Lab 9: Indexes

Part 1 - Selecting Indexes

A database contains the following table for former-employee records

- Minimal set of additional secondary indexes: composite index on ("Start Date", "End Date")
- Reasoning: The composite index on ("Start Date", "End Date") will efficiently support these queries. The "Start Date" part of the index will help find all employees who started on a certain date, and the "End Date" part of the index will assist in identifying those employees whose employment period includes the specified end date. Creating a single composite index saves us from adding separate indexes for each date column, reducing index maintenance overhead.

A database contains the following table for tracking student grades in classes

1. Get all students with a grade better than 'B':
   - Additional Index: (Grade)
2. Get all classes where any student earned a grade worse than 'D':
   - Additional Index: (Grade)
3. Get all classes ordered by class name:
   - Additional Index: (className)
4. Get all students who earned an 'A' in a certain class:
   - Additional Index: (className, Grade)

After merging additional index from 3 & 4, we get Index: (className, Grade)

Summary of Additional Indexes: - (Grade) - (className, Grade)

Queries on the chess database

1. `select Name from Players where Elo >= 2050;`
   - Additional Index: `Players.Elo`
2. `select Name, gID from Players join Games where pID = WhitePlayer;`
   - Existing Index: `Players.pID` (as it is the default primary key of Players table and already indexed).
   - Additional Index: `Games.WhitePlayer`

Summary of Additional Indexes: - `Players.Elo` - `Games.WhitePlayer`

Queries on the public Library database

Analysis: - The `Inventory` table has a primary key (PK) on the `Serial` column. - The `CheckedOut` table also has a primary key (PK) on the `Serial` column. - Since `Serial` is the only shared column between `Inventory` and `CheckedOut`,

and both tables have their primary keys indexed by default, there is no need to add any additional indexes for this specific query.

Summary: Based on the existing primary key indexes on the `Serial` column in both `Inventory` and `CheckedOut` tables, no additional indexes are needed to support the query.

More library queries

1. `select * from Inventory natural join CheckedOut where CardNum = 2;`
   - Natural join between `Inventory` and `CheckedOut` depends on the common primary key `Serial`.
   - Additional Index: `CheckedOut.CardNum`
2. `select * from Patrons natural join CheckedOut;`
   - `Patrons` PK is `CardNum`.
   - `CheckedOut` PK is `Serial`.
   - `Patrons` and `CheckedOut` share the common column `CardNum`.
   - Additional Index: `Patrons.CardNum`

Summary of Additional Indexes: - `CheckedOut.CardNum` - `Patrons.CardNum`

Still more library queries

1. `Titles` PK is `ISBN`, so an additional index on `Titles.Title` is needed.
   - Additional Index: `Titles.Title`
2. `Inventory`'s `Serial` is already a primary key, so it doesn't need to be added as an index.
   - No additional index needed for `Inventory.Serial`.

Summary: - `Titles.Title`

Part 2 - B+ Tree Index Structures

How many rows of the table can be placed into the first leaf node of the primary index before it will split?

The primary index is created in the order `(studentID, className)`. Let's calculate the size of each row in the index:

- `studentID` is an int, which occupies 4 bytes.
- `className` is a varchar(10), which occupies 10 bytes.
- `Grade` is a char(1), which occupies 1 byte.

Each row in the index occupies 15 bytes. The leaf node of the primary index has a size of 4096 bytes. Therefore, the number of rows that can be placed into the first leaf node of the primary index before it will split is `4096 / 15 = 273`.

What is the maximum number of keys stored in an internal node of the primary index? (Remember to ignore pointer space. Remember that internal nodes have a different structure than leaf nodes.)

Apologies for the confusion in my previous response. You are correct. The primary index is created in the order `(studentID, className)`.

Let's calculate the size of each key in the index and then determine the maximum number of keys that can be stored in an internal node:

- `studentID` is an int, which occupies 4 bytes.
- `className` is a varchar(10), which occupies 10 bytes.

Each key in the primary index will consist of both `studentID` and `className`. Therefore, the size of each key in the index will be `4 bytes (studentID) + 10 bytes (className) = 14 bytes`.

An internal node has a size of 4096 bytes, and we ignore any additional overhead data or pointer space, the maximum number of keys can be calculated as follows:

Available space in the internal node = 4096 bytes

Size of each key = 4 + 10 = 14 bytes

Maximum number of keys = floor(4096 bytes / 14 bytes) = 292

What is the maximum number of rows in the table if the primary index has a height of 1? (A tree of height 1 has 2 levels and requires exactly one internal node)

If the primary index has a height of 1, it means that the index consists of only one internal node and one level of leaf nodes. In this case, the primary index is essentially a flat index, and there are no intermediate levels of internal nodes between the root and the leaf nodes.

The height of the primary index is determined by the number of keys stored in an internal node, which we previously calculated to be approximately 292 keys.

Therefore, the maximum number of rows in the table with a primary index height of 1 is approximately 292 rows.

What is the minimum number of rows in the table if the primary index has a height of 1? (A tree of height 1 has 2 levels). The minimum capacity of a node in a B+ tree is 50%, unless it is the only internal/leaf node. The minimum number of children of a root node is 2.

In a B+ tree with a height of 1, we have one root node and one level of leaf nodes. Since it is the only internal node, its minimum capacity is not subject to the 50% rule.

The minimum number of children of a root node is 2. In other words, the leaf nodes must have at least two nodes.

Thus, the minimum number of rows in the table when the primary index has a height of 1 is at least 2 rows.

If there is a secondary index on Grade, what is the maximum number of entries a leaf node can hold in the secondary index?

Since the secondary index is created on the `Grade` column, each entry in the index will consist of the `Grade` value and the associated pointers to the rows in the main table.

Let's assume the following: - `Grade` is a `char(1)` column, which occupies 1 byte. - The size of each pointer (pointer to a row in the main table) is `P` bytes.

Available space in the leaf node = 4096 bytes

Size of each entry = 1 byte (Grade) + P bytes (pointer)

Maximum number of entries in the leaf node = 4096 bytes / (1 byte + P bytes)

Therefore, the maximum number of entries a leaf node can hold in the secondary index on the `Grade` column will be: 4096 / (1 + P), Where `P` represents the size of a pointer in bytes (e.g., 4 bytes or 8 bytes).

Another table

Given: - Page size: 4096 bytes - Row size: 128 bytes - Number of rows in the table: 48

Let's assume the B+ Tree's order (maximum number of keys in a node) is denoted by 'B'.

What is the maximum number of leaf nodes in the primary index if the table contains 48 rows? The leaf node contains [ceil(B/2), B] records.

Thus the maximum number of leaf nodes in the primary index is ceil(48 / ceil(B/2)).

What is the minimum number of leaf nodes in the primary index if the table contains 48 rows? The minimum number of leaf nodes in the primary index is ceil(48 / B).