School of Computer Science

Web and Database Computing 2019

Lecture 32: SQL Database Optimisation

adelaide.edu.au

seek LIGHT

Optimisation Overview

As the size of the data in our database increases, queries can become more expensive and time-consuming.

- Poorly written queries can cause a sigificant performance impact if left unchecked.
    - This can result in blocking database resources, preventing other queries from running.
    - That in-turn delays HTTP requests being handled on the server, using up the system's available connection pool.
    - If the TCP connection pool for a given server becomes exhaused, the server will no-longer be able to handle requests.
    - Fun fact; this is what makes DDOS attacks effective.
- Poorly written queries can also be expensive; especially if your host is charging you for use of system resources.

Understanding how queries work and how we can debug/improve them will help us to improve our website's user experience and allow it to handle more traffic.

Query Execution

SQL Order of Operations

Understanding how SQL evalutes queries is the first step in understanding how we can improve our queries.

SQL queries are *logically* executed according to the following order of operations:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT

SQL Order of Operations

1. FROM

- The first thing that happens is the tables to be used in the query are opened, and, if needed, joined.
- This means a JOIN is one of the first things to happen in a SQL query.

2. WHERE

- Next, unnecessary rows are filtered out using WHERE.

3. GROUP BY

- Aggregate data from the remaining rows is now grouped together.

4. HAVING

- Results can now be further filtered using aggregate data.

SQL Order of Operations

5. SELECT

- Columns that will not be in the result are now excluded

6. ORDER BY

- These remaining rows & columns can now be sorted.

7. LIMIT

- We can now retrieve the top results from the remaining rows if not all rows to be returned.

Implications

So what does this all mean?

- Joins have a large data overhead;
    - Minimise joins across more than 2 tables if possible.
- WHERE operations apply on a large number of rows
    - If we can limit the rows before the query, it may run better.
    - Improving the efficiency of searching through our remaining rows can also help here.
- ORDER BY needs to sort the remaining rows.
    - If we can pre-sort the data, this operation will be much faster.

What actually happens

The SQL Server does however automatically perform a number of optimisations to make this run faster.

- As long as the result is the same, the order of operations can be changed.
    - A WHERE may be executed before a JOIN if the result will be the same.

Checking performance

We can use the EXPLAIN operation to display information about how a query will be executed.

- The explain operation breaks down each select operation that will be needed for the query and can tell us information such as:
  - How many rows will need to be examined,
  - Whether any keys were able to be used for optimisations, or
  - If table was able to use and index if avialable
  - etc.

```
EXPLAIN SELECT * FROM Customers WHERE cust_name LIKE "%John%";
```

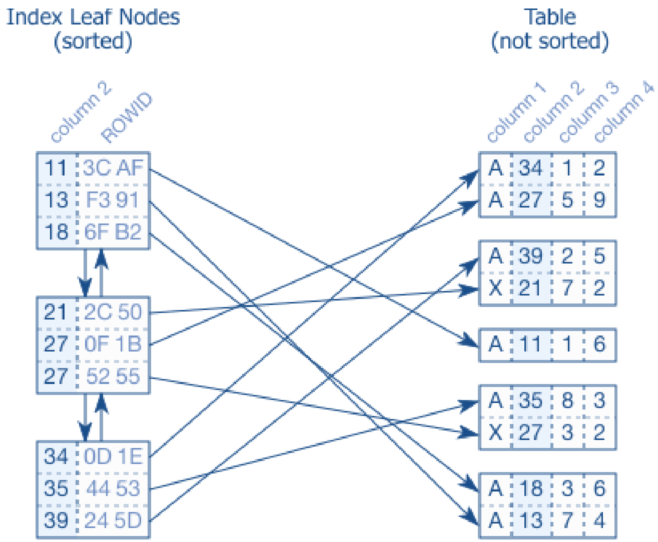
See https://www.sitepoint.com/using-explain-to-write-better-mysql-queries/
and https://www.eversql.com/mysql-explain-example-explaining-mysql-explain-using-stackoverflow-data/

Improving Performance with Indexes

If we expect to filter on a given column regularly, we may want to index the column.

- Indexing creates a sorted duplicate of the data that is linked to original actual table.
    - Sorted data can be more easily searched and reordered.
    - The downside is that insert and updates take longer because new indexes need to be created/updated.

```
CREATE INDEX CustName ON Customers(cust_name);
```



From https://use-the-index-luke.com/

- Can work with multiple columns:

```
CREATE INDEX CustName ON Customers(cust_name_given,cust_name_family);
```

Avoiding Queries that are not compatible with Indexes

Indexes allow us to take advantage of sorted data, but some WHERE conditions are not able to be sorted

```
SELECT * FROM Customers WHERE cust_name LIKE "%John%";
```

In the above case, the wildcard `%` operator at the start means that we can't take advantage of a sorted name index.

If possible, avoid these types of queries.

- There are other options for dealing with this such as indexed full-text search.
- Use EXPLAIN to help determine if your query doesn't use an index.

Prefiltering table results with Common Table Expressions

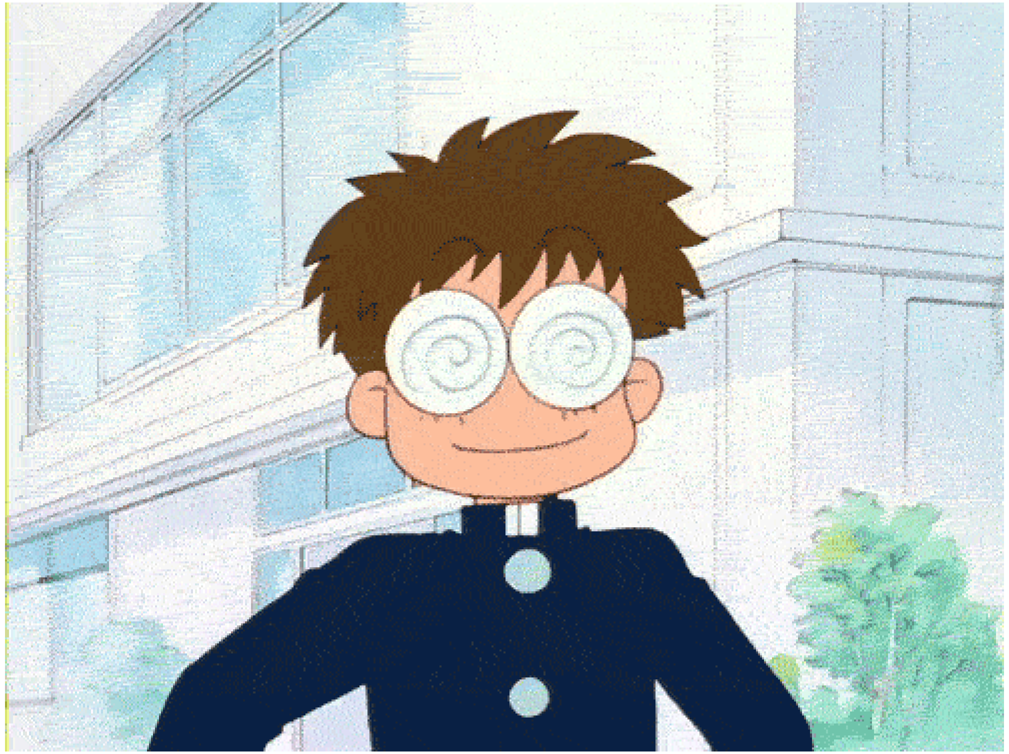
Common Table Expressions can be used to create temporary tables for later use in a query:

```
WITH
  TempCusts AS (SELECT cust_id, name FROM Customers WHERE cust_name = 'John'),
  TempOrds AS (SELECT cust_id, item_name, item_price FROM Orders WHERE item_name
SELECT SUM(item_price) FROM TempCusts
  INNER JOIN TempOrds
  ON TempCusts.cust_id = TempOrds.cust_id;
```

- The WITH keyword allows us to define aliases for query results that are treated as a virtual table.
    - Similar to Views, but only last for the duration of the Query.
    - We can exclude columns not needed, and use WHERE to pre-filter rows.

Questions?

Quiz!



Today's Quiz is available in MyUni until after Week 13, *Questions and all*

THE UNIVERSITY
of ADELAIDE

What's happening

Due:

- Prac Exercise 8 due Today.
- Prac Exercises 9 & 10 available soon - Due end of semester.

Next week:

- Security

Further learning:

- Keep working on your group projects
- Check out this article on query optimisation