Sujan Aryal
Julian Fung
Harsha
Anurag Baral

CS430: Project Submission Part 3

1. **Front-end source code used to:**
    a. **Populate database:**
        i. We used a random data generator(mockaroo.com) to generate random data and populate our database and had 1.55 million records in both SQL and Mongo, therefore we do not have source code for this.
    b. **Measure delay in search**:

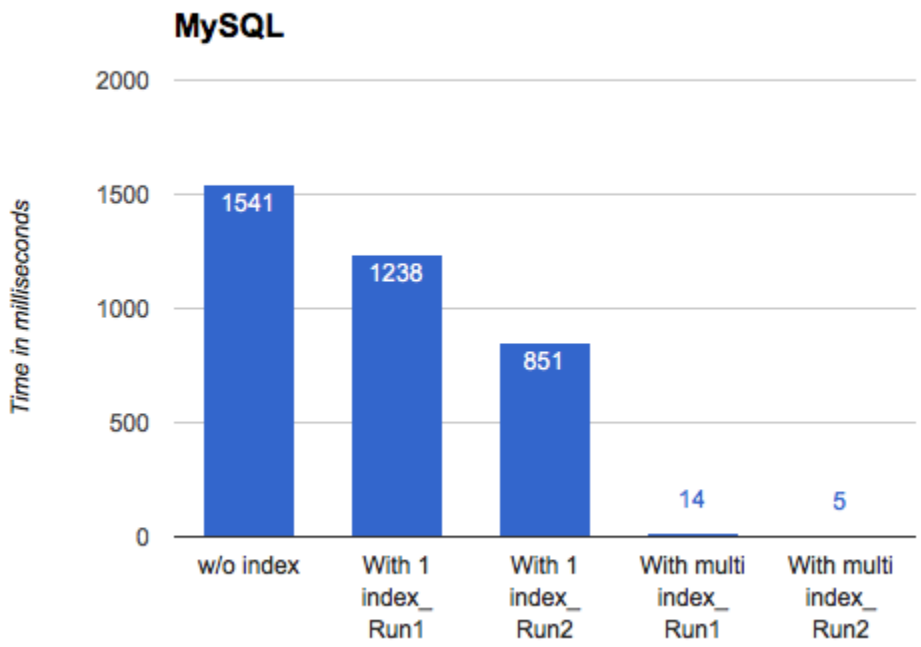
```
public class Main {
    public static void main(String[] args) throws SQLException {

        mysql();
        mongodb();
    }

    public static void mongodb() {

        MongoClient mongoClient = new MongoClient("localhost", 27017);
        MongoDatabase db = mongoClient.getDatabase("database");

        MongoCollection<Document> coll = db.getCollection("Player");

        BasicDBObject query = new BasicDBObject("name", "Jessica Ruiz");
        long start = System.currentTimeMillis();
        FindIterable cursor = coll.find(query);
        long end = System.currentTimeMillis();

         System.out.println(start);
         System.out.println(end);
         System.out.println("Difference");
         System.out.println( end - start);

        mongoClient.close();
    }

    public static void mysql() throws SQLException {
         Connection connection = null; //need to initialize a
java.sql.Connection from JDBC.
          try {

              Class.forName("com.mysql.jdbc.Driver");

              connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/db-
class","root","root");

          } catch (Exception ex) {
```

Sujan Aryal
Julian Fung
Harsha
Anurag Baral

```
            ex.printStackTrace();
        }

            String sql = "SELECT * from Player as p where p.Name = 'Jessica
Ruiz'";

            PreparedStatement ps = connection.prepareStatement(sql);

            long start = System.currentTimeMillis();

            ResultSet rs = ps.executeQuery();

            long end = System.currentTimeMillis();

            ArrayList<String> matchingResult = new ArrayList<String>();
            while (rs.next())
            {
                String name = rs.getString("name");
                matchingResult.add(name);
            }
            for (String single : matchingResult) {
                System.out.println(single);
            }
            System.out.println(start);
            System.out.println(end);
            System.out.println("Difference");
            System.out.println( end - start);
        }
}
```

Sujan Aryal
Julian Fung
Harsha
Anurag Baral

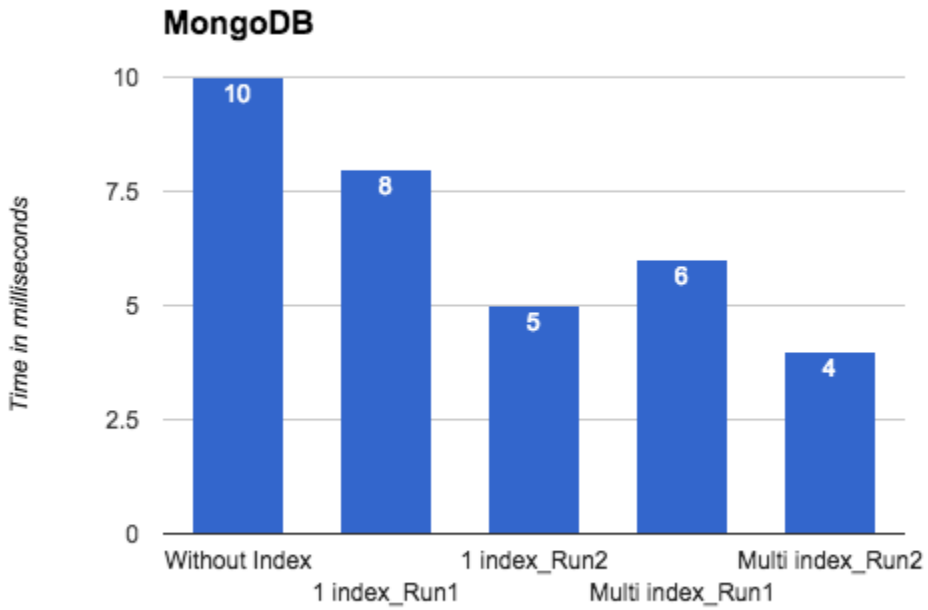
2. Graphs & Analysis



- The above graph shows the relation between time taken for a query to run on table stored in MySQL with and without indexes.
- For the first run there is no index for the particular table on the database. When the query runs, since there are no indexes, SQL Server does a full Table Scan against the table to look through every row to determine if any of the records have the data we are looking for. Without an index, MySQL must read every row sequentially and then read through the entire table to find the relevant rows.The time taken for the query to run is 1541 milliseconds.
- In the second and third run of the query we define an Unique Index. The larger the table, the more time this takes . If the table has an index, MySQL can quickly determine the position to seek to the middle of the data file without having to look at all the data. The time taken was 1238 and 851 milliseconds on two consequent runs of query. We believe the second time was slower because of the cache on the computer.
- At last for fourth and fifth run of the query we defined table with Composite Index. We choose composite index because there were two columns that are frequently used in the query we were running. This further reduced the time to 14 and 5 milliseconds on two consequent runs of same query.

Sujan Aryal
Julian Fung
Harsha
Anurag Baral

Conclusion: Good indexes for database will help search engine to speed up data retrieval. Simply put, an index is a pointer to data in a table which points to the specific data groups we are searching for without searching the whole database.



- The above graph shows the relation between time taken for a query to run on data stored in MongoDB with and without indexes.
- For the first run we did not define any index for the collection. Without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. The time taken for the query to run is 10 milliseconds.
- In the second and third run of the query we define Single Field Index. MongoDB supports the creation of user defined ascending/descending indexes on a single field of a document. Single Field Index is special data structure, that store a small portion of the data in an easy to traverse form. The index stores the value of a specific field, ordered by the value of the field as specified in the index. The time taken was 8 and 5 milliseconds on two consequent runs of query.
- At last for fourth and fifth run of the query we defined table with Compound Index. MongoDB also supports user defined indexes on multiple fields. The order of fields listed in a compound index is always important. For instance, if a compound index consists of { firstname: 1, lastname: -1 }, the index sorts first by

firstname and then, within each firstname value, sorts by lastname. This further reduced the time to 6 and 4 milliseconds on two consequent runs of same query.

Conclusion: Indexes helps to make database faster while doing a search. For searching purpose, without indexes, MongoDB must perform a collection scan, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect and retrieve.

3. Lessons learned:

   For this project our team had to research on various topics outside the class materials which gave us more in-depth knowledge about Database management systems. For the final part of the project we populated the tables in order to use our use cases. We populated our tables with 1.5 million records using an online dummy data generator called mockaroo. The next step that we did was to access the database using JAVA which was for the front end part of the project. Everyone in the team learned how to use java to access SQL and Mongo database and also to run queries on them. Finally we started measuring the delays using different queries from the front end, which then helped us to create our graphs for the comparison. One of the interesting topics that we learned was to create indexes on our tables. Indexes are smart lookup tables that our database used to retrieve the data faster and efficiently.

4. Problems faced and adopted solutions:

   Our first problem we ran into was not being able to generate 1.5 million records from the random data generator site, and we fixed this by duplicating and uploading the same dataset, while creating a new player_id column and auto incrementing it to make it unique as well as a key. Second, we ran into a problem where we could not access the sql and mongo database from Java front-end because we couldn't figure out the syntax used for Java-Mysql and Java-MongoDB connection as well as the driver that we had to download and include into Java project , and we had to google solutions in order to fix the errors. Third, we had a problem when we were trying to figure out how to create multiple indexes, as we weren't completely sure on what we needed to do - between creating multiple indexes or creating different types of indexes. We went through few online documents and find out how to create and use different kinds of indexes, and also how to accurately use more than one index to effectively decrease delay time.

5. Team member roles
   a. Julian
      i. Populating SQL database
      ii. Learned to use Java to access SQL database and run queries
      iii. Learned to define indexes in SQL
      iv. Prepared and consolidated submission documents

   v. Wrote analysis of graphs

b. Sujan
   i. Populating SQL database
   ii. Learned to use Java to access SQL database and run queries
   iii. Learned to use Java to access Mongo database and run queries
   iv. Learned to define indexes in SQL
   v. Learned to define indexes in Mongo
   vi. Wrote front-end to measure delay for SQL and Mongo

c. Anurag
   i. Populated Mongo database
   ii. Learned to use Java to access Mongo database and run queries
   iii. Learned to define indexes in Mongo
   iv. Wrote analysis of graphs
   v. Wrote lessons learned

d. Harsha
   i. Populated Mongo database
   ii. Learned to use Java to access Mongo database and run queries
   iii. Learned to define indexes in Mongo
   iv. Created graphs for each use case
   v. Wrote analysis of graphs