**EXPERIMENT 1**

**Aim:** Case study on professional and commercial database: Summary and Comparison.

**Case Study on:** E-Commerce website (own project)

**Framework:** PHP

**Database**: MySQL

MySQL is a relational database management system based on SQL – Structured Query Language. The application is used for a wide range of purposes, including data warehousing, e-commerce, and logging applications.

Used in association with a scripting language such as PHP or Perl (both offered on our hosting accounts) it is possible to create websites which will interact in real-time with a mySQL database to rapidly display categorised and searchable information to a website user.

MySQL provides many things, which are as follows:

* It provides the Referential Integrity between rows or columns of various tables.
* It allows us to implement database operations on tables, rows, columns, and indexes.
* It defines the database relationship in the form of tables (collection of rows and columns), also known as relations.

**Features:**

1. **Optimized Searching**

I have used mySQL along with PHP for my project. For searching a product by category I have written a SQL query using WHERE clause that just searches into the entire product table that has all the other products as well. Thus, it has to traverse through the entire product table to fetch the specified category products tsking a long time for product information retrieval. making it highly time inefficient. Also, if there a minor spelling mistake then the search fails showing “no results found”.

**Recommendation:**

Currently I have used basic search for the website that just fetches the data when a query is fired. Instead I can use PHP with AJAX because it passes the request to an API that fetches the data and returns back in the JSON format. Firstly, this makes it platform independent as data is being received in JSON format. Also using AJAX we need not re-render the entire page. Only the segment of the page where data is being displayed is re-rendered. Thus, reducing the time and load on the server.

Secondly, can use ElasticSearch in Java for optimized searching. Elasticsearch uses a distributed system on top of Lucene StandardAnalyzer for indexing and automatic type guessing and utilizes a JSON based REST API to refer to Lucene features. Thus, using distributed system it much faster to fetch data than the basic SQL search which I implemented.

For the second problem the spelling mistake one, we can use node.js as it includes libraries that helps in coding **fuzzy search feature** i.e. if there’s a minor spelling mistake then it would direct to the most relevant route. This enhances the searching of our webpage. Also, node has tensorflow.js and other ML libraries which help in optimizing the searching featuring in the website.

1. **Product Details Storing**

MySQL uses RDBMS which in turn has rigidity with respect to the number of attributes for each instance(data record). I had made a table “Products” which had products of all the categories e.g. Necklace, Earing, Rings and so on where each category differs from the other in storing the dimensions. For example, Ring only requires one attribute “Diameter” while Earing requires attributes like “Size, Length and Height” and Category like Bridal Sets required even more. Thus, while storing the dimensions for different categories I had made different attributes into the tables which resulted into many empty cells for records of categories which did not have that attribute.

**Recommendation:**

To overcome this, we can use MongoDB being noSQL(non-relational) is more flexible with the schema and unstructured data. Thus, we can store data as key-value pair.

Moreover, mongoDB also supports storing value multiple datatypes we can just make a dimension key and store all the values in an array.

MySQL:

Product( ProductID , Name, ……. , Diameter, Size, Length, Height)

MongoDB:

Product

{

ProductID: Number,

Name: String,

….

Dimension : [Number],

}

1. **Security**

My project used MySQL which is highly vulnerable to SQL injection attacks. A hacker can skip the authentication process and manually inject SQL statements (or malicious payload) into the database. The database does not recognize the threat and executes the statement as if the application is sending the request.

**Recommendation:**

1. Certain measures that can be taken in order to lower the risk of an attack, such as using parameterized queries instead of a concatenated user input. Parameterized queries require the developers to define all the SQL code first and pass each parameter to the query later. This coding style enables the database to differentiate between code and data, a feature that is not possible with dynamic SQL queries. This is particularly important in WHERE clauses and INSERT or UPDATE statements.
2. Setting up a Firewall. Use a Web Application Firewall (WAF) to filter all traffic between web applications and your database. A firewall protects all web-facing apps, so blocks the SQLi and similar cyberattacks.

**Conclusion:**

In this experiment I have suggested recommendations on 3 features in my E-Commerce project based on PHP & mySQL. Firstly Optimized search where we can use AJAX and REST API for dynamic search instead of re-rendering the entire page, and using fuzzy search in node.js which increases the searching capacity for the user or use ElasticSearch in Java which improves the searching time by using distributed databases. Second feature was product details storing where SQL has schema rigidity while mongoDB is flexible with the data schema. Lastly security, since my project in SQL is vulnerable to SQL injection attacks we can implement parameterized queries or placeholders or use Web Application Firewall (WAF) to block the SQLi attacks.

**EXPERIEMNT 2**

**AIM:** To explore different datasets and download a dataset for performing experiments in the further lab sessions.

**DATASET:**

Name: Wine Quality Dataset

Data Set Characteristics: Multivariate

Number of Instances: 4898

Number of Attributes: 12

**Attribute Information:**

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol

12 - quality (score between 0 and 10)

**OUTPUT:**

**Wine quality dataset**

Graphical user interface

Description automatically generated

**CONCLUSION:** In this experiment, I downloaded Wine Quality Dataset from Kaggle website. The dataset was in the excel (i.e. .csv format) which I imported into the Workbench. During the import there were datatypes conflict and data incompatibility issues between the csv format and the workbench(sql) but those were resolved. The screenshot of the dataset is added above.

**EXPERIEMNT 3**

**AIM**: To implement indexing on our selected dataset

**PERFORMANCE:**

Graphical user interface, text, application

Description automatically generated

**OUTPUT:**

**Created Index: idx\_density**

Graphical user interface, text, application

Description automatically generated

**Created Index: idx\_q\_php**

Graphical user interface, text, application

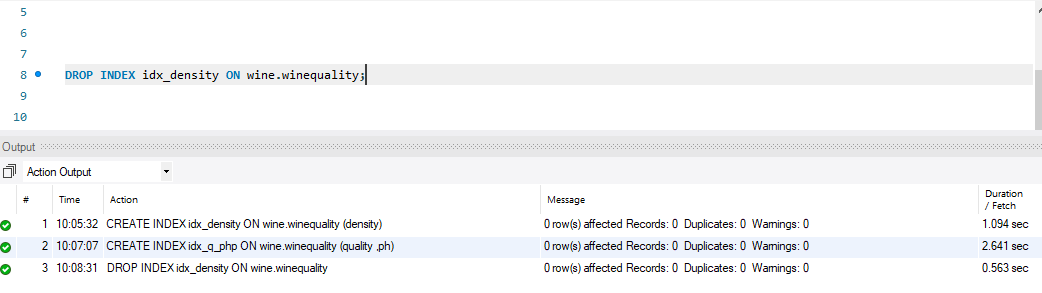
Description automatically generated

**OUTPUT:** Index Created

Graphical user interface, text, application, email

Description automatically generated

**Drop idx\_density**



**OBSERVATION:**

A screenshot of a computer

Description automatically generated with medium confidence

In the console, we can observe that earlier it took a duration of 0.062sec to fetch the data of ph=3 but after adding index it takes 0.015sec. Thus, the overall duration of the process has decreased. The decrease is less in Wine dataset as the number of instances is quite small, there will be significant decrease in time for very large datasets.

**CONCLUSION:** In this experiment, I created 2 indexes ‘idx\_density’ on density attribute and ‘idx\_q\_php’ on attributes quality followed by pH. For both the indexes, duplicate values are allowed and for restricting duplicate values we can add UNIQUE keyword in the query while creating the index. The overall data fetching speeds up, however, updating a table with indexes takes more time than updating a table without because the indexes also need an update. The implementation of the experiment was error free.

**EXPERIEMNT 4**

**AIM**: To implement Fragmentation on our selected dataset.

**PERFORMANCE:**

**1) Horizontal Partitioning:**

Table

Description automatically generated with medium confidenceGraphical user interface, text

Description automatically generated with medium confidence

**OUTPUT:**

Graphical user interface, text, application, email

Description automatically generated

**2) Vertical Partitioning:**

**A) Wines Table:**

Graphical user interface, text

Description automatically generated with medium confidence

**OUTPUT:**

A picture containing table

Description automatically generated

**B) Constituents Table** Graphical user interface, text, application

Description automatically generated

**OUTPUT:**

Graphical user interface, application

Description automatically generated

**CONCLUSION:** In this experiment, I created firstly, Horizontal Partition on the wine quality table by grouping the data items with respect to the pH values. Partition pH0(less than 4), pH1(4-5) and pH2(5-6) are created and the entire partition is stored on a new table ‘partitionpH’ so as to preserve the original wine table. Secondly, I created Vertical Partitioning of the dataset into tables ‘wines’ and ‘Constituents’, where the ID of wines is linked with WineID of Constituents’.

**EXPERIEMNT 5**

**AIM**: Perform basic SQL Queries on the selected dataset.

**PERFORMANCE:**

**SQL QUERIES:**

**a) Original Dataset:**

Text

Description automatically generated with medium confidence

**b) Fragmented Dataset:**

Graphical user interface, text, application

Description automatically generated

**OUTPUT:**

Table

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**CONCLUSION:** In this experiment, I implemented the basic SQL queries on the Wine Quality dataset. Queries were executed on the original ‘winequality’ dataset as well as the fragments ‘wines’ and ‘constituents’. There were no errors while executing the queries.

**EXPERIEMNT 6**

**AIM**: Storing of Dataset Fragments of the selected dataset into different drives.

**PERFORMANCE:**

**Before distributing the fragments:**

Graphical user interface, application, table

Description automatically generated

**After distributing the fragments:**

Graphical user interface, application, table

Description automatically generated

Graphical user interface, application, table, email, Excel

Description automatically generated

**OUTPUT:**

**Before**

Graphical user interface, text, application

Description automatically generated

**After**

Graphical user interface, application

Description automatically generated

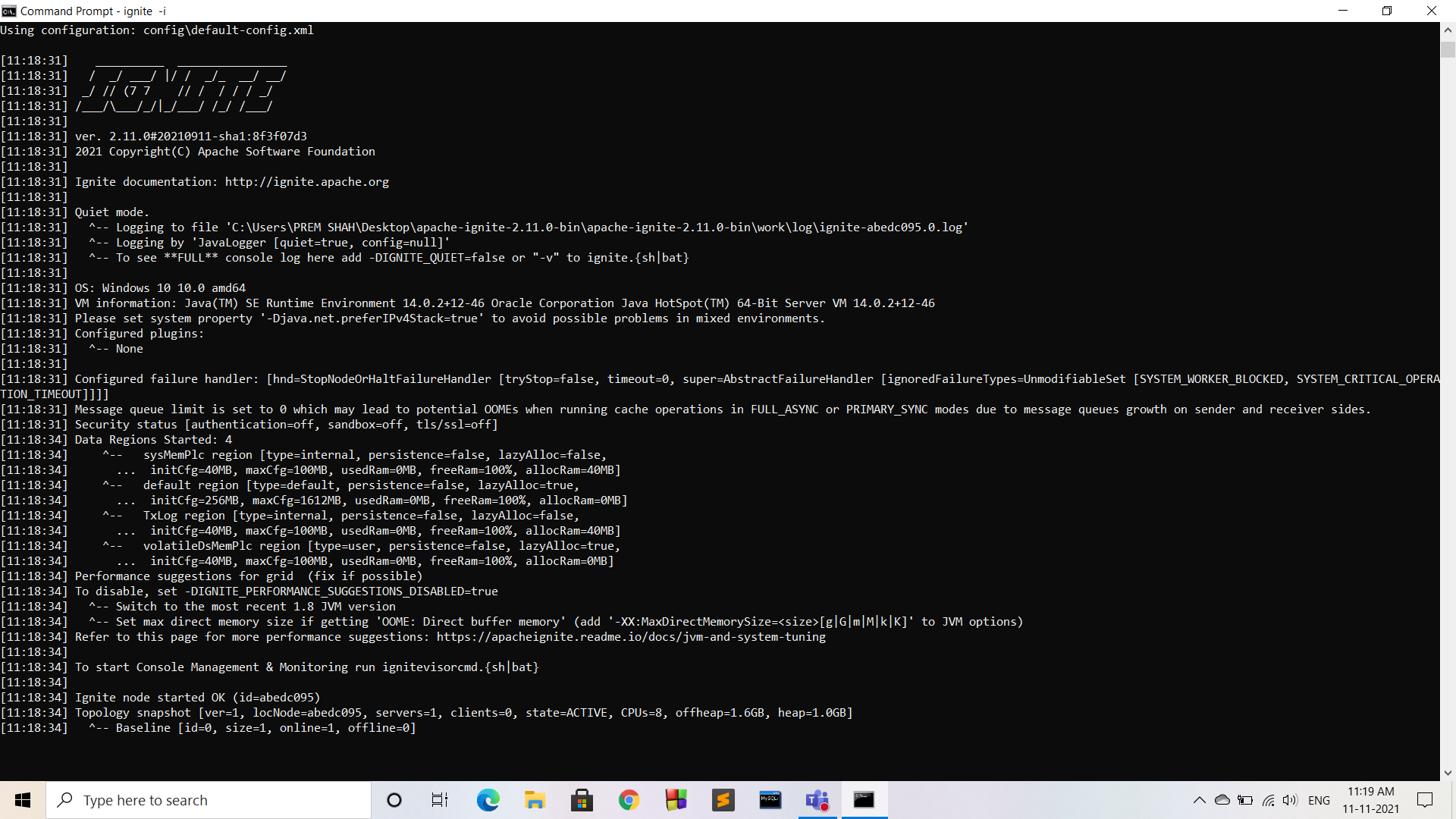
**CONCLUSION:** In this experiment, I stored the datasets fragments into different drives but the MySQL workbench is not recognizing any fragments present on the other drives. The error displayed is 1812 i.e. the table is not found(missing).

**EXPERIEMNT 7**

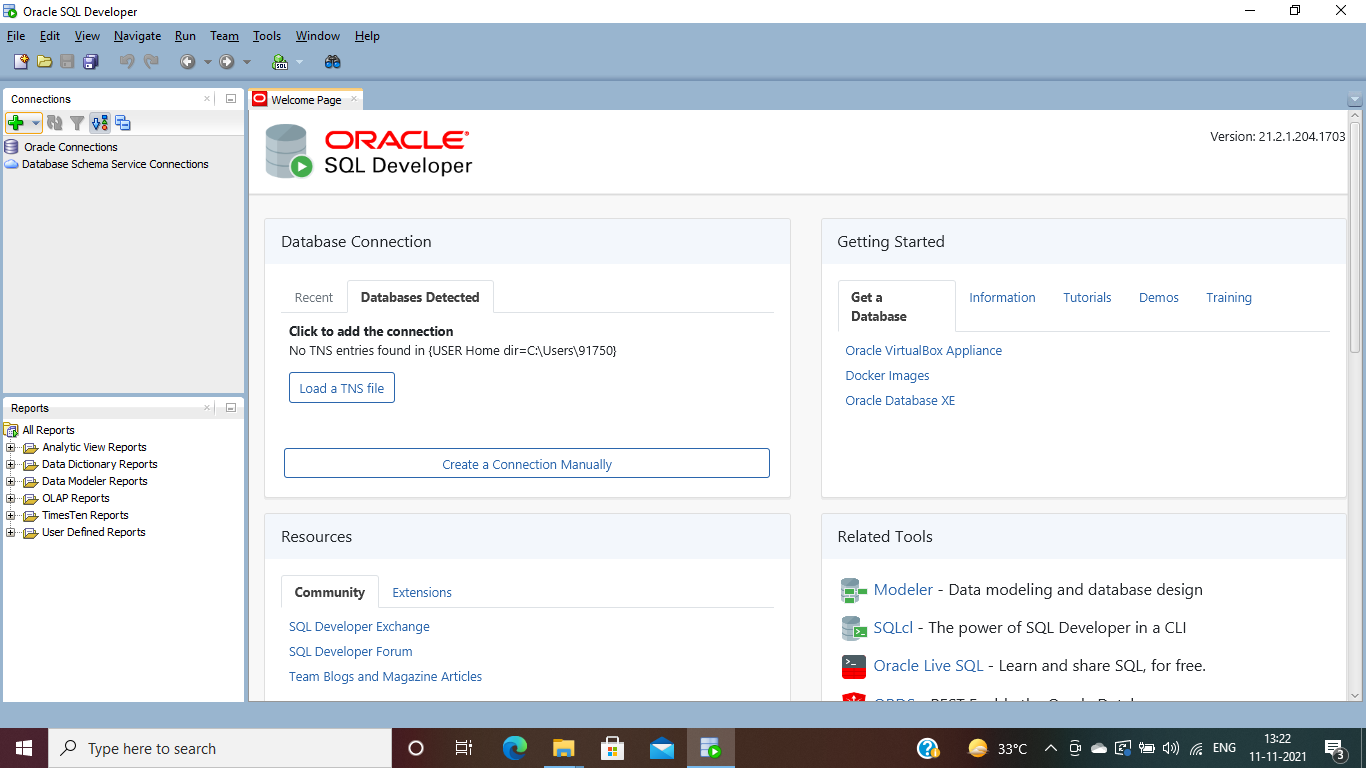
**AIM**: Exploring different Distributed Databases Software and Tools and downloading.

**PERFORMANCE:**

**1) apache-ignite-2.11.0**

****

**2) Oracle sql developer**

****

**Apache Ignite:**

Apache Ignite is a distributed database for high-performance computing with inmemory speed. Apache Ignite's database utilizes RAM as the default storage and processing tier, thus, belonging to the class of in-memory computing platforms. Regardless of the API used, data in Ignite is stored in the form of key- value pairs. The database component scales horizontally, distributing key-value pairs across the cluster in such a way that every node owns a portion of the overall data set. Data is rebalanced automatically whenever a node is added to or removed from the cluster. On top of its distributed foundation, Apache Ignite supports a variety of APIs including JCache-compliant key-value APIs, ANSI-99 SQL with joins, ACID transactions, as well as MapReduce like computations.

Apache Ignite cluster can be deployed on-premise on a commodity hardware, in the cloud (e.g. Microsoft Azure, AWS, Google Compute Engine) or in a containerized and provisioning environments such as Kubernetes, Docker, Apache Mesos, VMWare.

**CONCLUSION:** In this experiment, I explored through different Distributed Databases software and finally downloaded apache-ignite-2.11.0 and Oracle sql developer for the upcoming experiments for performing distributed databases concepts. I will be using Apache Ignite for my Distributed Databases experiments.