Class 5 BigQuery
Elements of Databases
Feb 25, 2022
Database Indexes

- Critical for many databases
- At least one index per table
- DBA analyzes workload and chooses which indexes to create (no easy answers)
- Creating indexes can be an expensive operation
- They work “behind the scenes”
- Query optimizer decides which indexes to use during execution

```
CREATE INDEX empid_idx ON Employee(empid);
CREATE INDEX empid_idx ON Employee(empid, salary);
```
B-Trees

- Standard index implementation in most database systems
- Keep key-value pair sorted by key
- Designed to speed up lookups and range queries
- One tree node maps to one disk page (4KB - 8KB)
- A node is packed with index entries (typically 100+)
- Index entry = (key, reference)
- High branching factor = references to child nodes (100+)
- Search speed ≈ height of tree
- Height of tree is $O(\log n)$ where $n =$ index keys
Postgres Code Lab, Part 3

- Clone snippets repo
- Open postgres idx notebook
- Generate an explain plan
- Use \timing
- Create an index to speed up a query
Global Aggregate Queries

SELECT <aggregate function>
    [, <aggregate function>]
FROM <single table>
[JOIN <single table>
    ON <join condition>]
[WHERE <boolean condition>]
ORDER BY <field(s) to sort on>
Global Aggregate Queries

```
SELECT <aggregate function> [, <aggregate function>] 
FROM <single table> 
[JOIN <single table> 
  ON <join condition>] 
[WHERE <boolean condition>] 
ORDER BY <field(s) to sort on>
```
Group By Queries

```
SELECT <unaggregated field(s)>
FROM <single table>
[JOIN <single table>
ON <join condition>]
[WHERE <boolean condition>]
GROUP BY <unaggregated field(s)>
```
Aggregate Group By Queries

SELECT <unaggregated field(s)>,
    <aggregate function(s)>
FROM <single table>
[JOIN <single table>
    ON <join condition>]
[WHERE <boolean condition>]
GROUP BY <unaggregated field(s)>
[HAVING <boolean condition>]
[ORDER BY <field(s) to sort on>]
Aggregate Group By Queries

```
SELECT <unaggregated field(s)>,
    <aggregate function(s)>
FROM <single table>
[JOIN <single table>
    ON <join condition>]
[WHERE <boolean condition>]
GROUP BY <unaggregated field(s)>
[HAVING <boolean condition>]
[ORDER BY <field(s) to sort on>]
```
The semantics of \texttt{COUNT()} 

\texttt{SELECT COUNT(*)} 
FROM Employee

\texttt{SELECT COUNT(department)} 
FROM Employee

\texttt{SELECT DISTINCT department} 
FROM Employee

\texttt{SELECT COUNT(DISTINCT department)} 
FROM Employee

<table>
<thead>
<tr>
<th>row</th>
<th>employee</th>
<th>department</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunil</td>
<td>ENG</td>
</tr>
<tr>
<td>2</td>
<td>Morgan</td>
<td>ENG</td>
</tr>
<tr>
<td>3</td>
<td>Rama</td>
<td>Product</td>
</tr>
<tr>
<td>4</td>
<td>Drew</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Jeff</td>
<td>Research</td>
</tr>
<tr>
<td>6</td>
<td>Danielle</td>
<td>HR</td>
</tr>
<tr>
<td>7</td>
<td>Grace</td>
<td>ENG</td>
</tr>
</tbody>
</table>
BigQuery Overview

- Data warehouse / analytics database service
- Distributed database system
- Optimized for large data (petabyte-scale)
- Data model: tables with optional nesting
- Query language: standard SQL
- Data Types:
  - Primitive: BOOL, BYTES, FLOAT64, INT64, NUMERIC, STRING
  - Temporal: DATE, DATETIME, TIME, TIMESTAMP
  - Geospatial: GEOGRAPHY
  - Complex: ARRAY, STRUCT
- No provisioning needed, easy to use
- Not an operational database, no referential integrity
## Nested Columns

<table>
<thead>
<tr>
<th>personId</th>
<th>name</th>
<th>gender</th>
<th>cityLived (nested and repeated)</th>
<th>state</th>
<th>country</th>
<th>phone</th>
<th>email</th>
</tr>
</thead>
</table>

- **cityId**
- **cityName**
- **startDate**
- **endDate**

**ARRAY + STRUCT type**
BigQuery Architecture*

- **Replicated, Distributed Storage** (high durability)
- **Petabit Network**
- **High-Available Cluster Compute** (Dremel)
- **Streaming Ingest**
- **Bulk Loading**
- **Distributed Memory Shuffle Tier**
- **REST API**
- **Web UI, CLI**
- **Client Libraries In 7 languages**

* Very approximate
BigQuery Code Lab

- Clone snippets repo
- Open bigquery notebook
- Create college dataset
- Populate college tables
- Explore the data
- Write aggregate queries
Exercise 1: Group-by queries

For each class in the database, obtain the number of students taking the class.

Return the cno for the class along with its enrollment count.

Sort the results by enrollment in descending order.

Database Schema:
Student(sid, fname, lname, dob, status)
Class(cno, cname, credits)
Instructor(tid, name, dept)
Takes(sid, cno, grade)
Teaches(tid, cno)
Exercise 2: Group-by queries

For each class in the database which has at least two students enrolled, how many students are taking the class?

Return the cno for the class along with its enrollment count.

Sort the results by enrollment in descending order.

Database Schema:
Student(sid, fname, lname, dob, status)
Class(cno, cname, credits)
Instructor(tid, name, dept)
Takes(sid, cno, grade)
Teaches(tid, cno)
Project 4: explore K-12 enrollment data

Assignment sheet

- Open [project4 notebook](#)
- Run starter code:
  - Create and populate tables
  - Explore the data
  - Write aggregate query
  - Create database view
  - Created data visualization