, OPEN the file in sequential INPUT mode. This prepares the file for reading.
2. Read data in from the file. Use PowerBASIC's INPUT# or LINE INPUT# statements.
3. Close the file. The CLOSE statement closes a file after the program has completed all I/O operations.

The drawback to sequential files is, not surprisingly, that you only have sequential access to your data. You access one line at a time, starting with the first line. This means if you want to get to the last line in a sequential file of 23,000 lines, you will have to read the preceding 22,999 lines.

Sequential files, therefore, are best suited to applications that perform sequential processing (for example, counting words, checking spelling, printing mailing labels in file order) or in which all the data can be held in memory simultaneously. This allows you to read the entire file in one fell swoop at the start of a program and to write it all back at the end. In between, the information can be stored in an array (in memory) which can be accessed randomly.

Index sequential files:

ISAM (Indexed Sequential Access Method) is a file management system developed at IBM that allows records to be accessed either sequentially (in the order they were entered) or randomly (with an index). Each index defines a different ordering of the records.

ISAM (Indexed Sequential Access Method) is a file management system developed at IBM that allows records to be accessed either sequentially (in the order they were entered) or randomly (with an index). Each index defines a different ordering of the records. An employee database may have several indexes, based on the information being sought. For example, a name index may order employees alphabetically by last name, while a department index may order employees by their department. A key is specified in each index. For an alphabetical index of employee names, the last name field would be the key.

Direct Access File System (DAFS):

Direct Access **File** System (DAFS) is a network **file** system that is based on NFSv4 and the Virtual Interface (VI) data transfer mechanism. DAFS uses remote **direct** memory access (RDMA) to perform efficient network access to data in remote **files**.

Hashing:

For a huge database structure, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data. Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure.

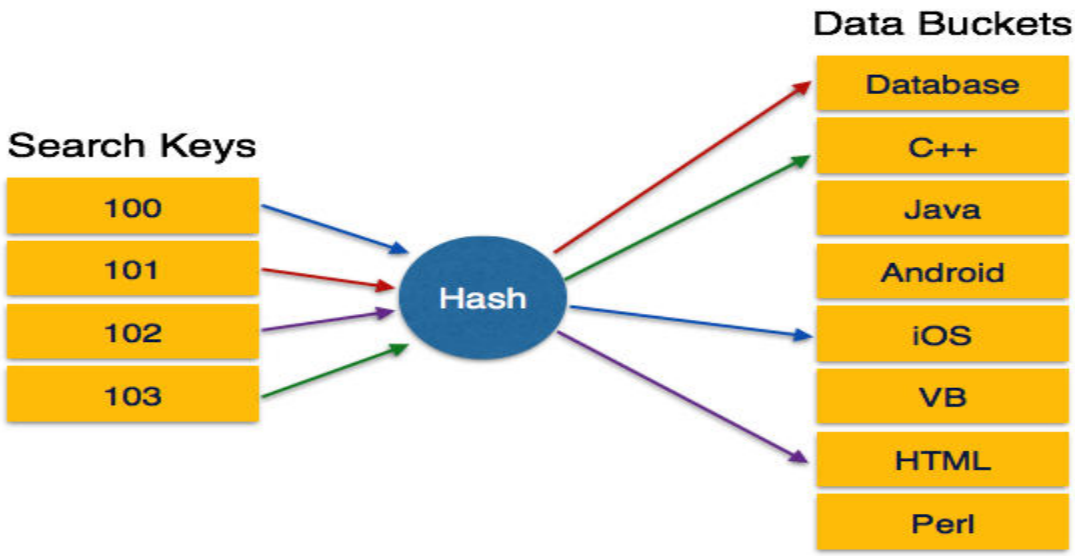
Hashing uses hash functions with search keys as parameters to generate the address of a data record.

Hash Organization

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

Static Hashing

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



Operation

- **Insertion** − When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.

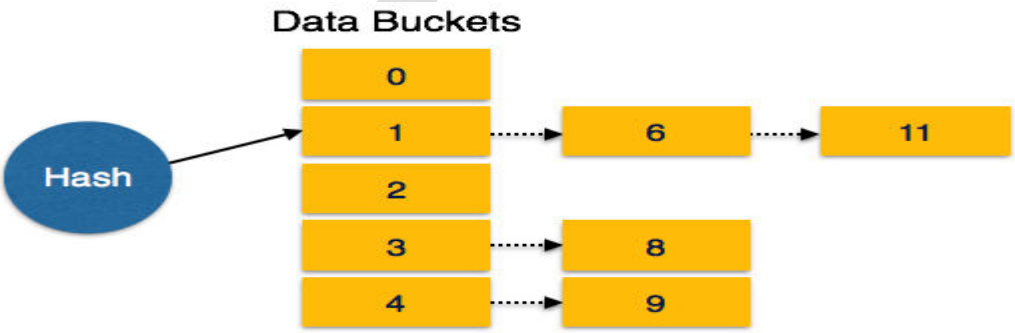
  Bucket address = h(K)

- **Search** − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.
- **Delete** − This is simply a search followed by a deletion operation.

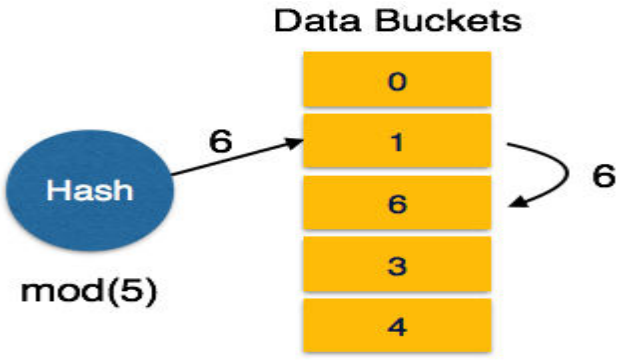
Bucket Overflow

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

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



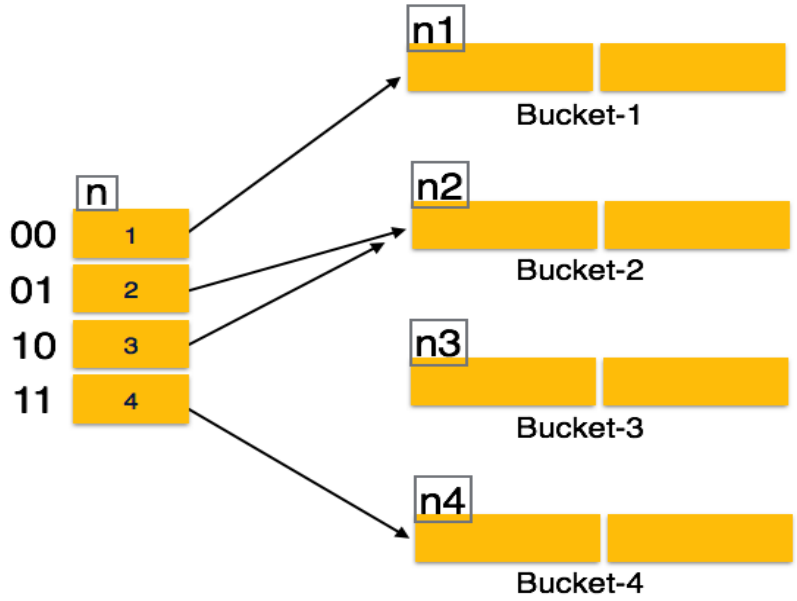
- **Linear Probing** − When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.



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 is also known as **extended hashing**.

Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.



Organization

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

Operation

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

Hashing is not favorable when the data is organized in some ordering and the queries require a range of data. When data is discrete and random, hash performs the best.

Hashing algorithms have high complexity than indexing. All hash operations are done in constant time.

B-Tree Index Files

1. B-tree indices are similar to B +-tree indices.
    - Difference is that B-tree eliminates the redundant storage of search key values.
    - In B +-tree of Figure 11.11, some search key values appear twice.
    - A corresponding B-tree of Figure 11.18 allows search key values to appear only once.
    - Thus we can store the index in less space.

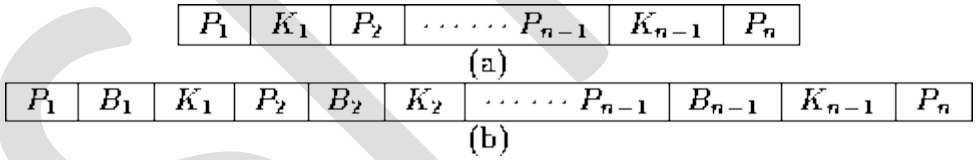


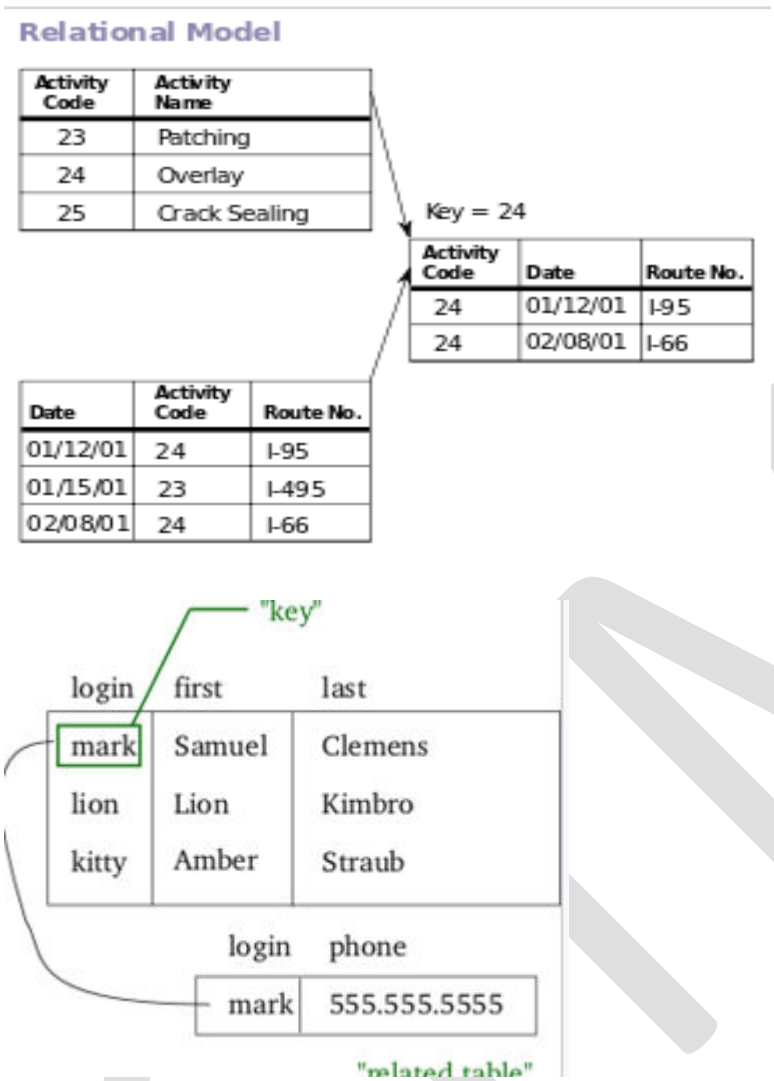
Figure 11.8: Leaf and nonleaf node of a B-tree.

2. **Advantages:**
    - Lack of redundant storage (but only marginally different).
    - Some searches are faster (key may be in non-leaf node).
3. **Disadvantages:**
    - Leaf and non-leaf nodes are of different size (complicates storage)
    - Deletion may occur in a non-leaf node (more complicated)

Generally, the structural simplicity of B +-tree is preferred.

Relational Model:

The **relational model** (RM) for database management is an approach to managing data using a structure and language consistent with first-order predicate logic, first described in 1969 by Edgar F. Codd, where all data is represented in terms of tuples, grouped into relations.



The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

Most relational databases use the SQL data definition and query language; these systems implement what can be regarded as an engineering approximation to the relational model. A *table* in an SQL database schema corresponds to a predicate variable; the contents of a table to a relation; key constraints, other constraints, and SQL queries correspond to predicates. However, SQL databases deviate from the relational model in many details, and Codd fiercely argued against deviations that compromise the original principles.

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}(Books)$

Output − Selects tuples from books where subject is 'database'.

$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An}(r)$

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

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

Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as −

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

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.
- Duplicate tuples are automatically eliminated.

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

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

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r − s**

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

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

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

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

$r X s = \{ q t \mid q \in r \text{ and } t \in s\}$

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

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_x (E)

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

Additional operations are −

- Set intersection
- Assignment
- Natural join

Relational and Tuple Calculus:

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms −

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples

Notation − {T | Condition}

Returns all tuples T that satisfies a condition.

For example −

{ T.name | Author(T) AND T.article = 'database' }

Output − Returns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).

For example −

{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output − The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation −

$\{ a_1, a_2, a_3, ..., a_n \mid P(a_1, a_2, a_3, ... ,a_n)\}$

Where a1, a2 are attributes and **P** stands for formulae built by inner attributes.

For example −

{< article, page, subject > | ∈ TutorialsPoint ∧ subject = 'database'}

Output − Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.

Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.

The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.