ANSI/SPARC Architecture Overview

External Level:

- Provides tailored views of the database for individual users.
- Hides parts of the data and presents it in a useful form.

Conceptual Level:

- Manages the organization of data across the entire database.
- Uses abstractions to simplify the internal level's complexities.

Internal Level:

- Focuses on the physical storage of data.
- Details how records are stored on disk using files, pages, and blocks.

Logical vs. Physical Data Independence

Logical Data Independence:

- The ability to change the conceptual schema without altering the external schema or application programs.
- Ensures that changes in the logical structure (e.g., adding new fields or tables) do not affect how end users interact with the data.

Physical Data Independence:

- The ability to change the internal schema without altering the conceptual schema.
- Ensures that changes in the physical storage (e.g., using a different file system or storage device) do not affect the overall organization or structure of the data as understood by the conceptual level.

Differences between Three Levels of ANSI-SPARC Architecture

1. **External Level:**
    - Provides tailored views of the database for individual users.
    - Hides parts of the data and presents it in a useful form.
2. **Conceptual Level:**
    - Manages the organization of data across the entire database.
    - Uses abstractions to simplify the complexities of the internal level.
3. **Internal Level:**

    - Focuses on the physical storage of data.
    - Details how records are stored on disk using files, pages, and blocks.

Data Independence

Logical Data Independence:

- Immunity of external schemas to changes in the conceptual schema.
- Conceptual schema changes (e.g., addition/removal of entities) should not require changes to external schemas or application programs.

Physical Data Independence:

- Immunity of the conceptual schema to changes in the internal schema.
- Internal schema changes (e.g., using different file organizations, storage structures/devices) should not require changes to the conceptual or external schemas.

Data Independence and the ANSI-SPARC Three-level Architecture

- **Logical Data Independence:** Achieved through the separation of the external level from the conceptual level.
- **Physical Data Independence:** Achieved through the separation of the conceptual level from the internal level.

Data Model

Data Model Comprises:

- **Structural Part:** Represents data structure.
- **Manipulative Part:** Represents operations on data.
- **Integrity Rules:** Ensures data correctness and validity.

Purpose:

- To represent data in an understandable way.

Relational Model

Introduced by E. F. Codd in 1970:

- Information is stored as tuples (records) in relations (tables).
- Most modern DBMS are based on the relational model.

Covers Three Areas:

- **Data Structure:** Organization of data.
- **Data Manipulation:** Operations on data.
- **Data Integrity:** Rules to maintain data accuracy.

Relational Model Terminology

- **Relation:** A table with columns and rows.
- **Attribute:** A named column of a relation.
- **Domain:** The set of allowable values for one or more attributes.
- **Tuple:** A row of a relation.
- **Degree:** The number of attributes in a relation.
- **Cardinality:** The number of tuples in a relation.
- **Relational Database:** A collection of relations with distinct names.

Properties of Relations

- **Distinct Relation Name:** Unique within a schema.
- **Atomic Values:** Each cell contains a single value.
- **Distinct Attribute Names:** Each attribute has a unique name.
- **Domain Consistency:** Values of an attribute are from the same domain.
- **Distinct Tuples:** No duplicate rows.
- **Insignificance of Order:** The order of attributes and tuples does not matter.

Keys in Relational Model

- **Candidate Key:** A minimal set of attributes uniquely identifying a tuple.
- **Primary Key:** A selected candidate key used to identify tuples.
- **Foreign Key:** An attribute set in one relation that matches a candidate key in another, enforcing referential integrity.

Referential Integrity

- **RESTRICT:** Prevents actions violating integrity constraints.
- **CASCADE:** Propagates changes through related tuples.
- **NULLIFY:** Sets problem values to NULL when constraints are violated.

Views

- **Base Relation:** A physically stored named relation in the database.
- **View:** A dynamic result of one or more relational operations on base relations.

Purpose of Views

- **Security Mechanism:** Hides parts of the database from certain users.
- **Customized Access:** Allows users to see data differently based on their needs.
- **Simplification:** Makes complex operations easier by presenting data in simpler forms.

Updating Views

- **Reflect Changes in Base Relations:** Updates in base relations should immediately reflect in views.
- **Propagate View Updates to Base Relations:** Changes made to views should affect the underlying base relations.

UNION, DIFFERENCE, INTERSECTION
binary operations - they take two
relations
The two relations must be union-compatible i.e same degree and matching domains (ith column of first relation and ith column of second relation have same domain)

Need for Permanent Data Storage

- Databases need to store data permanently for long periods of time.

File Organizations

- **File Organization:** The way records are stored in blocks and how blocks are placed on the storage medium.
- **Types:**
    - Unsorted
    - Sorted
    - Hashing

Records

- A file is a sequence of records.
- Records can be fixed-length or variable-length.
- Records comprise a sequence of fields (columns, attributes).

Blocks

- **Block:** The largest contiguous amount of disk space that can be allocated to a file and accessed in a single I/O operation.

Blocking Factor

- The number of records stored in a block is called the **blocking factor**.
- **Formula:** If B is the block size and R is the record size, then the blocking factor bfr is:
  $bfr = \lfloor RB \rfloor$
- Leftover space in each block:
  $\text{Leftover space} = B - (bfr \times R)$

Spanned and Unspanned Records

- **Unspanned:** Extra space in blocks is left unused.
- **Spanned:** Records can be split across blocks if the record size R is greater than the block size B.

Unordered File Organization (Pile Files)

- Records are inserted in the order of their arrival.
- Simplest file organization.

- Insertion is easy, but deletion causes fragmentation and is inefficient in terms of space usage.

Sorted Files

- Records are sorted based on an ordering field.
- Ordering field should be a key field and belong to an ordinal domain.
- Insertion and deletion are expensive.
- Efficient for key-based searches and read-only databases with key-based retrievals.

Hashing Techniques

- Provides fast access to records on certain search conditions, particularly equality conditions on a key field.
- Uses a hashing function to map keys onto buckets hosting records.
- Two levels of indirection: hashing to buckets and searching within buckets.

Handling Overflows

- Occurs when a bucket receives more records than it can hold.
- **Techniques:**
    - **Open Addressing:** Use the next available bucket.
    - **Chaining:** Pointers to extra overflow buckets.
    - **Rehashing:** Use a second hashing function to hash without overflow.

Dynamic Hashing

- Static hashing is inefficient with skewed data sets.
- Extensible or dynamic hashing allows the number of buckets to change dynamically based on data requirements.
- Buckets grow or shrink depending on additions and deletions.
- Strategy:
    - Start with a single bucket.
    - Split the bucket into two when full.
    - Redistribute records between the two split buckets to ensure near-uniform distribution.

Formulas

1. **Blocking Factor (bfr):**
   $bfr=\lfloor RB \rfloor$

2. **Leftover Space in Each Block:**
   $\text{Leftover space} = B - (bfr \times R)$

Example Questions and Answers

Question 1

Q: Define blocking factor and provide the formula to calculate it. **A:** The blocking factor is the number of records that can be stored in a block. The formula is:

$bfr = \lfloor RB \rfloor$

where B is the block size and R is the record size.

Question 2

Q: Explain the difference between spanned and unspanned records. **A:**

- **Unspanned Records:** Extra space in blocks is left unused; records do not span across multiple blocks.
- **Spanned Records:** Records can be split across multiple blocks if the record size exceeds the block size.

Question 3

Q: What are the key differences between pile files and sorted files? **A:**

- **Pile Files:**
  - Records are inserted in the order of arrival.
  - Easy insertion but deletion causes fragmentation.
- **Sorted Files:**
  - Records are sorted based on an ordering field.
  - Insertion and deletion are expensive.
  - Efficient for key-based searches and mostly read-only databases.

Question 4

Q: Describe how dynamic hashing works. **A:**

- Dynamic hashing allows the number of buckets to change dynamically based on data requirements.
- Start with a single bucket, split it into two when full, and redistribute records to ensure near-uniform distribution.

- Buckets grow or shrink depending on additions and deletions.

Question 5

Q: Explain the concept of overflow management in hashing. **A:**

- **Open Addressing:** Use the next available bucket when a bucket overflows.
- **Chaining:** Use pointers to extra overflow buckets.
- **Rehashing:** Use a second hashing function to handle overflows without conflicts.

Question: You have a database where each block on the storage medium has a size of 4096 bytes. Each record in the database is 512 bytes in size. Calculate the blocking factor and determine the leftover space in each block.

Answer:

To calculate the blocking factor (bfr) and the leftover space in each block, we can use the following formulas:

1. **Blocking Factor:**

   $bfr=\lfloor RB \rfloor$

   where:

   - B is the block size (4096 bytes).
   - R is the record size (512 bytes).

   Plugging in the values:

   $bfr=\lfloor 5124096 \rfloor=\lfloor 8 \rfloor=8$
2. **Leftover Space in Each Block:**

   $\text{Leftover space}=B-(bfr\times R)$

   Plugging in the values:

   $\text{Leftover space}=4096-(8\times 512)=4096-4096=0 \text{ bytes}$

Explanation:

- The blocking factor is 8, meaning each block can store 8 records.
- There is no leftover space in each block because the records perfectly fit into the block size.

Follow-up Question: What would be the leftover space if each record were 500 bytes instead of 512 bytes?

Answer to Follow-up:

1. **New Blocking Factor:**

   $bfr = \lfloor 5004096 \rfloor = \lfloor 8.192 \rfloor = 8$
2. **New Leftover Space in Each Block:**

   $\text{Leftover space} = 4096 - (8 \times 500) = 4096 - 4000 = 96 \text{ bytes}$

With each record being 500 bytes, the blocking factor remains 8, but now there are 96 bytes of leftover space in each block.

Indexes

- **Definition:** Secondary or auxiliary files that help speed up data access in primary files.
- **Types:**
  - **Single Level Index:** Index file maps directly to the block or the address of the record.
  - **Multi-Level Index:** Multiple levels of indirection among indexes.

Indexes as Access Paths

- **Definition:** A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- **Structure:** File of entries <field-value, pointer-to-record> ordered by field-value.
- **Function:** Binary search on the index yields a pointer to a file record.

Types of Indexes

1. **Primary Index:**

   - Defined on an ordered data file.
   - Data file is ordered on a key-field.
   - Includes one index for each block in the data file; the index entry has a key field value for the first record in the block (block anchor).
2. **Clustering Index:**

   - Defined on an ordered data file.
   - Data file is ordered on a non-key-field.
   - Includes one index entry for each distinct value of the field; points to the first data block containing records with that field value.
3. **Dense Index Files:**

    - Index record appears for every search-key value in the file.
4. **Sparse Index Files:**
    - Index records for only some search-key values.
    - Less space and maintenance overhead compared to dense indexes.
    - Generally slower for locating records than dense indexes.
5. **Secondary Index:**
    - Defined on an unordered data file.
    - Can be defined on a key field or a non-key field.
    - Includes one entry for each record in the data file.

Handling Duplicates in Secondary Indexes

- **Techniques:**
    - **Duplicate Index Entries:** Index entries are repeated for each duplicate occurrence of the non-key attribute.
    - **Variable Length Records:** Use variable length records for the index table to accommodate duplicate key entries.
    - **Extra Redirection Levels:** Use an additional level of indirection to handle duplicates.

Questions and Answers

1. What is the difference between a primary index and a secondary index?

- **Primary Index:** Defined on an ordered data file and is based on a key field. It includes one index for each block in the data file.
- **Secondary Index:** Defined on an unordered data file and can be based on either a key or non-key field. It includes one entry for each record in the data file.

2. Why might a sparse index be preferable to a dense index?

- A sparse index might be preferable because it uses less space and has lower maintenance overhead for insertions and deletions. However, it is generally slower than a dense index for locating records.

3. How does dynamic hashing differ from static hashing?

- **Static Hashing:** Number of buckets is fixed. Inefficient when data set is skewed.
- **Dynamic Hashing:** Number of buckets changes dynamically in accordance with data requirements. Allows the hash structure to grow or shrink as data is added or deleted.

4. Provide an example of a situation where a clustering index would be used.

- A clustering index would be used when a data file is ordered on a non-key field, such as ordering students by their department. Each department is a non-key field and the index would point to the first data block that contains records for that department.

5. Calculate the blocking factor for a block size of 2048 bytes and a record size of 256 bytes.

Blocking Factor (bfr)$=\lfloor RB \rfloor=\lfloor 2562048 \rfloor=\lfloor 8 \rfloor=8$

6. What are the potential issues with using variable length records in indexing?

- Potential issues include increased complexity in managing and accessing records, potential for fragmentation, and the need for additional metadata to keep track of record sizes and locations.

7. Describe a scenario where extra redirection levels might be necessary in a secondary index.

- Extra redirection levels might be necessary in a secondary index when indexing a non-key field with many duplicate values, such as indexing a list of students by their major, where many students have the same major. Using extra redirection levels helps efficiently manage and locate these duplicates.

8. Explain the concept of blocking factor and its significance in database storage.

- **Blocking Factor:** The number of records that are stored in a block. It is significant because it determines how many records can be accessed with a single I/O operation. Higher blocking factors reduce the number of I/O operations needed to access a set of records, improving performance.

9. Why is it generally more efficient to use a binary search on an index file than on the data file itself?

- It is more efficient because the index file is usually smaller and occupies fewer disk blocks than the data file. A binary search on the smaller index file yields quicker results and provides a pointer to the specific location in the larger data file, reducing the overall search time.

10. Given a data file with records ordered by a key field and a primary index defined on this key field, what happens if a new record is inserted?

- If a new record is inserted, the primary index needs to be updated to include an entry for the new record. If the data file is densely packed, this may involve shifting records to maintain order, which can be computationally expensive. The index must then be updated to reflect the new block structure and record locations.

Formulas

- **Blocking Factor:**
  $bfr=\lfloor RB \rfloor$
- **Leftover Space:**
  $\text{Leftover space}=B-(bfr\times R)$

Summary

- Storage of databases requires permanent and efficient organization.
- Indexes improve data access speed and come in various types (primary, clustering, dense, sparse, secondary).
- Hashing and dynamic hashing provide quick access methods but require appropriate handling of overflows and dynamic data changes.
- Blocking factor and efficient file organization are crucial for optimizing database performance and storage.

Indexes

- **Definition:** Secondary or auxiliary files that help speed up data access in primary files.
- **Types:**
  - **Single-Level Index:** Index file maps directly to the block or the address of the record.
  - **Multi-Level Index:** Multiple levels of indirection among indexes.

Multi-Level Indexes

- **Definition:** A multi-level index is a form of a search tree that can be created for any type of first-level index as long as the first-level index consists of more than one disk block.
- **Process:** Create a primary index for the index itself, and repeat until the top-level index fits in one disk block.

- **Challenges:** Insertion and deletion of new index entries are challenging because each level of the index is an ordered file.

Index Update

- **Deletion:**
  - **Dense Indices:** Similar to file record deletion.
  - **Sparse Indices:** Replace the deleted entry with the next search-key value, or delete if the next value already has an index entry.
- **Insertion:**
  - **Dense Indices:** Insert the new search-key value if it does not appear.
  - **Sparse Indices:** No change unless a new block is created; insert the first search-key value of the new block.

Bitmap Index

- **Definition:** A bitmap index for attribute A of relation R is a collection of bit-vectors where the number of bit-vectors equals the number of distinct values of A in R, and the length of each bit-vector equals the cardinality of R.
- **Example:**

  css

```
A, B
30, foo
30, bar
40, baz
50, foo
40, bar
30, baz
```

mathematica

Value	Vector
foo	100100
bar	010010
baz	001001

Dynamic Multilevel Index

- **Definition:** To reduce index insertion and deletion overhead while retaining the benefits of multilevel indexing, a dynamic multilevel index leaves some space in each block for new entries.
- **Implementation:** Often implemented using B-trees and B+ trees.

B-Trees

- **Definition:** A tree data structure where each node has a predetermined maximum fan-out (order) p.
- **Terminologies:** Root node, leaf nodes, internal nodes, parent, children.
- **Structure of a Node:**

  css

-
  ```
  Data        Pointers
  Left-most   K1 K2 ... Ki-1 Ki
  Subtree
  ```

B-Tree Insertion

- **Process:** Insert a key into the appropriate leaf node. If the leaf node overflows, split it and propagate the middle key up to the parent node. Repeat until no overflow occurs or the root is split.

B+ Trees

- **Definition:** A type of B-tree where all values are at the leaf level and internal nodes store pointers only.
- **Structure:**

  perl

- Internal Node: Only stores keys and pointers.
  Leaf Node: Stores keys and associated data or pointers.

Formulas

Blocking Factor

- **Formula:**
  Blocking Factor (bfr)=$\lfloor RB \rfloor$
  Where B is the block size and R is the record size.
- **Leftover Space:**
  Leftover space=$B-(bfr \times R)$

Order of B-Tree and B+ Tree

- **Order (p):** The maximum number of children nodes a B-tree or B+ tree node can have.
- **Minimum keys in a node:** $\lceil 2p \rceil - 1$

- **Maximum keys in a node:** $p - 1$
- **Minimum children in a node:** $\lceil 2p \rceil$
- **Maximum children in a node:** p

Questions and Answers

1. What is a multi-level index and why is it used?

- A multi-level index is a form of a search tree used to reduce the number of disk blocks accessed when searching for a record. It is used to make searching more efficient, especially for large datasets.

2. Explain the difference between a dense index and a sparse index.

- **Dense Index:** Contains an index entry for every search-key value in the file.
- **Sparse Index:** Contains index entries for only some search-key values. Less space and maintenance overhead but generally slower for locating records.

3. How does a bitmap index work? Provide an example.

- A bitmap index uses bit-vectors to represent the presence of attribute values in records. Each bit-vector corresponds to a distinct attribute value, and each bit indicates whether the corresponding record has that attribute value. **Example:**

  css

```
A, B
30, foo
30, bar
40, baz
50, foo
40, bar
30, baz
```

mathematica

Value	Vector
foo	100100
bar	010010
baz	001001

4. Calculate the blocking factor for a block size of 2048 bytes and a record size of 256 bytes.

Blocking Factor (bfr)=$\lfloor 2562048 \rfloor$=$\lfloor 8 \rfloor$=8

5. Describe the process of inserting a key into a B-tree.

- **Step-by-step Process:**
    - Find the appropriate leaf node for the key.
    - Insert the key into the leaf node in sorted order.
    - If the leaf node overflows, split the node and propagate the middle key up to the parent node.
    - Repeat the splitting process until no overflow occurs or the root is split to create a new root.

6. How does a B+ tree differ from a B-tree?

- In a B+ tree, all values are stored at the leaf level, and internal nodes store pointers only. This makes B+ trees more efficient for range queries as leaf nodes are linked.

7. What are the advantages of using dynamic multilevel indexing?

- **Advantages:**
    - Reduces insertion and deletion overhead by leaving space in each block for new entries.
    - Maintains the benefits of multilevel indexing, such as efficient search and update operations.
    - Dynamic adjustment of the index structure as data is added or removed.

8. Given a B-tree of order 4, what is the maximum number of keys and children a node can have?

- **Maximum number of keys:** $4-1=3$
- **Maximum number of children:** 4

9. Why might a sparse index be preferable to a dense index in some situations?

- A sparse index might be preferable because it uses less disk space and requires less maintenance overhead for insertions and deletions. It is more efficient when there are fewer search-key values compared to the number of records.

10. Provide a scenario where a clustering index would be beneficial.

- A clustering index is beneficial in scenarios where data retrieval is based on a non-key field, such as grouping students by their major in a university database. The clustering index allows efficient access to all students within the same major.

Tricky Questions

1. Why might insertion into a multi-level index be more complex than insertion into a single-level index?

- Insertion into a multi-level index is more complex because it requires maintaining order at multiple levels. Each level must be updated to reflect the changes, which can involve splitting nodes and propagating changes upwards.

2. How would you handle the deletion of a search-key value in a sparse index if the next search-key value already has an index entry?

- If the next search-key value already has an index entry, the entry for the deleted search-key value is removed without replacement. This may require adjusting pointers to maintain the index's structure.

3. Explain how dynamic hashing handles skewed data distribution more efficiently than static hashing.

- Dynamic hashing adjusts the number of buckets based on data requirements, allowing it to handle skewed data distribution more efficiently. It splits buckets when they overflow and merges buckets when they underflow, ensuring a near-uniform distribution of records.

4. What challenges arise when handling duplicate values in a secondary index on a non-key field?

- Challenges include managing multiple index entries for the same value, which can lead to increased storage requirements and complexity in maintaining the index. Techniques like duplicate index entries, variable length records, and extra redirection levels are used to address these challenges.

5. Given a B+ tree where each internal node can hold up to 5 pointers, what is the minimum and maximum number of keys each internal node can have?

- **Minimum number of keys:** $\lceil 25 \rceil - 1 = 2 - 1 = 1$
- **Maximum number of keys:** $5 - 1 = 4$