
LUT
University
LAND
OF THE
CURIOUS

CT60A4304 - BASICS OF DATABASE SYSTEMS

INDEX AND OPTIMIZATION

Lecture

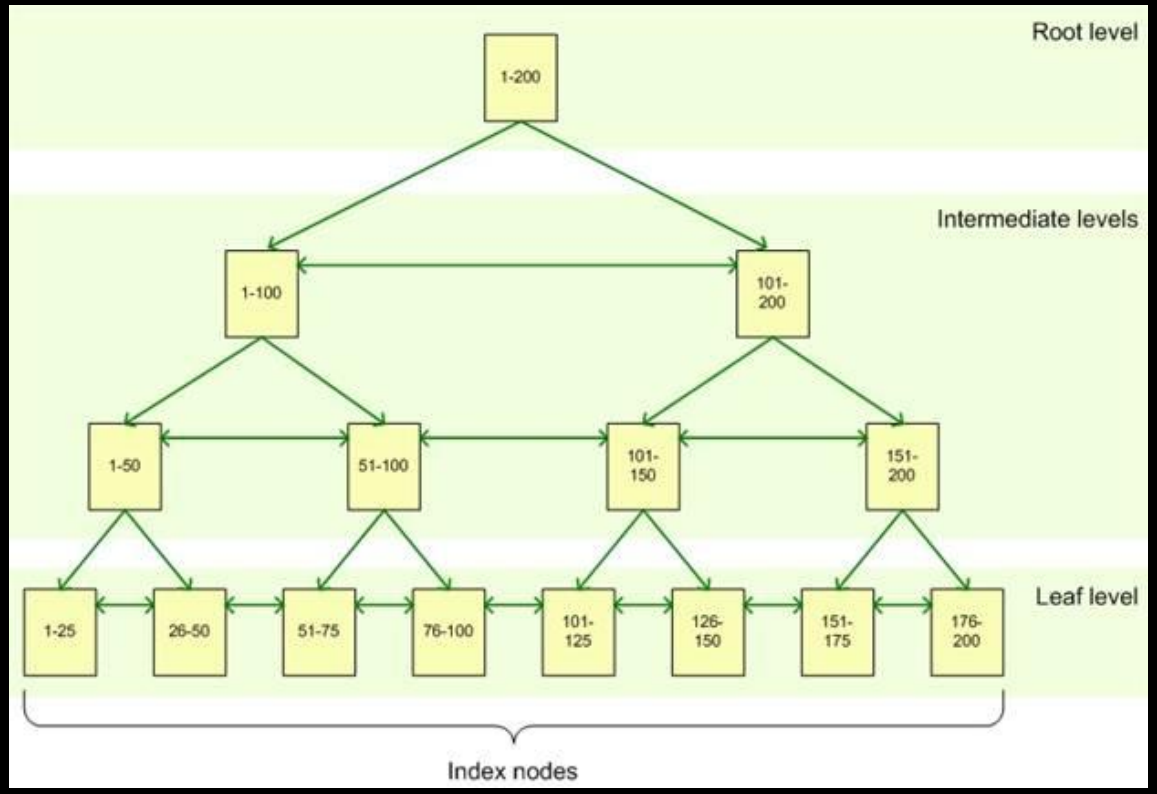
Jiri Musto, D.Sc.

TABLE OF CONTENTS

INDICES / INDEXES IN GENERAL

- Dictionary style data structure
- Purpose is to have faster queries
- No need to go through all rows, indices point to the “correct” rows and enables faster retrieval
- How indices work
  - B-tree (balanced tree) algorithm
  - Speeds up SELECT queries with WHERE statements
  - Makes UPDATE and INSERT slower
  - Can be created or dropped with no effect on data

HOW INDICES WORK



https://www.red-gate.com/simple-talk/databases/sql-server/learn/sql-server-index-basics/

WHEN TO USE INDICES

- Columns are often used in search conditions (WHERE or JOIN statements)
- Columns are often used in ordering results
- Column has a wide range of values
- Column has lots of null values
- Table is large and most queries retrieve 2 to 4 % of rows

WHEN NOT TO USE INDICES

- Table is small
- Most queries retrieve more than 2 to 4 % of rows
- Table is often updated
  - Indices can be removed temporarily when performing updates
- Columns are rarely used for queries

- These are basic guidelines, you should test what works for you

INDICES IN SQL

- CREATE INDEX index_name ON table_name(columns);
- The order of columns is important!
  - In most databases, columns are used in-order
- DROP INDEX index_name

```
 9  CREATE INDEX PlayerIndex ON Player(last_name);
10  CREATE INDEX RankingIndex ON Player(rank, score);
11
```

QUERY PLANNER

- Database optimizes the query and makes a query plan based on the SQL query and database structure
- Each DBMS has commands for timers and viewing query plans
  - SQLite, PostgreSQL and MySQL has EXPLAIN (QUERY PLAN)
- SQLite timer is: .timer on
- Shows three different times: real, user, sys
  - real time is the elapsed time
  - user time is the time spent executing instructions in user mode
  - sys time is the time spent executing instructions in supervisor mode

EXPLAIN QUERY PLAN

- View the steps of each query

```
QUERY PLAN
|--SCAN M1
|--MULTI-INDEX OR
|   |--INDEX 1
|   |   `--SEARCH P1 USING INTEGER PRIMARY KEY (rowid=?)
|   `--INDEX 2
|       `--SEARCH P1 USING INTEGER PRIMARY KEY (rowid=?)
|--MULTI-INDEX OR
|   |--INDEX 1
|   |   `--SEARCH P2 USING INTEGER PRIMARY KEY (rowid=?)
|   `--INDEX 2
|       `--SEARCH P2 USING INTEGER PRIMARY KEY (rowid=?)
|--SEARCH R1 USING AUTOMATIC COVERING INDEX (FK_playerid=?)
|--SEARCH R2 USING AUTOMATIC COVERING INDEX (FK_playerid=?)
`--USE TEMP B-TREE FOR ORDER BY
Run Time: real 0.002 user 0.000000 sys 0.000000
sqlite>
```

```
QUERY PLAN
|--MATERIALIZE SUBQUERY 2
|   `--COMPOUND QUERY
|       |--LEFT-MOST SUBQUERY
|       |   `--SCAN Matches
|       `--UNION USING TEMP B-TREE
|           `--SCAN Matches
|--SCAN LoserRanking
|--SEARCH SUBQUERY 2 USING AUTOMATIC COVERING INDEX (LoserID=?)
|--SEARCH Winner USING INTEGER PRIMARY KEY (rowid=?)
|--SEARCH Loser USING INTEGER PRIMARY KEY (rowid=?)
|--SEARCH WinnerRanking USING AUTOMATIC COVERING INDEX (FK_playerid=?)
`--USE TEMP B-TREE FOR ORDER BY
Run Time: real 0.007 user 0.000000 sys 0.000000
sqlite>
```

```
QUERY PLAN
|--SCAN M
|--SEARCH W USING INTEGER PRIMARY KEY (rowid=?)
|--SEARCH Ws USING INTEGER PRIMARY KEY (rowid=?)
|--SEARCH L USING INTEGER PRIMARY KEY (rowid=?)
|--SEARCH Ls USING INTEGER PRIMARY KEY (rowid=?)
`--USE TEMP B-TREE FOR ORDER BY
Run Time: real 0.003 user 0.000000 sys 0.000000
```

OPTIMIZING QUERIES

- Less data you retrieve, the better
  - Limit the amount of rows and columns
- Use the minimum amount of queries so optimizer can work its magic
  - But remember, optimizer is not perfect
- IN condition is slow compared to “ < AND > “ operation
  - NOT queries are also slower than regular ones
- Use indices to speed up the most important parts of queries
  - Over doing indices may slow the queries
- Do not store large amounts of binary data in a database (use a separate file)

JOIN VS SUBQUERY

- JOIN is generally faster than subquery
- NOT because JOIN is inherently faster
- ...But because automatic query optimizer is better at handling JOINs. Subqueries need more manual optimization
- Correlated subqueries are slow
  - Subquery that is run on each row, often inside the outer SELECT statement
- Both are valid options. JOINs are easier to use in general but sometimes a subquery may speed up the process if done correctly

OPTIMIZATION TECHNIQUES

- Mass insert is always faster than multiple inserts
  - One insert adding 1000 rows vs. 1000 inserts adding one row
- There are useful functions, such as conversion, substrings and dateparts for query conditions
  - These functions slow down the query
- Comparing different data types in SQL queries requires the conversion of one type to match another
  - Implicit conversion is done for the whole table before the query is executed


LUT
University