Lab 9: Indexes

Part 1 - Selecting Indexes

A database contains the following table for former-employee records

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.

Summary of Additional Indexes:

- Index("Start Date", "End Date")

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:

- None

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`.
   - `Patrons.CardNum` is already the primary key of `Patrons` table, so it is already indexed.

Summary of Additional Indexes:

- `CheckedOut.CardNum`

Still more library queries

`Serials = from i in t.Inventory` joins Titles with Inventory on Titles.ISBN = Inventory.ISBN.

Titles has a primary key on ISBN, so it is already indexed. Inventory has a primary key on Serial, so we still need to index on Inventory.ISBN.

Summary of Additional Indexes:

- `Inventory.ISBN`

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 floor(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.)

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)

From the last question we know that the B+ tree's order is 292 + 1 = 293. And the max keys per internal node could hold is 292 keys.

The max leaf node number is 293 leaf nodes * 273 rows per leaf node = 79989 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.

The minimum number of leaf nodes is 2. The minimum number of rows per leaf node is ceil(273/2) = 137. Thus the minimum number of rows in the table is 2 * 137 = 274.

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

The Grade serves as the primary key, takes 1 byte. The pointer to the record takes 4 bytes. The total size of each entry is 5 bytes. The leaf node has a size of 4096 bytes. Therefore, the maximum number of entries a leaf node can hold in the secondary index is floor(4096 / 5) = 819.

Another table

What is the maximum number of leaf nodes in the primary index if the table contains 48 rows?

A leaf node could hold maximum floor(4096/128) = 32 rows. A leaf node should at least hold ceil(32/2) = 16 rows. The maximum number of leaf nodes is ceil(48/16) = 3.

What is the minimum number of leaf nodes in the primary index if the table contains 48 rows?
The minimum number of leaf nodes is ceil(48/32) = 2.