

# SPPU-BE-COMP-CONTENT – KSKA Git

## IR UNIT 2 – PYQ Answers

➤ OCT 2022

Q3)

a) Enlist different types of Index Construction and explain any two with suitable examples. [7]

### Types of Index Construction:

1. Inverted Index Construction
2. Forward Index Construction
3. Positional Index Construction
4. Bi-word Index Construction
5. Signature Files
6. Suffix Tree / Suffix Array

### 1. Inverted Index Construction

- **Definition:** Maintains a mapping from terms to a list of documents in which the term occurs.
- **Working:**
  - Scan documents, extract terms, and normalize them.
  - For each term, create a *posting list* containing document IDs (and possibly term positions).
- **Example:**

Term	→	Posting List
"data"	→	[D1, D3, D5]
"mining"	→	[D1, D2]

**Advantage:** Fast retrieval of documents for a given query term.

### 2. Positional Index Construction

## SPPU-BE-COMP-CONTENT – KSKA Git

- **Definition:** Extends inverted index by also storing the positions of each term within a document.
- **Working:**
  - Helps in **phrase queries** and **proximity searches**.
  - Posting list contains (Document ID, Position list) pairs.
- **Example:**

```
"mining" → [(D1: 5, 17), (D2: 3)]
```

This means “mining” occurs at positions 5 and 17 in D1, and position 3 in D2.

**Advantage:** Enables accurate matching for phrases like “data mining”.

### b) Dictionary in Static Inverted Indices & Sort-based vs Hash-based Dictionary [8]

In Information Retrieval, a dictionary is a data structure that stores all distinct terms present in the document collection. Each term in the dictionary is associated with its *posting list*, which contains document IDs where the term appears. In a **static inverted index**, the dictionary is fixed after construction and not updated dynamically.

#### 1. Sort-based Dictionary:

- **Concept:** Terms are stored in sorted order (alphabetically or lexicographically).
- **Lookup:** Binary search is used to find the term’s posting list.
- **Advantages:**
  - Efficient in range queries (e.g., finding all terms starting with “ab”).
  - Easy to compress using prefix compression.
- **Disadvantages:**
  - Lookup time is  $O(\log n)$  due to binary search.

```
Terms: ["apple", "banana", "cat"]  
Searching "banana" → Binary Search → Found at index 1.
```

#### 2. Hash-based Dictionary:

## SPPU-BE-COMP-CONTENT – KSKA Git

- **Concept:** Terms are stored in a hash table using a hash function to map terms to positions.
- **Lookup:** Constant time on average ( $O(1)$ ).
- **Advantages:**
  - Faster term lookup compared to sort-based.
- **Disadvantages:**
  - Cannot efficiently support range queries.
  - Collisions require resolution strategies.

```
Hash("apple") → Index 3
Hash("banana") → Index 7
```

Feature	Sort-based Dictionary	Hash-based Dictionary
Lookup Time	$O(\log n)$ (Binary Search)	$O(1)$ Average
Range Queries	Supported	Not Supported
Compression	Easier (Prefix Compression)	Difficult

### Q4)

#### a) List and explain different classes of operators used in Query optimization. [7]

In query optimization, **operators** are the fundamental building blocks used by a query execution engine to process data. They define how tuples (rows) are retrieved, filtered, joined, or aggregated. Operators can be broadly classified into the following **classes**.

##### 1. Relational Algebra Operators

These operators are based on **relational algebra** and perform operations on relations (tables) to produce new relations.

- **Selection ( $\sigma$ )** – Retrieves rows that satisfy a given condition.  
*Example:*  $\sigma$  salary > 50000 (Employee)
- **Projection ( $\pi$ )** – Retrieves specific columns from a table.  
*Example:*  $\pi$  name, salary (Employee)

## SPPU-BE-COMP-CONTENT – KSKA Git

- **Join ( $\bowtie$ )** – Combines tuples from two relations based on a join condition.  
*Example:* Employee  $\bowtie$  Dept

### 2. Logical Operators

These operators represent the **logical plan** of a query, independent of physical implementation.

- Define **what** operation to perform, not **how** to perform it.
- Examples: Logical Selection, Logical Join, Logical Group By.
- Used in the **query rewrite** phase before physical optimization.

### 3. Physical Operators

These represent **actual algorithms** used to implement logical operators.

- **Examples:**
  - **Index Scan** – Fetches rows using an index.
  - **Nested Loop Join** – Joins two tables by iterating through one and matching with the other.
  - **Merge Join** – Joins two sorted tables efficiently.
- The optimizer chooses the best **physical operator** based on cost estimation.

### 4. Set Operators

Used to combine results of multiple queries.

- **Union ( $\cup$ )** – Combines results without duplicates.
- **Intersection ( $\cap$ )** – Retrieves common tuples from two results.
- **Difference ( $-$ )** – Retrieves tuples in one relation but not in another.

### 5. Aggregate Operators

Perform computations on sets of tuples and return a single value or grouped results.

- Examples: **SUM, AVG, MIN, MAX, COUNT.**
- Often combined with **GROUP BY** in SQL queries.
- Optimizers can use **hash aggregation** or **sort-based aggregation** for efficiency.

## SPPU-BE-COMP-CONTENT – KSKA Git

b) Write a short note on: [8]

i) Query Optimization

ii) Lightweight Structure

### i) Query Optimization (8-Mark Answer)

**Definition:**

Query optimization is the process of selecting the **most efficient execution plan** for a database query by evaluating multiple possible strategies while ensuring the same correct result.

**Goal:**

- **Minimize execution time** (reduces latency).
- **Reduce resource consumption** (CPU, memory, I/O operations).
- **Improve scalability** for large datasets.

**Types of Query Optimization:**

#### 1. Logical Optimization

- Involves **rewriting the query** using relational algebra laws without altering its output.
- Techniques:
  - Predicate pushdown (applying filters early).
  - Join reordering (optimizing join sequence).
  - Subquery flattening (converting subqueries to joins).
- Example: Transforming `SELECT * FROM Employee WHERE salary > 5000 AND age < 30` into an optimized predicate evaluation order.

#### 2. Physical Optimization

- Chooses the **best physical execution methods** (access paths, algorithms).
- Techniques:
  - **Index Scan** (faster than full table scan for selective queries).
  - **Hash Join vs. Sort-Merge Join** (picking based on data size/sortedness).
  - **Parallel execution** (splitting work across CPUs/nodes).
- Example: Using a **B+ tree index** for a WHERE clause on a primary key.

**Importance:**

- **Critical for performance** in large-scale databases (OLTP, OLAP).

## SPPU-BE-COMP-CONTENT – KSKA Git

- Reduces **query response time** for end-users.
  - Saves **computational resources** (lower cost in cloud databases).
- 

### ii) Lightweight Structure in Information Retrieval (8-Mark Answer)

#### Definition:

A lightweight structure refers to **efficient indexing and storage techniques** that minimize memory and disk usage while maintaining fast retrieval speeds in search systems.

#### Purpose:

- **Reduce storage overhead** (compressed representations).
- **Speed up search operations** (faster lookups).
- **Maintain acceptable accuracy** (trade-offs with heavy structures).

#### Key Features:

1. **Compressed Data Structures**
  - **Posting lists** (stored using variable-byte encoding, delta encoding).
  - **Dictionary storage** (using tries, minimal perfect hashing instead of B-trees).
2. **Avoiding Redundant Metadata**
  - Storing only essential term mappings (e.g., in an inverted index).
  - Skipping unnecessary positional info for some queries.
3. **Efficient Lookup Methods**
  - **Bloom filters** for membership checks.
  - **Front-coding** in dictionary storage for similar terms.

#### Example:

- Instead of a **B-tree** for term lookup, a **trie** or **hash table** reduces access time.
- **Compressed inverted indexes** in search engines (e.g., Elasticsearch, Lucene).

#### Advantages:

- **Faster query processing** (lower latency).
- **Reduced disk/memory usage** (cost-effective for large datasets).
- **Scalability** (better suited for distributed IR systems).

➤ 2024

Q3)

a) Enlist different types of Index construction and explain any two with suitable examples.[7]

ALREADY DONE

b) Describe the different Query processing techniques like Query Processing for Ranked Retrieval, Document-at-a-Time Query Processing and Term-at-a-Time Query Processing. [8]

## 1) Query Processing for Ranked Retrieval

- **Definition:** In ranked retrieval, documents are scored and sorted based on their relevance to the query using models like TF-IDF or BM25.
  - **Process:**
    1. Compute a **score** for each document based on matching terms.
    2. Sort documents in **descending order** of score.
    3. Return the **top-k** most relevant results.
  - **Example:** Search engines ranking web pages for a given keyword.
  - **Advantage:** Users see the most relevant documents first.
- 

## 2) Document-at-a-Time (DAAT) Query Processing

- **Definition:** Processes one document at a time, examining all query terms for that document before moving to the next.
  - **Working:**
    - Traverse the posting lists of all query terms in parallel.
    - Compute the document score as soon as all term contributions are available.
  - **Example:** If query = “data mining,” fetch postings for “data” and “mining” together, calculate score for doc1, then move to doc2.
  - **Advantage:** Efficient when scoring requires contributions from all query terms.
- 

## 3) Term-at-a-Time (TAAT) Query Processing

- **Definition:** Processes one query term’s posting list at a time, updating partial scores for all documents containing that term.
- **Working:**

## SPPU-BE-COMP-CONTENT – KSKA Git

- Start with the first term, score its documents.
- Move to the next term, **add to existing scores**.
- Continue until all query terms are processed.
- **Example:** For “data mining,” first process “data,” then update scores with “mining.”
- **Advantage:** Can stop early if partial results are sufficient for top-k retrieval.

### Q4)

#### a) Inverted Index Construction, Index Components and Index Life Cycle [7]

##### 1) Inverted Index Construction

- **Definition:** An inverted index maps each term to the list of documents (and positions) in which it appears. It is the backbone of most search engines.
- **Steps:**
  1. **Tokenization** – Break documents into tokens (e.g., “data mining is fun” → “data”, “mining”, “fun”).
  2. **Stopword Removal & Stemming** – Remove common words and reduce words to root form.
  3. **Posting List Creation** – For each unique term, create a list of (DocID, positions).
  4. **Sorting & Merging** – Sort terms alphabetically or by hash and merge postings for efficiency.
- **Example:**

kotlin

Term: **data** → [Doc1, Doc3]  
      **mining** → [Doc1, Doc2]

##### 2) Index Components

- **a) Dictionary:** Stores unique terms and metadata (document frequency, pointer to posting list).
- **b) Posting Lists:** For each term, list of document IDs (optionally with term frequency, positions).
- **c) Document Store:** Stores raw documents or metadata for retrieval.
- **d) Skip Pointers (optional):** Added in postings for faster search jumps.



## SPPU-BE-COMP-CONTENT – KSKA Git

- **Example:**

```
makefile
```

```
Dictionary: data → ptr1, mining → ptr2  
Postings: ptr1 → [1, 3], ptr2 → [1, 2]
```

### 3) Index Life Cycle

- **a) Creation:** Build the inverted index from a document collection.
- **b) Updating:** Add, delete, or modify documents (update postings and dictionary).
- **c) Optimization:** Merge small indexes, remove deleted doc entries, compress postings.
- **d) Maintenance:** Keep the index synchronized with document changes.
- **e) Deletion:** Remove outdated index files when no longer needed.
- **Example:** Search engines periodically rebuild or update indexes as new web pages are crawled.

**b) Write a short note on: [8]**

**i) Query Optimization**

**ii) Lightweight Structure**

**ALREADY DONE !**

**“Check / Verify Answer – Read at Your Own Risk”**