TRINITY COLLEGE DUBLIN
COLÁISTE NA TRÍONÓIDE
THE UNIVERSITY OF DUBLIN

Replacing a Keyboard Based Client with a Touch Based Client for Hive

by

Nitin Shambhu

Dissertation

presented to the

Trinity College, University of Dublin

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

September, 2014

Supervisor

Dr. Alexander O'Connor

CNGL Center for Global Intelligent Content

School of Computer Science & Statistics

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Nitin Shambhu
August 26, 2014

Permission to lend and/or copy

Nitin Shambhu
August 26, 2014

Acknowledgements

Firstly, I want to convey my sincere gratitude to my supervisor Dr. Alexander O'Connor for his constant guidance for the research. I want to thank him for allowing me to pursue my dissertation under his supervision. His continued enthusiasm and motivation during the course of my research helped me in writing this thesis.

Besides my supervisor, I want to thank Dr.Neil Peirce for loaning his Android tablet without which the evaluation of this research would not have been possible.

My sincere thanks also go to all the volunteers for readily accepting to participate in the user study.

Last but not the least, I am grateful to my parents and my brother for their constant support throughout my life.

Abstract

Replacing a Keyboard Based Client with a Touch Based Client for Hive

Nitin Shambhu
Master of Science in Computer Science

Supervisor : Dr. Alexander O'Connor

September, 2014

During the times when engaging with social networking sites has become a way of life, the data collected due to the activities of the user is enormously high. Big data grows in large volume at a very high velocity in a variety of formats such as audio, video, image, text etc. At such a high rate of inflow of the data, it is difficult to process a high volume of data. However, Hadoop solves this problem of processing the big data by using MapReduce programming. The MapReduce technique breaks the problem into various sub-problems, which are solved individually. Though this technique is complex, technologies like Hive, Pig etc. create an abstraction layer that accepts queries as input and converts the queries into MapReduce jobs.

This research aims at replacing the keyboard based client with a touch based client by creating an extra layer of abstraction which takes a touch as an input and converts it into a query. This thesis describes the implementation of such a client for Android based devices called *Touch Client*. A study is conducted by performing a few tasks on both command line interface and the touch client. Finally, the evaluation is done by comparing the time taken to perform each task on both interfaces along with the system usability scale test. This touch client makes it easier for the user as it supports a feedback on their touch events of the user, allowing the user to construct the intended query quicker than a command line interface.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

A recent article claims that one in every five persons in the world owns a smartphone while one in every seventeen persons in the world owns a tablet [1]. There are articles that claim millions of people use the internet daily. One such article claims that 2 billion people around the world use the internet [2]. In the times when the internet has become a way of life, many netizens actively engage themselves in social networks like Twitter, Facebook, Linkedin, YouTube etc. Now imagine the amount of content or data generated by those 2 billion in the form of audio or video on YouTube, images or posts on Facebook etc. All these generated data and whatever user does are logged on to the servers. The high increase in the data generated by the online users led to the accumulation of data called “Big Data”.[3] Big data allows enterprises to leverage the large data available instead of subsets and have a competitive advantage over its competitors.

International Data Corporation (IDC) predicts 50 folds of growth in the world's data by 2020. The IDC study also predicts that among the data generated by 2020, 90 % of it is expected to be unstructured data like files, emails, videos etc. Unfortunately, the number of IT professionals is expected to grow only by 1.5 times, which shows that enough technical persons are not going to be available to manage all the data generated. The IDC study predicted that in 2011 alone, 1.8 Zetta bytes of content will be generated. To give a perspective, it is equivalent to every American writing 3 tweets/minute for a period of 26976 years.[3] It is estimated that there are 6 billion mobile subscriptions and 10 billion text messages are sent throughout the world.[4]

According to statistics, Facebook has 955 million monthly active user accounts with 70 languages, uploads of 140 billion photos, 2.7 billion likes and comments posted, 30 billion pieces of content daily and 125 billion friend connections. Every minute, 48 hours of videos are uploaded and every day 4 billion views are performed on YouTube. Google supports services such as monitoring 7.2 billion pages per day and processes 20 Petabytes (10^{15} bytes) of data daily.Twitter has 140 million active users and sees inflow of 1 billion tweets every 72 hours. 571 new websites are created every minute of the day. Within the next decade, the amount of information available will increase by 50 times; however, a number of IT specialists who keep up with all that data will increase by 1.5 times.[5]

Big data is data growing very fast in high volumes that its management becomes difficult with the existing tools. The difficult part of management could be to capture data, search, store, analyze, visualize etc. 3Vs are famously used to describe big data. They are Velocity, Volume and Variety.

- Volume represents very large data size, which can run into tera or petabytes
- Velocity represents the capability of being able to analyze the data streaming into an enterprise, to maximize the value of the time that can be saved if the data were not to be analyzed until the streaming is done
- Variety represents the varieties of data which could be structured or semi-structured or unstructured from various sources. The variety of data could include texts, images, videos, audio, posts, log files etc. [6]

It may become very difficult to manage with the traditional database software tools. Google alone has about 6 million servers across the world. [7] Big data led to an establishment of an industry that came up with supporting architectures such as Mapreduce and Hadoop.

MapReduce, a programming technique, is used for handling large-scale data in a distributed computing environment. When a computationally intensive problem or task needs to be solved or performed, MapReduce is very useful. The namenode (master) of the system takes the input and divides the task/problem into multiple subproblems. Each subproblem is assigned to a datanode (slave) which solves the problem and returns the answer to the node (namenode) that assigned the subproblem. Then the namenode collects the answers from all the datanodes and aggregates them to an output in some way. While this is a complex programming technique, some programming platforms such as Pig, Hbase, Hive etc. have been developed to abstract the Java MapReduce programming into a high level notational system of MapReduce programming. Though this abstraction helps programmers avoid writing MapReduce programs themselves, the programmers still ought to learn languages such as Pig Latin (for Pig) or HiveQL (for Hive).

Among the programming platforms mentioned above, Hive is the tool of choice for this prototype. Hive is conceptualized to enable ad-hoc querying, easy data summarization, and large-data analysis. It offers a query language called HiveQL. This query language is based on SQL and enables users familiar with SQL to easily do summarization, ad-hoc querying and data analysis. [8] Hive query language has a command line interface (CLI) to execute commands. This console based HiveQL interface is not so easy to use for non-experts or a new user, especially in circumstances where query language, the schema, structure of tables, underlying data etc. are unfamiliar.

1.2 Research Question

Is it possible to create a touch based query interface that is *usable* and *easy* for querying databases? If so, how does such a touch interface *perform* in comparison with a conventional interface such as command line interface?

Definition of terms :

Usable: A touch based interface that works and can be used in the place of a command line interface. The only difference is that the former takes touch as an input while the latter takes a query as an input. Though a command line interface supports a query language, it is a keyboard and mouse based. The system usability scale can suggest the usability of the touch based interface. This scale is captured during the evaluation of the developed prototype.

Easy: Given that the users are already familiar with a command line interface, merely creating a touch based interface shall not serve the purpose until it makes querying databases easier by providing a prompt feedback to help guide the user towards the intended query.

Perform: The performance can be defined by different factors. The time taken by a user to perform a query on an interface can be a factor. User rating for the ease of performing a task on each interface can be a second factor that can contribute to the performance of each interface.

1.3 Research Objectives

The project makes an attempt to explore the role of mobile devices in the context of big data.It abstracts the technical difficulties of learning new technologies like Hive, Hadoop etc., which can be an obstacle to analyze big data on a mobile. It focuses on the analysis of the data collected through different means using a mobile touch client. It observes the effect of analyzing the data on a mobile device, as opposed to the conventional way of doing it through the desktop.

This objectives of this research are

- Lower the learning curve
- Make it easy to switch from technologies like MySQL to HiveQL
- Identifying key tasks for the evaluation of the touch interface
- Evaluating the performance of the analysis on touch client (mobile based) against desktop (keyboard based) to get the right assessment of how effective touch based interface is
- Visualizing the analysis results on a mobile device

Effectively, the objective of the research is to move data analysis from being keyboard and mouse centric (desktop based) interface to touch centric (mobile based) interface.

1.4 Research Challenges

In the process of achieving the objectives enlisted above, there are significant challenges to overcome. Addressing these challenges will eventually end up making the prototype very effective. So, it is important to be clear with the challenges to overcome to achieve the objective of this research.

- The UI should display the data in a simple manner that helps user to relate what is displayed with the concept of table in databases. The challenge is the simplicity.
- The learning curve can be lowered by keeping the UI simple. It requires the user to follow similar steps to perform operations on the interface. The UI should be simple yet effective. The challenge is to let users experience the client with least manual assistance. This can be achieved through a feedback system that tells the user what to do next. The challenge is making the UI interactive with a feedback system.

- Identifying operations to be supported by the client to test the usability of the client developed. These operations are designed as tasks for the user to evaluate the client developed. These operations are demonstrated to the user before the evaluation is done. The challenge is to implement some complex operations like Union all and Join in a simpler manner.
- Overcoming all the three aforementioned challenges will automatically lead to moving the data analysis from desktop based interface to touch based client

1.5 Technical Approach

To handle the research objectives, a study of different technologies is done to finalize those to be used in the implementation. The mobile technology is to be decided to implement the touch client.The goal is to develop a mobile based touch client prototype which allows user to interact with databases as stated previously. The setup has a database server as a back-end and the touch client is always connected to the back-end using JDBC. Once the implementation is done, the prototype shall be evaluated against the conventional interfaces such as command line interfaces.

Based on the study of related work, five databases related tasks are picked out for the volunteers to perform. More on this is discussed in Chapter 3. The volunteers are requested to perform the tasks on a desktop (i.e. Command Line Interface) as well as the touch client, which is developed as a part of this project. The volunteers are requested to fill a survey. The survey required the volunteers to provide details like time taken to perform each task and rate the ease of those tasks on a desktop and touch client. All the volunteers shall be assisted to calculate the time taken so that they can focus on their task. The survey also requests volunteers to choose the preference between keyboard based interface and touch based interface after the tasks were executed. The System Usability Scale (SUS) tool shall be used to measure the usability of the touch client.

1.6 Thesis Outline

The rest of this report is structured as follows:

Chapter 2, **State of the art**, gives a brief description of already existing research that are published. It also does a survey of available technologies to implement the research.

Chapter 3, **Design**, discusses the design of the system along with the features to be supported by the prototype.

Chapter 4, **Implementation**, describes the implementation of the prototype system. It also discusses how the prototype should be used.

Chapter 5, **Evaluation**, discusses the methodology along with the results obtained from the evaluation. It also discusses the significance of the results obtained.

Chapter 6, **Conclusions and Further Work**, contains a summary of our results and ideas for future work.

Chapter 2

State of the art

This chapter has three sections. A section to discuss a variety of query tools that exist and surveys each tool to some depth. This section ends with a discussion on options of query tools that are available and ends by determining the tool of choice. Another section that discusses various mobile platforms that exist and ends with a discussion on the options to determine the platform of choice. A final section that discusses the related work.

2.1 Query Tools

The combination of automated data generation and inexpensive storage is contributing to an explosion in data which would be analyzed by the companies. SQL storage warehouses can surely be utilized in this analysis, but there exists applications where the relational data model is very inflexible and collecting data with no immediate need for it will result in unnecessary cost of warehouse. This leads to the emergence of platforms for the analysis of large scale data like Google's MapReduce, Microsoft's Dryad and Hadoop. MapReduce was an initio used by companies based on the internet to analyze webdata. Slowly the demand for Hadoop increased with enterprises wanting to analyze their data.Quite a few query tools have been developed on top of Hadoop. Some are listed below

1. Cheetah
2. JAQL
3. Pig
4. Hive

2.1.1 Cheetah

Cheetah [9] is a data warehouse system developed, by a company called Turn, on top of Hadoop. It takes an advantage of the flexibility and scalability Hadoop provides, to handle both unstructured and structured data and analyze large scale data. Turn is a software services company that provides end-to-end solutions to manage digital advertising. As a company that provides platforms to various global advertising companies, Turn wants to use the data effectively to target the appropriate audience while optimizing the performance and managing the advertising campaigns centrally.

Overview

The Figure 2.1 [9] shows the architecture of Cheetah

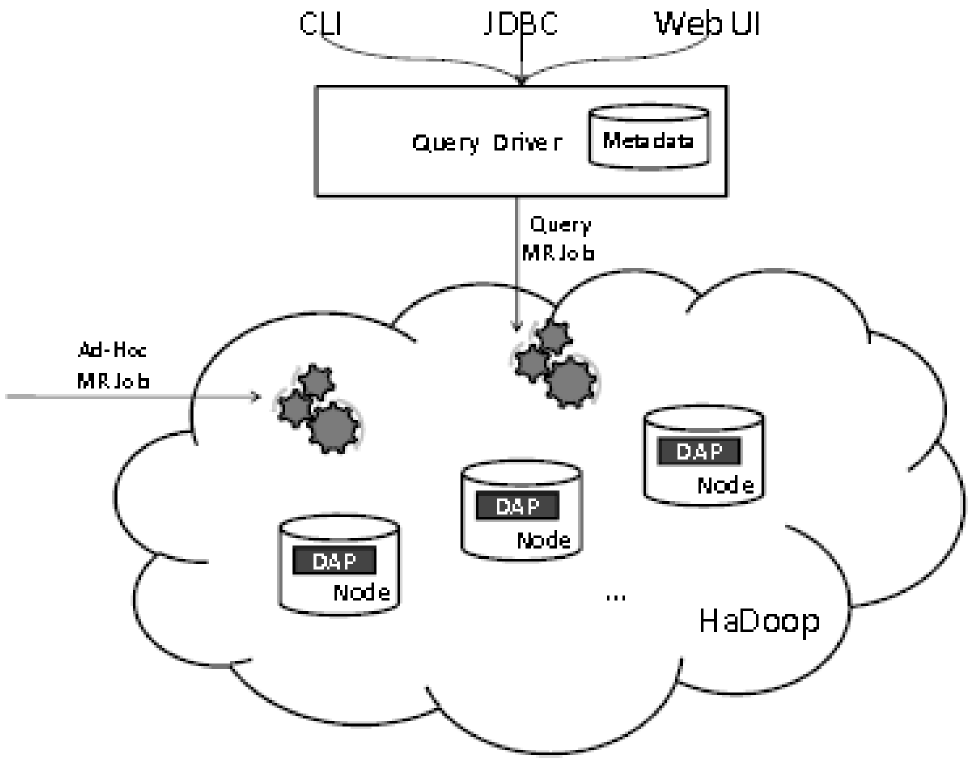


Figure 2.1: Cheetah Architecture

The users can use JDBC (Java code) or Command Line interface (CLI) or web UI to issue queries. A node executes Query Driver to which all the issued queries are sent. The Query driver receives the query and converts it into MapReduce jobs. In the cluster of Hadoop, each Datanode provides a data access primitive (DAP) interface, as shown in the figure above, which is essentially a scanner. The MapReduce job that is obtained by the query Driver from the query uses a scanner to execute SPJ portion of the query. Similar API calls can be made by ad hoc MapReduce jobs for fine grained data access.

StorageFormat

This architecture considers multiple methods for storing tabular data like serialized Java object, text in CSV format, columnar binary array and row-based binary array. A Java class can be serialized and write the member variables to the binary output stream in the implementation while the object can be reconstructed by reading from the binary input stream. The simplest format of storing data is text which is widely used in web access logs. In row-based database systems, row-binary array is widely used where every row is written and read after being deserialized into binary array. Only concerned columns are pulled-out from the binary array in case of the read operation. While all the three methods discussed so far as row-oriented columnar binary array is a hybrid of row-column store. One cell has “n” rows stored.

Query Language

Cheetah provides a query language which is more like SQL queries. Example of a query is given in the research paper [9]

```
SELECT advertiser name, sum(impression), sum(click), sum(action)
FROM Impressions, Clicks, Actions
DATES [2010 01 01, 2010 06 30]
WHERE site in ('cnn.com' ,'foxnews.com')
```

Important features

- *High Performance*: Cheetah exploits a bunch of Hadoop related optimization techniques which range from data compression to multi-query optimization. Each node (a commodity hardware) in a cluster can process the raw data at a speed of 1GB (Gigabytes) per second.
- *Seamless integration*: Cheetah integrated the MapReduce and data warehouse seamlessly. It provides an interface to directly access the raw data by applications. The developers

are expected to take full advantage of the power of MapReduce (scalability and massive parallelism) and data warehouse technologies (efficient and easy data access)

Turn designed Cheetah to allow various custom optimizations and simplifications specific to their online advertisement application. The cheetah is not an open source and hence not accessible.

2.1.2 JAQL

To deal with the needs of customers to analyze the enterprise data, lately IBM released Cognos Consumer Insights (CCI) and InfoSphere BigInsights. Both are Hadoop based and incorporate analysis flows which are more different to the ones observed in the SQL warehouse typically. CCI and BigInsights use JAQL, a declarative scripting language. The main aim of JAQL [10] is to allow developers to do away with the low-level MapReduce programs and work at higher level abstraction to simplify the job of writing the analysis flows. JAQL has a scripting language, a run time component for Hadoop and a compiler as well. All the three are referred as JAQL. Though the design of JAQL is influenced by Hive, DryadLINQ, pig etc., it has focused on some features like flexible data model, reusable & modular scripts, and scalability.

Overview

JAQL reveals each internal physical operator to allow scripting at various levels of abstraction, called as physical transparency, as a function in the language. Though providing low-level control is controversial, it gives JAQL some advantages.

1. Low-level operators provide users an opportunity to pin down an evaluation plan for queries, especially for regularly executed tasks which are well-defined.

2. It allows bottom-up extensibility. Users can add performance enhancements or functionality (like a new join operator) and use them right away in queries. Modifying the compiler or query language is not necessary.

The Figure 2.2 [10] shows the architecture of JAQL

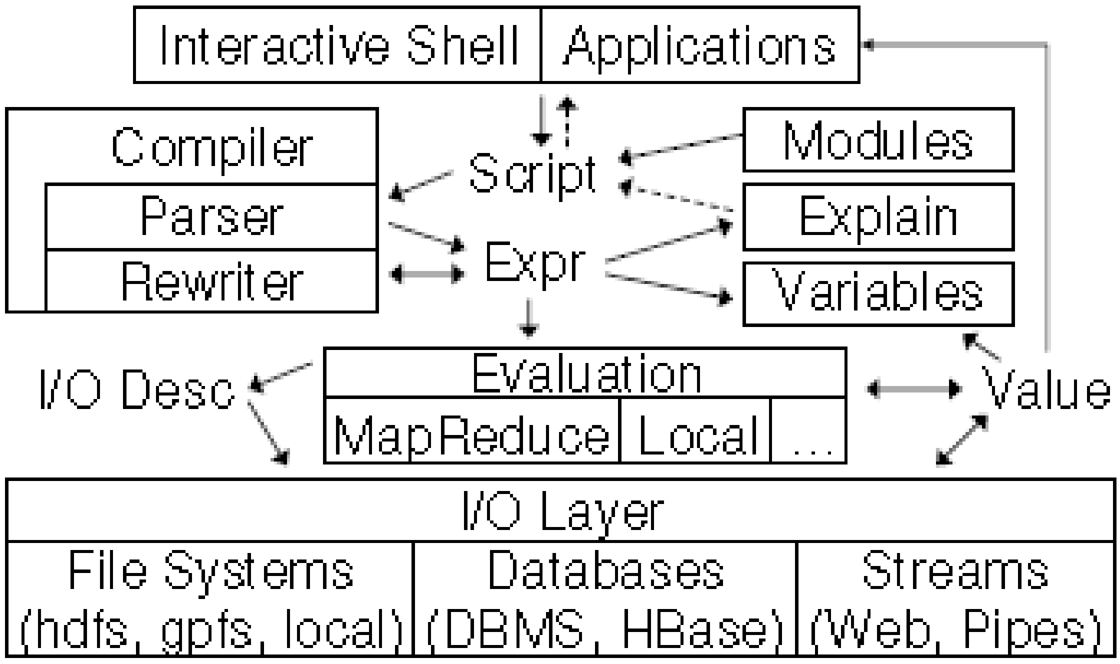


Figure 2.2: JAQL System Architecture

Storage format

The data model of JAQL is JSON based which is a simple format that makes it adaptable to handle semi structured documents in addition to structured documents. JAQL can access data with a partial schema or with no schema. As JSON is designed for the interchange of data there is a low mismatch between user defined functions and JAQL written in various languages.

Data Model

The following information [10] shows the textual representation of JDM of JAQL

```
[
{ uri: "http://www.acme.com/prod/1.1reviews",
content: "Widget 1.1 review by Bob ...",
meta: { author : "Bob",
contentType: "text",
language: "EN" } },
{ uri: "file:///mnt/data/docs/memo.txt",
content: "The first memo of the year ...",
meta: { author: "Alice",
language: "EN" } },
...
]
```

The value of JDM is a record or an array, an atom. The supported atomic types include nulls, numbers, strings and dates. Arrays and records are compound types which can be randomly nested. To add to the detail, arrays are ordered collection of values using which data structures like sets, lists, vector can be modelled. Records are a disordered assemblage of name-value pairs, known as fields and can be used to model maps, dictionaries and structs. Grammar and the textual representation for JDM values are illustrated by the figures below.

Important features

The disadvantage of the JAQL is that it doesn't have any user-defined types as it supports no schema or partial schema. It is not possible for users to define specific types and refer to them by name when they occur multiple times in an application. The downside of JAQL is there is no type safety as it depends on dynamic typing which is generally an expected feature while defining schema. [11]

2.1.3 Pig

Pig [12] is a high-level dataflow system which allows assembling of SQL like data manipulation constructs in a dataflow interleaved with custom Map-Reduce style executables or functions.

The Apache Software Foundation administers both Hadoop and Pig. The language that is used to write the dataflow programs is called "Pig Latin". The compilation of Pig programs generates sequences of Map-Reduce jobs which are executed in Hadoop environment. Pig benefits fault tolerance properties and impressive scalability as it depends on Hadoop as an execution engine. Indexes and column groups like optimized storage structures are currently missing in Pig. Yahoo has widely adopted Pig, despite having scope for improvement with hundreds of users using it to execute thousands of jobs daily.

Overview

The Pig Latin program is given as input in Pig system which is compiled to Map-Reduce jobs and execute these jobs on a Hadoop cluster. The different stages of compilation followed by execution is depicted in figure below 2.3

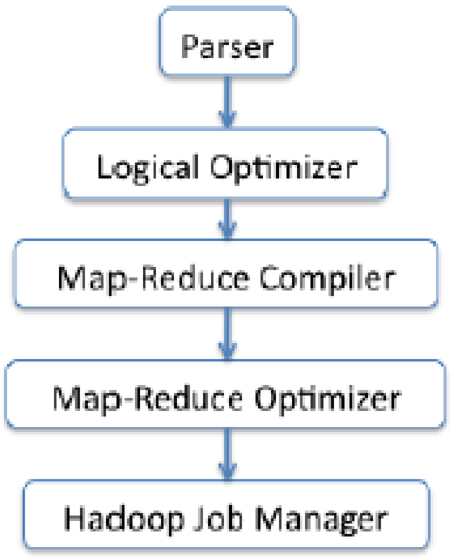


Figure 2.3: Pig Compilation and Execution Stages

Following example [12] gives a flavor of Pig Latin.

```
urls = LOAD ‘dataset’ AS (url, category, pagerank);
groups = GROUP urls BY category;
bigGroups = FILTER groups BY COUNT(urls)>1000000;
result = FOREACH bigGroups GENERATE
group, top10(urls);
STORE result INTO ‘myOutput’;
```

Some features demonstrated by the example above

- It is a dataflow language that chains multiple computation steps together by using variables.
- It also demonstrates high-level transformations like FILTER, GROUP etc.
- It allows the user to define schemas as part of a program
- It allows the use of user-defined functions (not possible with JAQL)

Important features

Pig offers different modes of interaction with the user

- *Interactive mode*: In this mode, the user is presented an interactive shell, called Grunt that directly takes and executes pig commands. Execution of the commands is triggered only when the STORE command is used by the user to request for the output. This is like executing SQL commands in MySQL environment.
- *Batch mode*: This mode lets the user to submit pre-written scripts which contain a sequence of Pig commands ending with a STORE command which triggers execution. More like PL/SQL which is a sequence of SQL commands
- *Embedded mode*: Pig provides Java library that allows the submission of Pig Latin commands through Java program via method invocations. With this option, Pig Latin commands can be dynamically constructed and flow can be controlled dynamically like looping for an undefined number of iterations, which is currently not supported by Pig Latin

2.1.4 Hive

Facebook currently uses Hadoop an open source Map-Reduce implementation to store very large data set and process the data on commodity hardware. Though Hadoop is open source, it is arduous for the users, especially for those who are unfamiliar with MapReduce. Writing Map-Reduce programs are necessary even for simple tasks. The model of MapReduce is a low-level programming, which involves developers writing custom programs which are hard to maintain. MapReduce is not expressive like query languages such as SQL which results in users spending hours in writing programs even for simple analysis.

Prior to 2008, Facebook built data processing infrastructure around a data warehouse built with a commercial RDBMS. While the data is generated at high rate and infrastructure being so deficient, some data processing jobs took beyond a day to be processed and the situation was getting only worse. So, Facebook used Hive and Hadoop technologies to address these requirements.

Hive [13] supports a language HiveQL which is declarative such as MysQL. The queries are converted into MapReduce jobs which are performed on Hadoop cluster. HiveQL allows MapReduce scripts to be plugged in into queries. HiveQL supports tables containing primitive types, collections like maps and arrays etc. It also allows the extension of underlying IO libraries to query for data in custom formats.

Overview
A HiveQL statement can be submitted via a client using JDBC or ODBC interfaces or Command Line Interface (CLI) or web user interface. The driver initially delivers the query to compile. There the query goes through parse phase, type check phase and semantic analysis phase using metadata in Metastore (Metastore DB). The compiler then creates a logical plan which is optimized for MapReduce tasks and these tasks are executed on Hadoop.

Main components of Hive:

- *Metastore*: Stores metadata about partitions, tables, columns, etc. and the system catalog
- *Driver*: Controls the HiveQL statement lifecycle as it goes through Hive
- *Query Compiler*: Compiles HiveQL statements into Map-Reduce tasks

- *HiveServer*: Provides a JDBC/ODBC server and a way of integrating Hive with other applications
- Client components like the web UI, JDBC/ODBC driver and Command Line Interface (CLI)

The following figure 2.4 [13] explains the system architecture of Hive

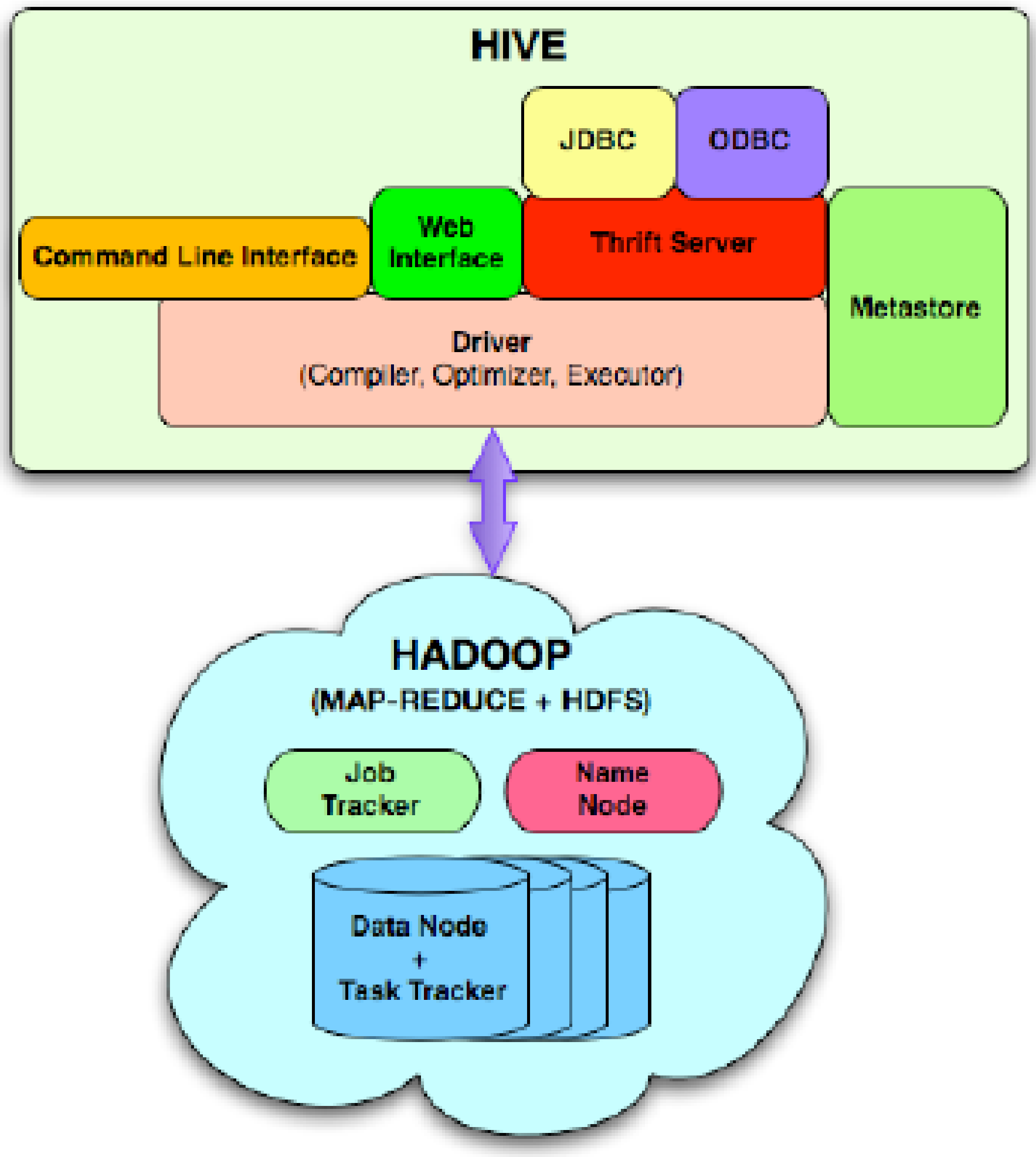


Figure 2.4: Hive System Architecture

Data Model and Type System

Hive stores loads data into tables like traditional databases. Every table has multiple rows where each row has a number of columns. Every column as a type associated with it. The type can either be a complex type or a primitive type.
Primitive types like integers, floating point numbers, String and complex types like

- Structs - struct<file-name: field-type, ... >
- Lists - list<element-type>
- Associative arrays - map<key-type,value-type>

are currently supported.Complex types of arbitrary complexity can also be created. For instance,

```
list<map<string, struct<q1:int, q2:int>>>
```

```
CREATE TABLE tab1(str string, flt float,lst list<map<string,
struct<q1:int, q2:int>>);
```

'.' Operator can be used to access fields within structs.'[ ]' operator can be used to access values in the lists and associative arrays. In the above example, the first element of the list can be accessed with *tab1.lst[0]* and *tab1.lst[0][‘key’]* returns the structure related to 'key' in the associative array. Field q2 can be accessed by *tab1.lst[0][‘key’].p2*.Hive supports serializers and deserializers which are used to serialize and deserialize the tables created in the above manner.

Query Language

HiveQL, the query language of Hive constitutes of a subset of SQL along with some extensions which are useful. It supports conventional SQL features such as

- different type of joins
- group by
- union all
- FROM clause

and many useful features on complex and primitive types which makes HiveQL very easy for anyone familiar with SQL to start Hive command line interface (CLI) and start issuing queries right away.Commands like Show tables and describe are supported.

Important features

There is a limitation that allows only equality predicates in a join predicate and ANSI join syntax is used to perform join such as

```
SELECT t1.x1 as a1, t2.y1 as b1 FROM t1 JOIN t2 ON (t1.x2 = t2.y2);
```

While SQL way is

```
SELECT t1.x1 as a1, t2.y1 as b1 FROM t1, t2 WHERE t1.x2 = t2.y2;
```

INSERT is another limitation. Data partition or inserting into an existing table is not supported by Hive and existing data is overwritten by insert operation.

2.1.5 Discussion

[14] is a study on performance, comparison performed on three High level Query Languages (HLQL) on a 32 node Beowulf cluster based on a range of Hadoop configurations. The key metrics that the study talks about are runtime, scale up and scale out of each language. A direct implementation using Hadoop AP is a performance baseline for each benchmark. Following figure 2.5 [14] shows a high level comparison of query tools discussed thus far.

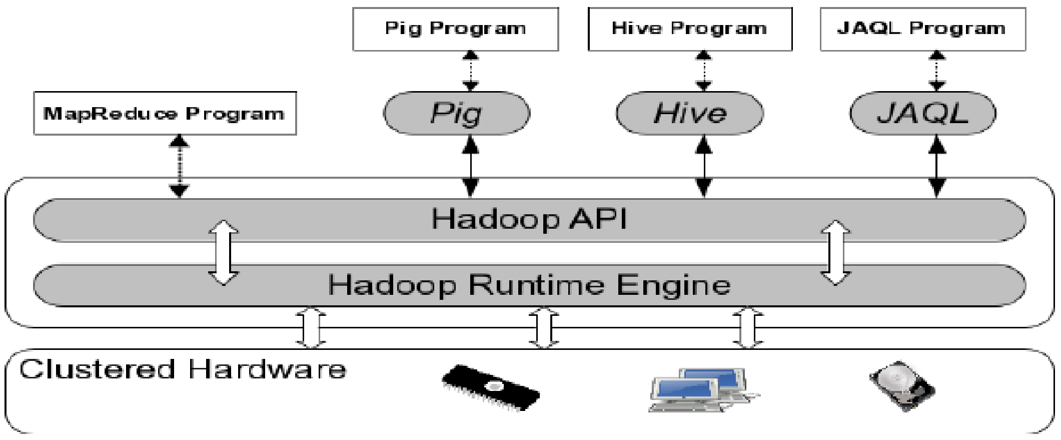


Figure 2.5: Comparison of JAQL, Pig and Hive

A common feature of JAQL, Pig and Hive is that special purpose Java functions can be used to extend them, which means that each language is not restricted to the core functionality and such user defined functions increase the computational power.

The pig has inbuilt optimization for skewed key distribution join which outmatches data handling which is observed in non-adaptive Java Map-Reduce join application. Pig exploits the benefits in raising the number of reduce tasks and manages an unwantedly high level of specified reducer tasks comparatively well.

JAQL is a functional language which is lazy higher-order and is Turing complete while Hive QL and Pig Latin are not. While JAQL doesn't compete with Hive or Java with respect to the runtime, it achieves a speedup for the join on data sets.

Yahoo data engineers believe that Pig is a good solution for data preparation as it provides pipelines and iterative processing. Similarly, Hive is a good solution for data presentation with the ad-hoc querying it supports. [15] concludes that Pig and Hive, combined together, offer a data processing toolkit for Hadoop. [14] found Hive to be the best performer for the speedup, scale-out and scale-up experiments for all the three benchmarks and during the experiments, it is also found that Hive is slower than Java by a fraction with respect to runtime. Hive takes the benefit of tuning the reducers parameter for the join operation on datasets. The computational power of Hive and Pig are shown to be equivalent to SQL. HiveQL is chosen to be the choice of query tools for its very high similarity with SQL.

2.2 Mobile platforms for Mobile Applications

Mobile devices slowly but surely acquired the acceptance as a multimedia platform. A variety of technologies like XHTML/CSS, Python, Mobile Ajax, Open C, JavaScript, Java are available to application developers to implement mobile applications which are highly functional. Multimedia Messaging, Video, audio and Flash can be used by content developers to generate compelling and rich mobile content. Though market generally drives the popular preference for development platforms, it also depends on the requirements of applications and the characteristics of the platforms.

A study [16] compares four application platforms for mobile which are

- .NET compact framework (CF)
- Flash Lite
- Java Mobile Edition (JME)
- Android

All these application platforms have a large development base and also have large deployment base.

2.2.1 Adobe Flash Lite

Flash Lite famous as game programming platform and as multimedia is a proprietary technology. Adobe specifically made it to let vendors quickly deploy interactive interfaces and rich content to mobile devices. A vector based SWF graphics and animation format are used to store the GUI description and the contents of a Flash Lite application. ActionScript is used to implement the presentation logic of an application. Flash Lite is widely adopted worldwide by developers, operators and Original equipment manufacturer (OEM).

Flash Lite is a reasonable option for graphic intensive applications on phones and PDAs. The ease in the changeover from desktop to mobile applications by the developers skilled in flash lead to increase in the adoption of flash by the industry. Features like easy to learn, rapid development and easy to migrate are the benefits of Flash Lite.

2.2.2 .NET compact framework (CF)

.Net CF is a subset of the full .NET platform of Microsoft. It is designed for the application development for Windows platform. CLR (Common Language Runtime) engine is preloaded in the memory of the device by .NET CF to facilitate the deployment of mobile applications. The CLR provides interoperability with the operating system of the device thereby allowing for the integration of components native to the operating system into mobile applications.

.NET CF in principle is similar to Java Virtual Machine (JVM). .NET developers write managed code rather than writing native code for the operating system underneath and is targeted at a managed execution environment. .NET platform was originally designed and developed by Microsoft to support multiple operating systems and languages to reach out to the large developer base. Visual studio, a .NET CF development tool, currently supports only VB.NET and C# .NET CF restricts its support to only Windows platform, which constitutes only a small part of mobile products today.
With reference to providing rich libraries, managed runtime environment, reusable components (like network connectivity, user interface component, XML web services, data management and so on) which can ease the changeover for desktop developers to the mobile domain.

2.2.3 Java Mobile Edition (JME)

JME is a subset of the Java platform. It provides a certifiable group of Java APIs to develop software for devices like set-top boxes, cell phones, and PDAs which are resource constrained. KVM (Kernel-based virtual machine) is a software that runs the applications of JME and allows access to underlying functionality of the device. JME supports cross-platform development. JME enabled devices support following specifications.
CLDC (Connected Limited Device Configuration) which is a framework for JME applications targeting devices which are resource-constrained. CLDC holds a subset of Java class libraries that are needed for application development on mobiles. MIDP (Mobile information device profile) sits on top of CLDC and is a part of JME platform. It is a specification to use Java on mobile phones like embedded devices.

Java mobile edition is a mobile software platform with a domination with respect to developer base and installation. ”Write once, run everywhere” axiom of Java language doesn’t apply to JME. Different application versions should be provided by the developers to address different Java specification Requests (JSR) sets and variation in implementations over a wide range of device capabilities.

2.2.4 Android

Android is an operating system for mobile devices with an open source license and is based on the Linux kernel. The software stack of Android includes middleware and key applications along with the operating system. The applications on Android are developed in Java and compiled into a Dalvik executable (DEX) format, a custom bytecode. Every application has a process and an instance of DVM (Dalvik Virtual Machine) assigned for its execution. The JAR files and standard classes are converted into DEX files during compile time, which are run on the DVM. DEX files, in comparison to class files, are more efficient and compact. APIs that core components use and frameworks are exposed to the developers. Simplifying component reuse is one of the main aims of software architecture of Android.

Android Software Development Kit (SDK) helps in writing applications with rich functionality. It handles accelerometers, GPS, 3D Graphics and touch screens like the iPhone. Android supports a comparatively large subset of JSE 5.0 (Java Standard Edition), thereby reducing the cost of migration from desktop applications. Third party libraries are supported as well. Famous integrated development environments (IDEs) for Java such as Eclipse and NetBeans provide support for Android application development.

2.2.5 Discussion

This study compares the all the four platforms with respect to multiple factors like functionality, portability, Development speed and performance. Though all the platforms responded differently for different factors, it is important to decide on a platform of choice for the development of the prototype.

As stated already, .NET CF restricts its support to only Windows platform, which constitutes only a small part of mobile products today. This implies that the developer base of .Net CF will remain only till Windows devices are in the picture. Due to high difficulty in porting the applications to popular mobile operating systems, this platform is not considered as a choice.

Adobe developed Flash lite more with an intent to support multimedia than developing a set of powerful APIs to support application development with rich functionality. Though efforts have been made to establish Flash lite as a gaming platform, it doesn't support for APIs or classes required for the game development. For instance, BitmapData object is not supported as of Flash lite 3.0. The study also shows that Flash lite demonstrates lower performance while it consumes more memory in comparison with Java based platforms. For portability and performance reasons, this platform isn't considered.

This study reveals that that development of Java based applications helps bring down the learning curve. The experience of application development on the desktop is always an add-on and makes it easy in moving the applications from desktop to mobile, thereby shortening the development time. Thus, Java Mobile Edition (JME) and Android are the options left. Due to the time constraints and for the mere experience of having worked with Android in the past, Android is chosen as the platform for development.

2.3 Related Work

2.3.1 dbTouch in Action - Database Kernels for Touch-based Data Exploration

DBTouch [17] is a research proposal to change the database exploration from console based to touch based interactive and intuitive tool, which allows the users to explore data without being aware of priori. This idea aims at making the exploration of data more interactive by using a touch based interfaces to manipulate data while searching for patterns. This research proposes to redesign the database kernel to an architecture that makes the data exploration touch-based.

In the context of this research, the data is represented in different shapes that allows the user to interact by touching the shapes. For example, each column of a table can be displayed in a shape. Each shape detects the touch gesture like the speed of the gesture, its direction and determines what action has to follow. Each object (shape) gives a high-level schema to the let the user have an idea of the data that can be searched by gestures. The efficiency of dbTouch engine lies in combining the processing of low level queries with the visualization techniques.

Discussion

Advantages : The advantage with this idea is that the database systems are represented in an adaptive and intuitive way that allows the user to scroll partial data processed by sliding finger over an object. Zoom-in gesture can be used to initiate the processing of more data while zoom-out can be used to visit the older data. The user can also rotate the table to modify the physical design of the table from row based to column based.

Challenges : This scope of allowing users to continually touch the shape for inputs give rise to challenges such as data access patterns when the user inputs, touch patterns such as slide finger or use two fingers to zoom-in etc., to interpret query equivalent to touch gestures. To quote a few, the user while continuously sliding finger, the sliding may be faster every time or get slower or may pause for some duration doing nothing, to translate gestures to appropriate algorithms of database, optimization issues that will help the dbtouch to anticipate the gesture to access, fetch and compute data beforehand to make the feel of user experience more interactive.

Every user touch on the shapes displayed is treated as a request to execute an operations(s) on the data. When the user gives inputs (through touch) data is analyzed but only some parts of it at a time while it refines the answers along with the continuous user input. The back-end of the system is still a standard database system which is supposed to consume to process the data to give back the results. This design is a bottleneck in case of big data. This is made interactive by processing partial data at a time and run the results on the shape while in the background more data is processed.

2.3.2 GestureQuery - A Multitouch Database Query Interface

MySQL interfaces on the desktops make it tough for the users especially when the user is unfamiliar with the underlying scheme or structure of the databases. This increases the learning that non-experts need to go through to familiarize themselves with the query language. The desktop consoles of MySQL interfaces are improved over the years to make it more user friendly to the end users through example queries or auto-complete query features or basic queries that are made readily available for the end users. But these features are mouse and/or Keyboard driven and do not assist the users with prompt feedback about the query the user intends to execute. So, an attempt had been made to make the keyboard driven interfaces more gestures based.

GestureQuery [18] is one such attempt that allows users to build and perform queries over relational databases through gestures which can also be multi-touch. The user is displayed with three parts mainly a header, a tray and a workspace. When the user touches the header, a list of available databases is displayed. The user can then select one database who tables are listed in the tray. Any table can be dragged on to the workspace. The user is allowed the drag multiple instances of the same table on to the workspace. In this way, the user can drag required tables on to workspace to perform queries like join, union, select, sort by etc. When the user drags a table on to the workspace and scrolls to the right, all the rows of a typical table in MySQL databases are displayed. When the user does a long press on a single cell in the table displayed, the gesture map engine takes it a conditional select statement like

```
Select * from table employee where COLUMN_NAME = CELL_VALUE;
```

where COLUMN_NAME is the name of the row against which long press event occurred and CELL_VALUE is the value of the cell which is long pressed. The user is allowed to rearrange the attributes of the table. When the user swipes an attribute from right to left, the gesture engine interprets as a sort by the attribute in question. The user can also drag multiple tables on to the workspace to perform operations like join and union. JOIN on two tables can be performed by dragging them towards one another. Similar UNION can be performed by dragging two tables one below the other like a stack.

Discussion

Advantages : This approach provides a different way to build queries, visualize the schema but unlike the traditional MySQL console it promises to provide immediate feedback. The idea proves that the keyboard driven SQL interfaces can be replaced by a multi-touch gesture based query language.

Challenges: On the flip side, given the fact that research is published very recently, providing gesture based interface to replace traditional SQL interface might not be beneficial. For non-experts, learning curve is the almost the same with a slight difference. In traditional approach query language is learnt while gesture based query language is learnt in the case of gesture interface. In the times when Big Data technologies are taking the limelight and when Big data is going to be the future, the gesture based interface kind of idea is needed more to deal with big data rather than MySQL which is very well familiar to many programmers across the world.

2.4 Summary

This chapter comprises three sections. The first section reviews the existing query tools for a Hadoop system. It discusses their features in some depth. This section ends with a comparison of all the query tools and justifies the selection of HiveQL as the back-end. The second section reviews the existing mobile platforms that are available for the development of touch based applications. This section also ends with a comparison of all the platforms and justifies the selection of

Android as a platform to develop the prototype. The third section surveys the related work in the field of databases. It discusses the features, advantages and challenges of each idea separately.

Chapter 3

Design

This chapter provides the design decisions for this research. The chapter describes the development process that was followed, features to be supported and the overall flow of the actions. Prior to that, it discusses how the related work reviewed in chapter 2 is used as basis for the features to be developed.

3.1 Requirements

With reference to Chapter 2, Android is chosen to be the mobile platform to implement the touch client and the Hive is chosen for the back-end. As observed already in related work section, Gesture Query [18] proves the possibility of implementing a complete SQL through gestures. It also proves that gesture based interface can potentially be utilized to querying databases devoid of the keyboard. The functional requirements of the system are designed based on Gesture Query. The following is the list of functional requirements for the touch based client.

- The user should be able to perform FETCH operation to fetch all the records of a table
- The user should be able to perform FILTER operation of the records displayed
- The user should be able to perform the SORT operation to sort all the records displayed
- The user should be able to perform UNION ALL operation to combine the record sets of two select statements.
- The user should be able to perform a JOIN operation to join two tables based on a common column.

3.2 Design Structure

One of the objectives of the research is to lower the learning curve. This can be achieved only by keeping the implementation as simple as possible. Though the features were based on Gesture Query, the UI design of the touch client is totally different from it. Instead of gestures, the functional requirements listed above are provided as UI subcomponents in the form of a menu. The menu shows the list of features the current screen supports. So, the user can directly choose any operation to perform without having to remember the gesture that maps to the operation. The system is designed in a way that every user action follows the same set of steps to perform a query. Every action (database query) follows the following 6 steps

- The User touches a UI component
- The UI component converts the touch to a query, initiates a background task and executes it
- The background task, then requests the database server for the result of the query
- The database server returns the result to the background task
- The background task feeds the UI component with the data.
- The UI component displays the result of the query to the user

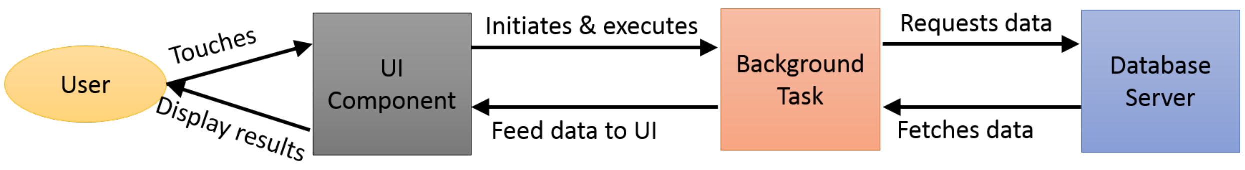


Figure 3.1: Sequence of steps for a query

3.3 Sample Database Structure

A sample database is created and populated for this research purpose. The structure of tables considered for the *sample* database is as follows

albumid	albumname	artistid	numofsongs

Table 3.1: Albums

artistid	artistname

Table 3.2: Artists

songid	songname	artistid	albumid

Table 3.3: Songs

3.4 Conclusion

Gesture Query supports gestures such as single swipe, a swipe with two fingers, pinch in, pinch out etc.. These gestures are interpreted as queries and the displayed results are updated accordingly. On the contrary, the touch client is designed to do the same job without gestures. It doesn't need any gestures as the required features are presented to the user in the form of a menu. Two factors that contributed to this decision are

- *Simplicity*
  When many operations are possible, it is difficult to remember the mapping of gestures and operations. It can easily confuse the users, thus requiring more time to get used to the system. The gestures may not be accurate as well. Due to this reason, the implementation of Gesture Query is a bit complex to follow.

- *Intuitive UI*
  Consider FILTER operation for instance. For a user to filter the records based on a column value in the case of Gesture Query, a user has to *long press* the table cell. However, it doesn't explain what the selection of multiple table cells means. But the touch client handles it by providing a menu option "**Filter**". When the user clicks on it, a filter mode will be activated and the user is allowed to select as many filter options as possible. The options such as "AND " and "OR" are displayed to the user to allow the user to select multiple filter options based on the necessity. Therefore, the user can filter the records using a clause <Filter option1> **AND** <Filter option2> to consider both filter options or use <Filter option1> **OR** <Filter option2> to consider either of the options.

3.5 Summary

The objective of this chapter is to provide the decisions that are made to design the system that allows the research question to be addressed. Firstly, with reference to the related work, the functional requirements are gathered. Secondly, a high level design of the front-end is discussed along with the reasons to choose a way of implementing it differently from Gesture Query. Thirdly, the conclusion of the section clearly describes the factors that influenced to choose touch based system totally ignoring the gestures.

Chapter 4

Implementation

This chapter discusses the overview of the system architecture along with the UI that accommodates all the necessary features. It also discusses the problems faced during the implementation and the changes that are made to the design because of those problems. It then discusses how well the intended design is implemented and the impact of the changes on the user experience. Finally, the chapter concludes with a brief analysis of how good the implementation is.

4.1 System Overview

The prototype was implemented on an Android device. The setup has a database server as a back-end. The touch client is connected to the back-end using JDBC. The architecture consists of three main components

- UI Layer
- Task Layer
- Database Server

An instance of MySQL server runs on the database server. The touch client is always connected to this server, which is the back-end.

UI layer and Task layer are a part of the touch client. The UI layer handles all the touch events through activity. An activity is an android application component which is a screen that lets the user interact (through touch events) with a device/an application to perform an action. The UI layer consists of multiple activities which are responsible for multiple queries. Each activity takes the touch event and converts the touch event to a database query. It then creates and executes the query with the help of an asynchronous task. An asynchronous task is something that initiates multiple worker threads to perform an action in the background. The asynchronous task, then creates a JDBC connection, queries and fetches the data from the database server. The activity then receives the data from the asynchronous task, parses it and displays it to the user.

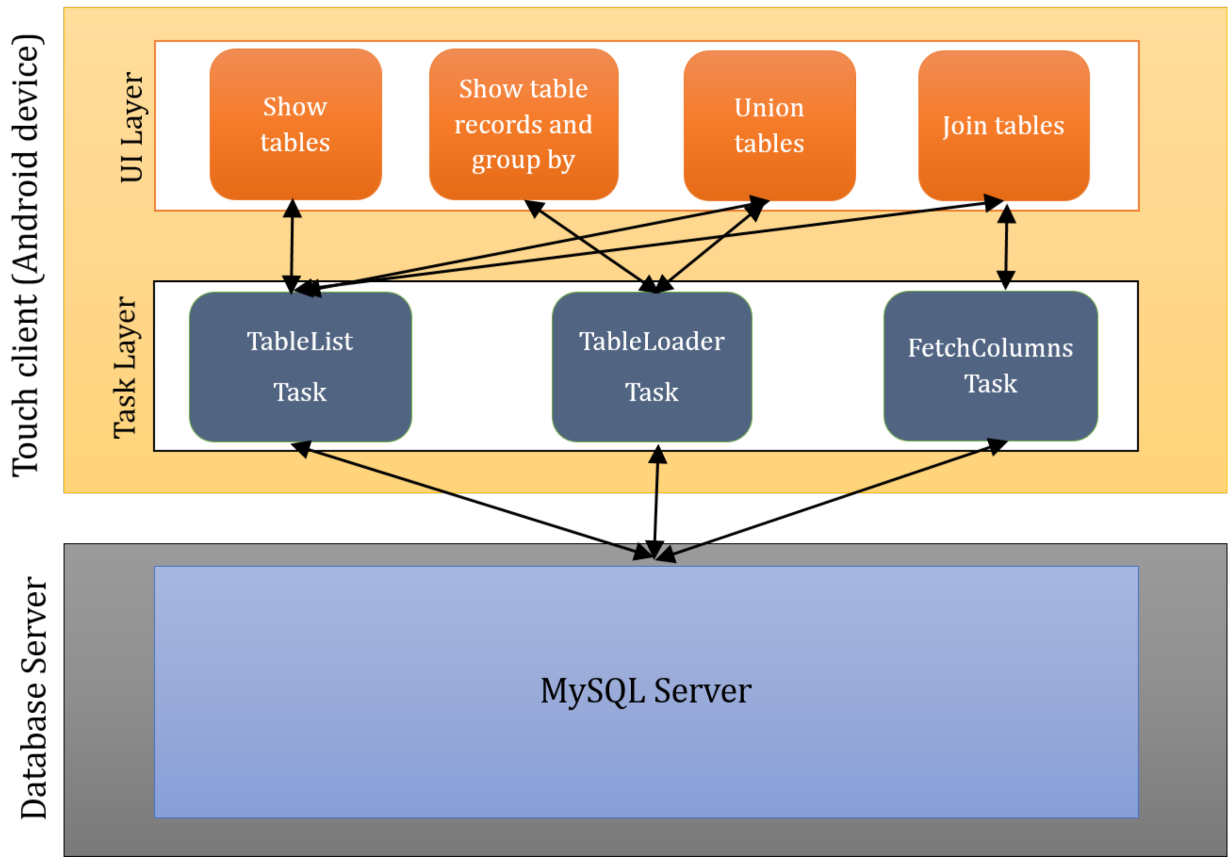


Figure 4.1: Functional Architecture

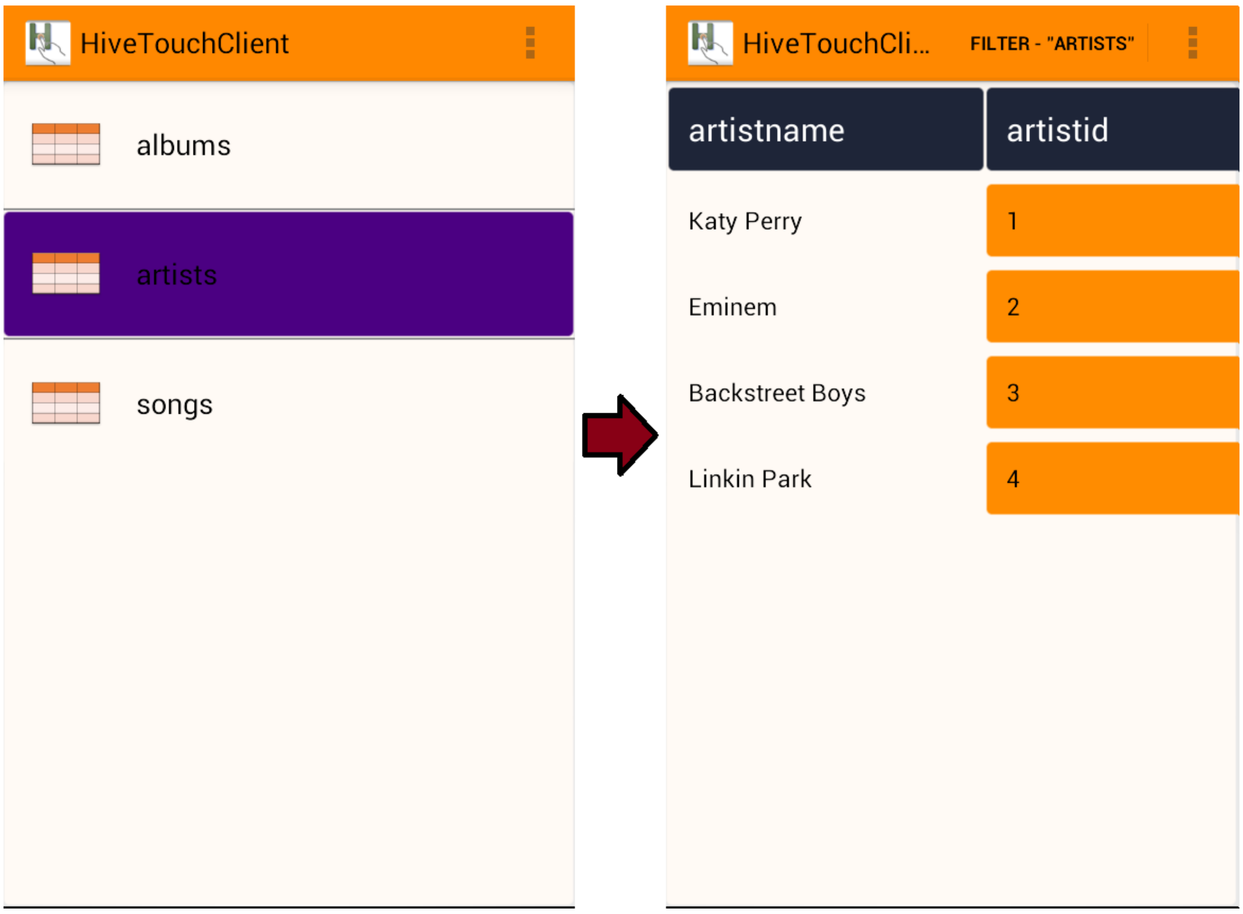
4.2 Support for different screen sizes

A wide variety of devices run the Android operating system. These devices offer different screen sizes. Android operating system offers a coherent environment for developing across devices. It also handles the task of adjusting the user interface of an application to the screen of the device on which the application is displayed. With the help of Android component called “Fragment”, the prototype is implemented according to the screen size. This can be achieved by using fragment inside the Android UI component called “Activity”. For better understanding, consider two screens

Screen1 and Screen2. These two screens can be handled differently by devices with different screen sizes.

Handsets

On a handset, upon selecting an item on Screen1, Screen1 is replaced by Screen2, which is populated with the details of the item selected on Screen1. In this case, Screen1 is a fragment in ActivityA and Screen2 is a fragment in ActivityB. Please observe the following figure.



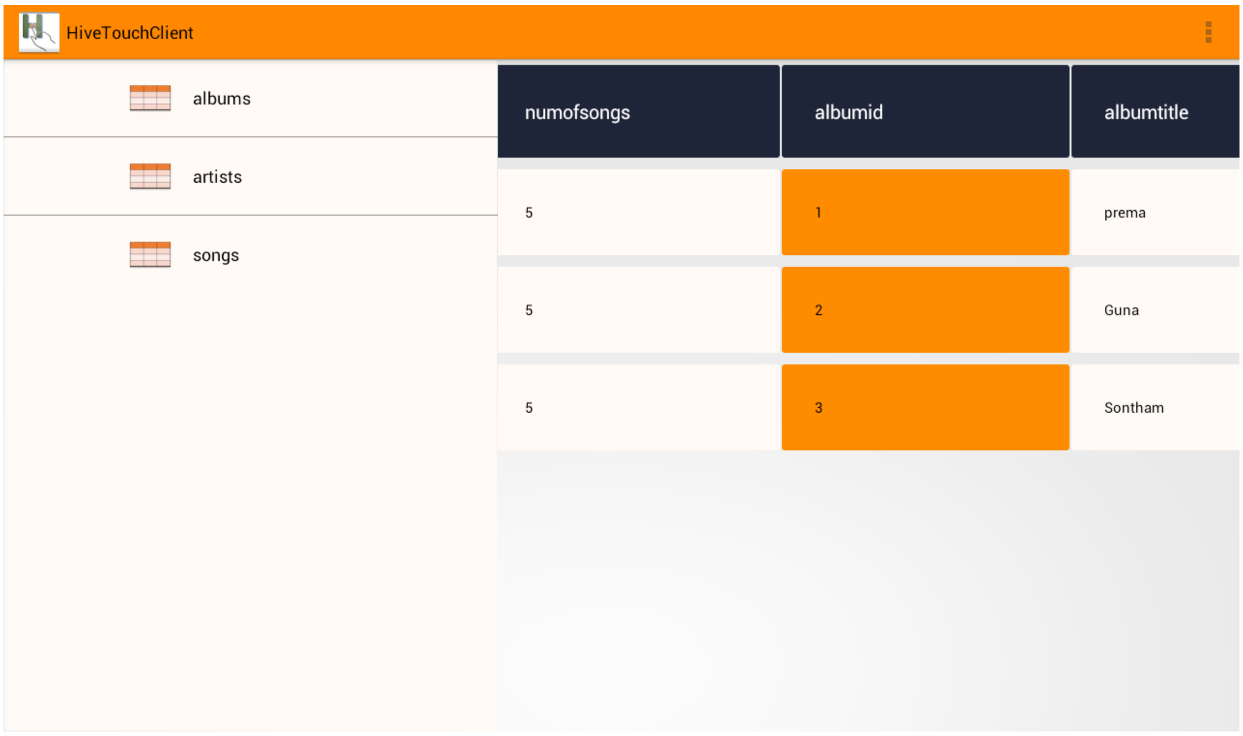
artistname	artistid
Katy Perry	1
Eminem	2
Backstreet Boys	3
Linkin Park	4

Figure 4.2: Upon selecting an item on one screen, the user is directed to a different screen

The user selects an item on the list of the tables and views the records of the table. As one can observe from the Figure 4.2, the image on the left shows the selection of an item on the list by a user while the image on the right shows the user being directed to a different screen where the records of the table are published. The image on the left and right are two different fragments in two different activities.

Tablets

The same scenario is handled differently in the tablet. Without having to implement everything specific to tablet, the same two fragments can be arranged adjacent to each other on the same activity for tablets there by making the implementation very easy. As the tablet has a larger screen size, Screen1 and Screen2 appear adjacent to each other. Upon selecting an item on Screen1, the Screen2 gets updated. In this case, Screen1 and Screen2 are both fragments in same activity which is not the case with handset. In handset, Screen1 fragment is one activity and Screen2 fragment is in a different activity.



numofsongs	albumid	albumtitle
5	1	prema
5	2	Guna
5	3	Sontham

Figure 4.3: List of the tables (to the left) and screen that populates the records (to the right) are on the same screen

Please observe the Figure 4.3. The fragment that shows the list of the tables (section on the left of the figure) and the fragment that is populated with records (section on the right of the figure) are on the same screen. In this way, taking advantage of this support, the development supports both tablets and handsets.

4.3 Hardwares

Following are the details of the tablets used for the development

- *Model name* - Acer Iconia Tab
- *Model number* - A501
- *Android Version* - 4.0.3
- *Internal Memory* - 32GB
- *CPU* - Dual-core 1 GHz
- *Screen size* - 10.1 inches

Following are the details of the handset used for the development

- *Model name* - Motorola Razr i
- *Model number* - XT890
- *Android Version* - 4.1.2
- *Internal Memory* - 8GB
- *CPU* - 2 GHz Intel Atom
- *Screen size* - 4.3 inches

The prototype was tested on Nexus 5 as well and found to be working fine.

4.4 Touch Query Client

4.4.1 Features

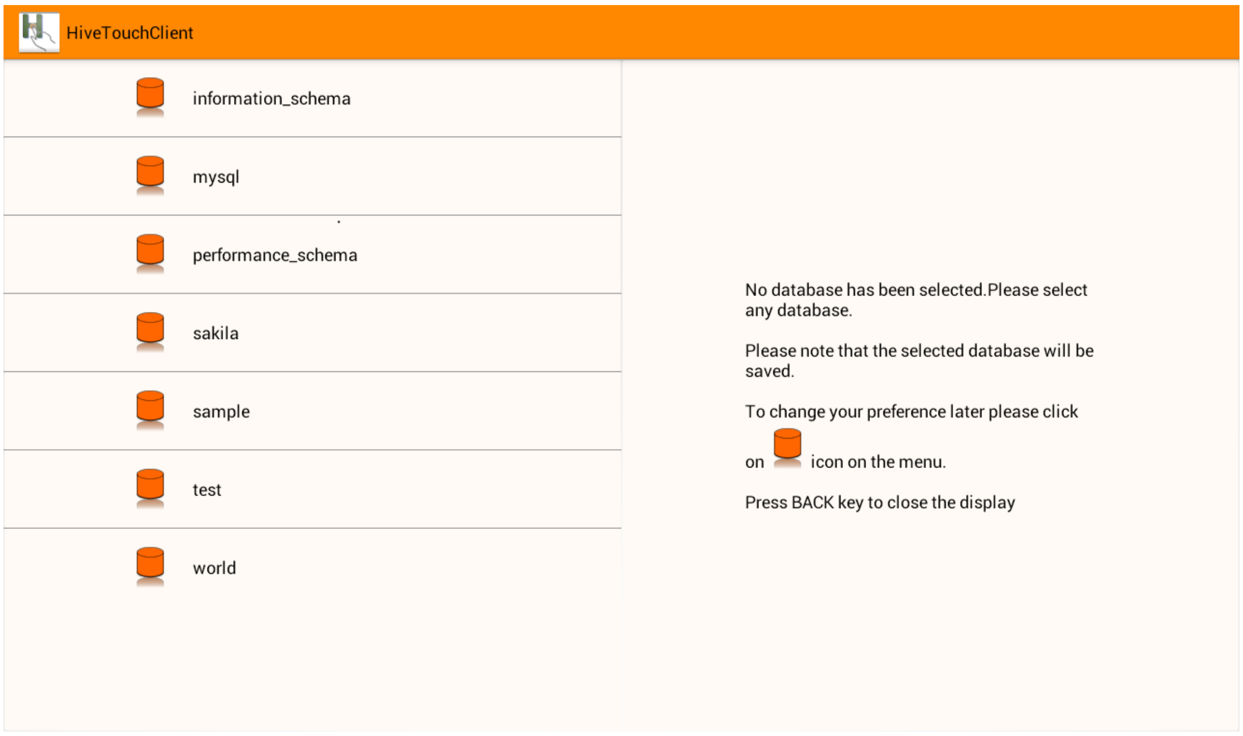
The system allows users to control and change the query whenever they want. For a touch query interface (or touch client) to be very effective, two properties are very important. First, users should be able to choose any query they want in a visual way. Second, an instantaneous feedback should be provided so that required assistance is provided to the user to construct a query without any interruption to the flow of thought. To assist every action a user does, the touch client prompts the next step appropriately followed by an action. For this study, the touch client is equipped to deal with some basic commands which are widely and highly used. Following are the commands that are currently supported

- Filter
- Sort
- Join
- Union All

This section gives a detailed explanation of how each query is implemented, how the touch client looks and how it is supposed to be used.

4.4.2 Databases

The touch client always remembers the database selected by the user. The very first time the user opens the application (touch client), the client shows the Figure 4.4 to the user. Here, the touch client notifies the user that no database has been selected and requests the user to choose one by clicking on any item on the list of databases displayed.



HiveTouchClient

- information_schema
- mysql
- performance_schema
- sakila
- sample
- test
- world

No database has been selected.Please select any database.

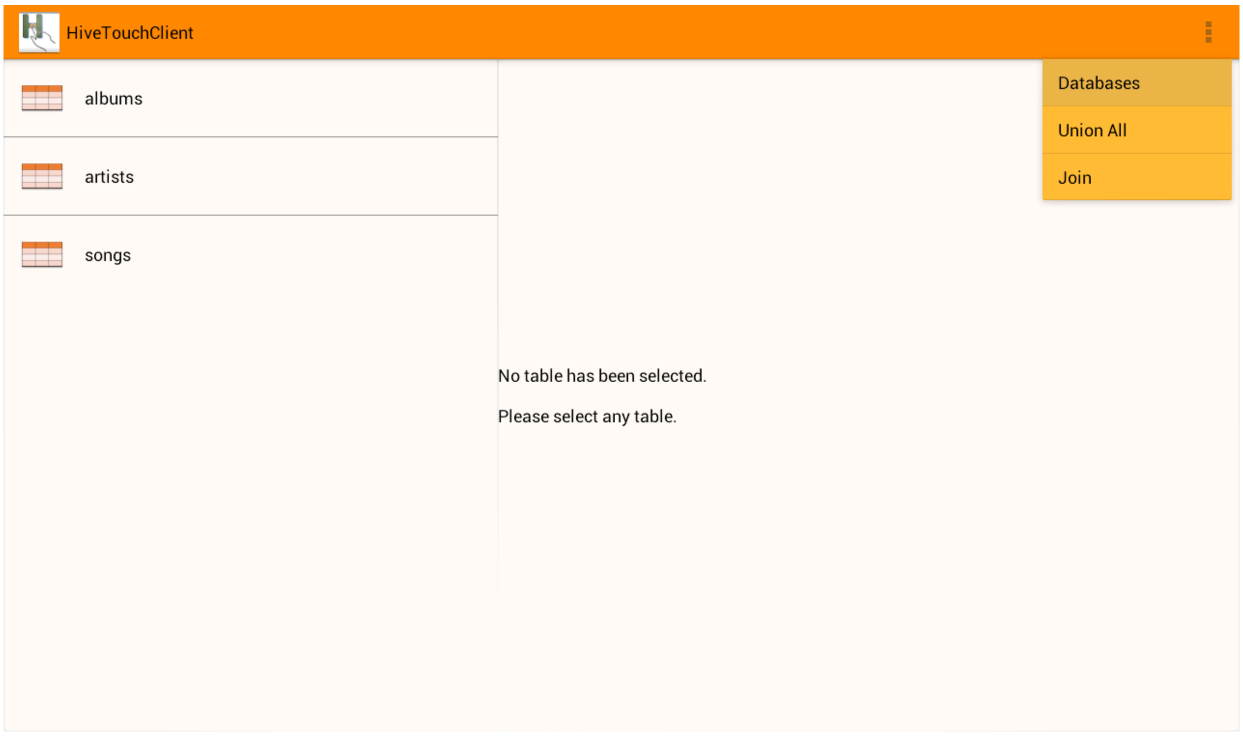
Please note that the selected database will be saved.

To change your preference later please click on icon on the menu.

Press BACK key to close the display

Figure 4.4: First screen to select a database

When the user clicks on an item, the corresponding database is saved as the choice of the user for all the forthcoming queries. Once the user selects the database, this screen is not displayed again. Thereafter, all the queries are assumed to be meant for the database that is just selected. Irrespective of the query being performed, the user always has an option of selecting a different database. Every screen of touch client has menu items which will let the user perform multiple queries. One of the menu items is *Databases*. The user can click on the *Databases* menu item to go to the list of databases, where the user can change the database. When the user clicks on *Databases*, as shown in the following figure, the user will be displayed with a screen that notifies the current selection of database. Then, the user can change the database, if necessary.
The Figure 4.5 shows where the menu item *Databases* is placed and also shows the menu being clicked.



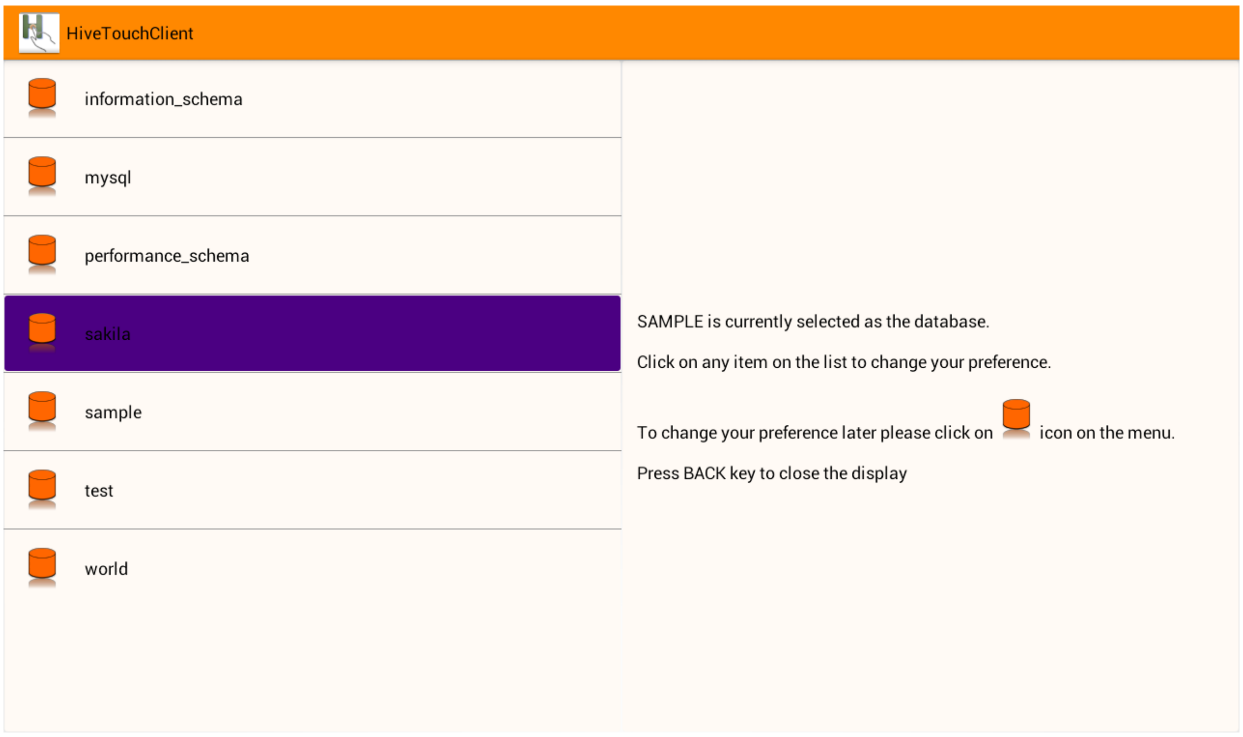
No table has been selected.

Please select any table.

Figure 4.5: Click on the "Databases" menu item to change preference of database

Please note: *The background color of the menu being clicked is always slightly different from the background color of the rest of the menu items.*

As previously stated, upon clicking the *Databases* menu item, the screen in the Figure 4.6 is displayed where the name of the currently selected database is notified to the user in the section on the right and the list of databases available for selection is displayed in the section on the left.



SAMPLE is currently selected as the database.

Click on any item on the list to change your preference.

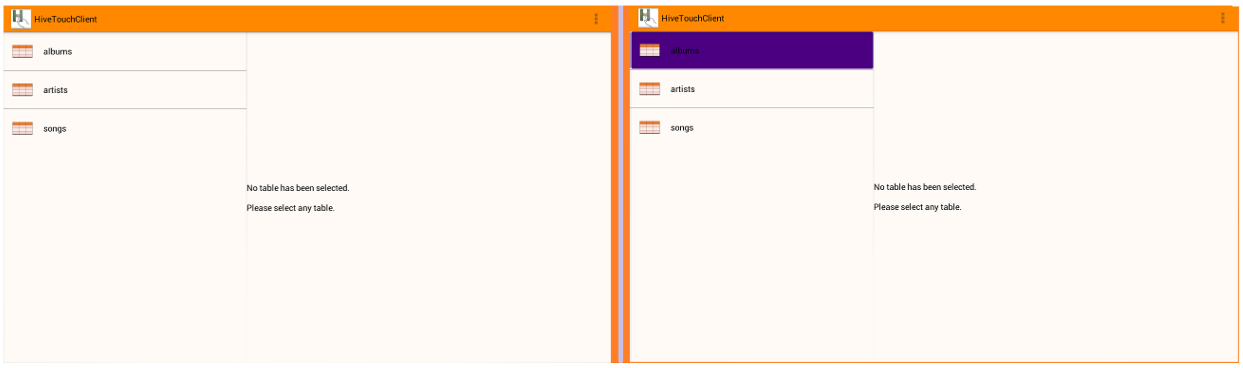
To change your preference later please click on icon on the menu.

Press BACK key to close the display

Figure 4.6: Screen to change the preference of database

4.4.3 Tables

The touch client has a different screen to list all the tables available in the selected database. Unlike a database, the tables selected by the user are not remembered. When the user visits this screen for the first time, where all the tables of the selected database are listed, the user is prompted *No table has been selected*. As shown in the image on the left of Figure 4.7, this screen is vertically divided into two sections. The section on the left displays the list of tables and the section on the right displays the records of the selected table.


HiveTouchClient

albums

artists

songs

No table has been selected.

Please select any table.

HiveTouchClient

albums

artists

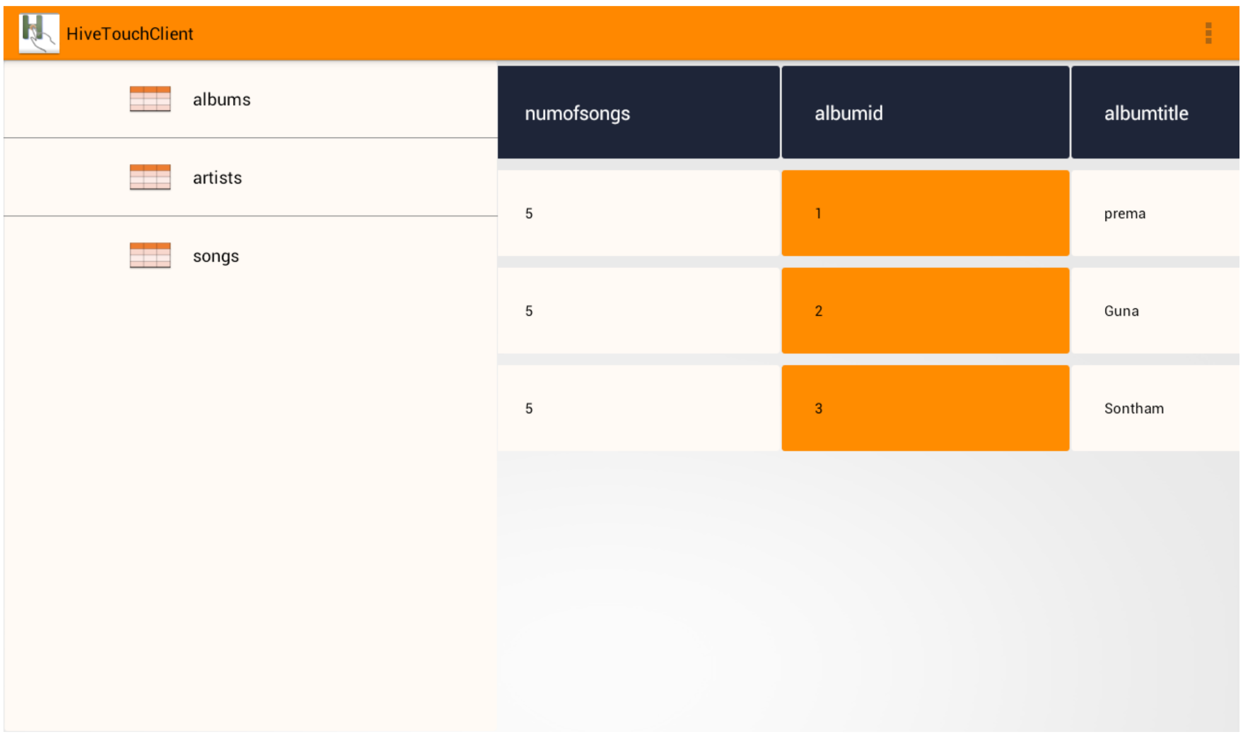
songs

No table has been selected.

Please select any table.

Figure 4.7: *To the left* : List of tables displayed *To the right* : Table being selected

All the items displayed in a list (in the section on the left), are clickable. The user can select a table by clicking on any item on the list. When the user clicks on any item on the list, the background of the list changes to violet color to indicate the user of the table that was just clicked/selected as shown in the image on the right of Figure 4.7 above. Upon clicking on an item on the list, all the records of that table will be displayed in the section on the right as shown in Figure 4.8


HiveTouchClient

albums

artists

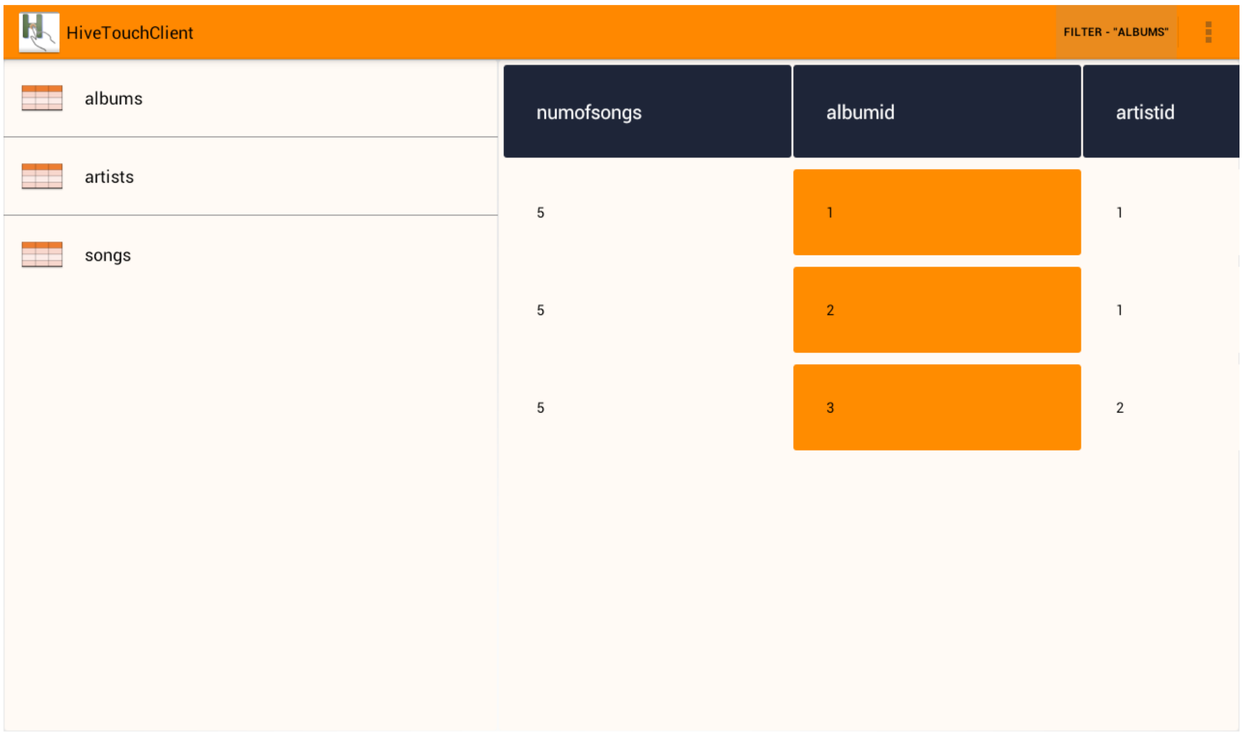
songs

numofsongs	albumid	albumtitle
5	1	prema
5	2	Guna
5	3	Sontham

Figure 4.8: Table records display

4.4.4 Filter

The touch client lets the user filter the records through the *Filter* menu item. The very first time a table is selected, all the records are fetched straightaway. As the user clicks on a different table, the section on the right gets updated with the records of the selected table. The user can click on the *filter* menu item to filter the records. The *filter* menu item is displayed on all those screens where the table records are displayed.



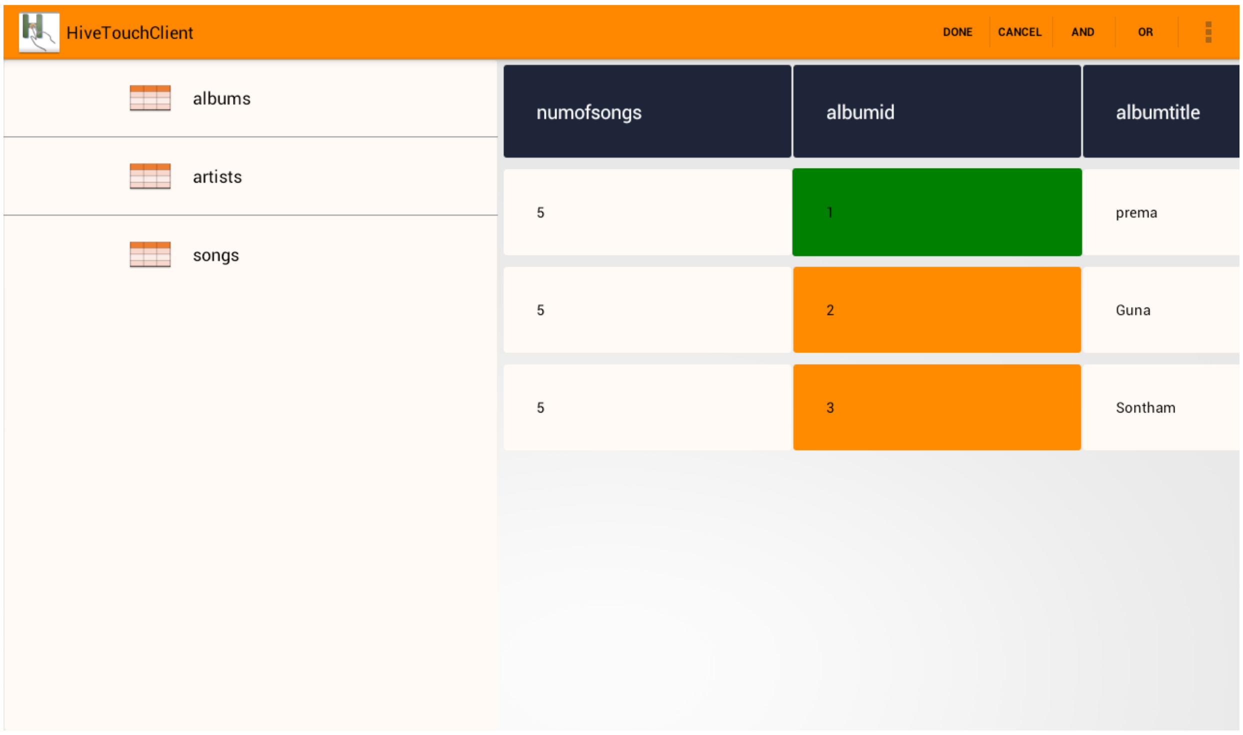
numofsongs	albumid	artistid
5	1	1
5	2	1
5	3	2

Figure 4.9: Filter menu item display

Once the user clicks on the *filter* menu item, "*Filter mode*" is activated and extra menu items required for the user to filter records are displayed. The menu items are otherwise not visible to the user unless the user chooses to activate the filter mode. For example, menu items AND, OR, DONE and CANCEL displayed in Figure 4.9 are the extra menu items which are displayed only in filter mode.

Filtering the records is like adding WHERE clause in a query statement. If clicking on a table "*albums*" is equivalent to query "*select * from albums;*", filtering is equivalent to query "select * from albums WHERE albumid = 1". This select query filters the records based on the condition that the value of *albumid* should match a particular value. This pair of the column name (*albumid* in this case) and the value it is supposed to match is defined as a "criterion" in this implementation. Similarly, the touch client allows the user to filter the records by selecting one or more criteria.

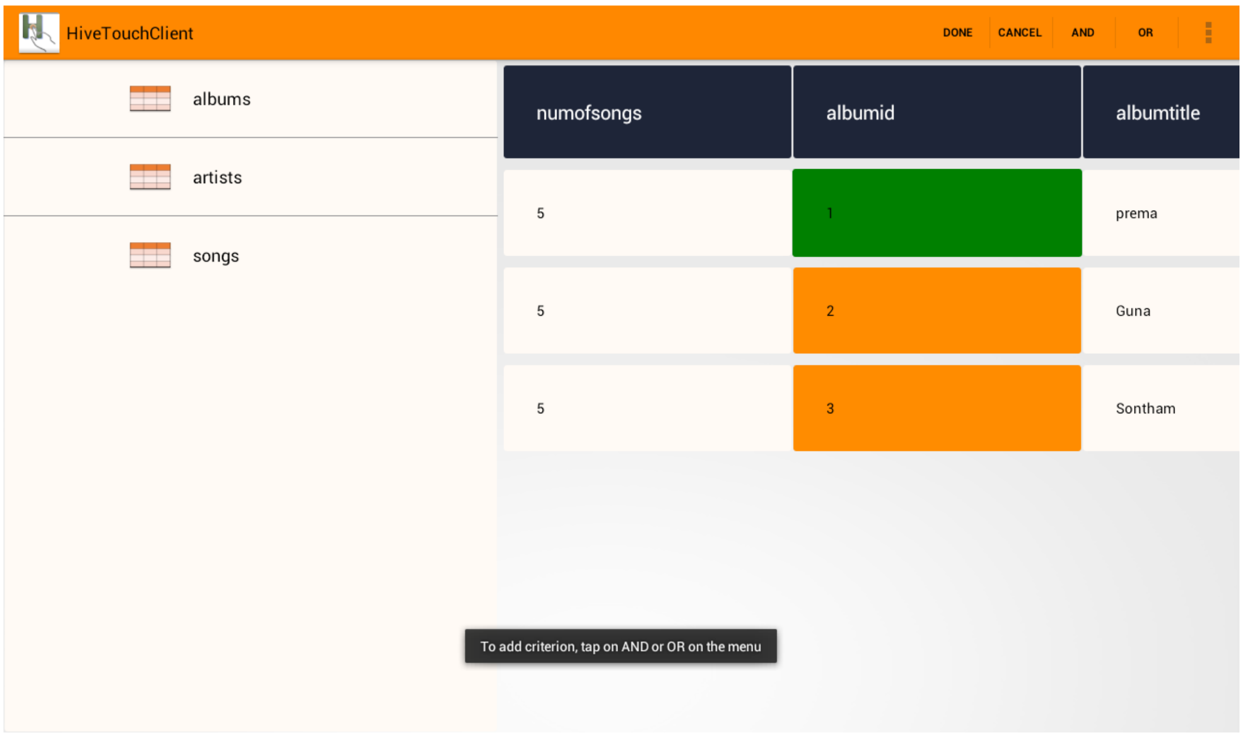
The selection of criterion is done by clicking on a box in the section on the right of the screen. Once the user clicks on a box in filter mode, the background of the box turns green color to notify the user of the criterion selected. The change in color not only helps the user know the selected criterion, but also helps the user remember all the criteria added before the user is done with filtering. The following figure indicates the criterion selected by the user to filter records. Figure 4.10 shows the user's intention of filtering all the records and return only those rows whose albumid is equal to 1.



numofsongs	albumid	albumtitle
5	1	prema
5	2	Guna
5	3	Sontham

Figure 4.10: Filter criterion selected

When the user selects a criterion, the client notifies the user what to do next to add more criteria. Figure 4.11 shows how the touch client gives feedback to the user about the next step. The user can add criteria by clicking on AND or OR menu items.



numofsongs	albumid	albumtitle
5	1	prema
5	2	Guna
5	3	Sontham

To add criterion, tap on AND or OR on the menu

Figure 4.11: Filter menu item

At any given time, the user has to click on AND or OR before the user clicks on another box (selects another criterion). If not, the client doesn't allow the user to select the extra criteria. The user clicks on the DONE menu item once he/she is done with the selection of criteria. When the user clicks DONE, the section on the right of the screen is updated with records of the filtered query. Before updating the section on the right, the client displays a dialog box to the user, where the columns of the tables to be fetched can be selected.

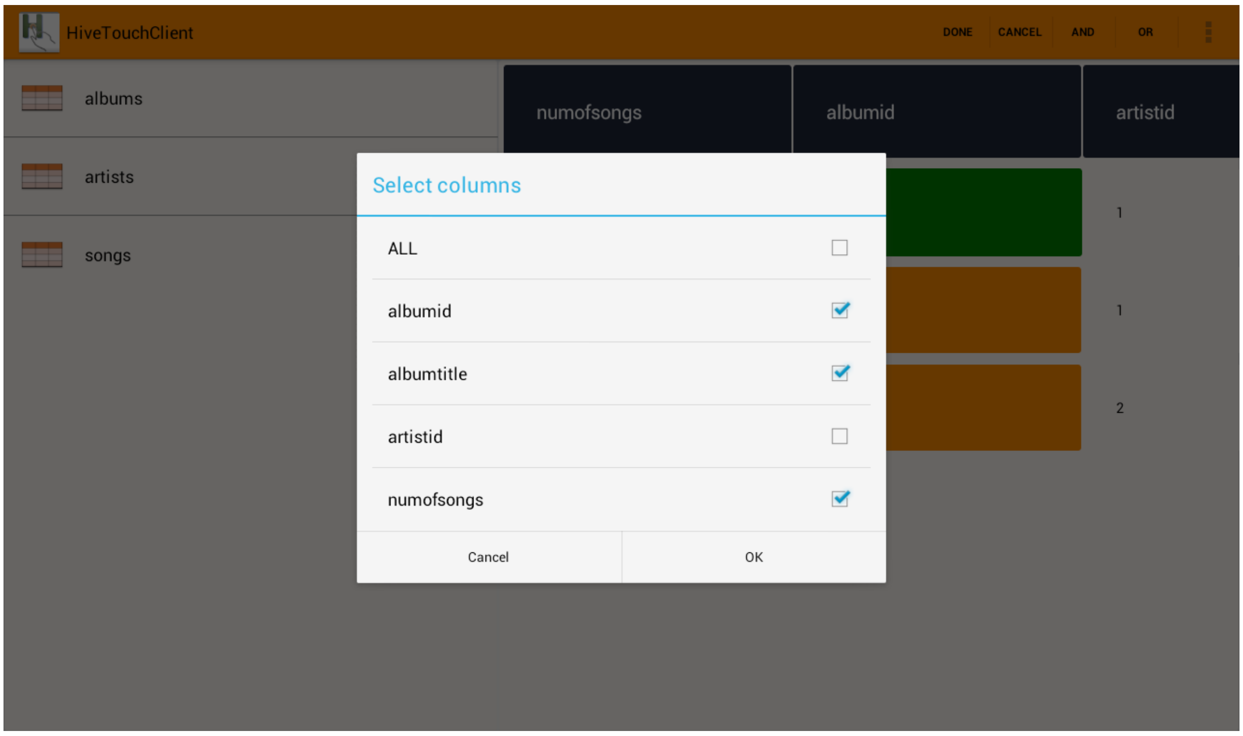


Figure 4.12: Select columns to be fetched

Once the user selects either “ALL” or specific columns and clicks on “OK” button on the dialog, the section on the right of the screen is populated with records of the table which are a result of the filter just performed by the user. The records displayed may contain all the columns or specified columns of the table based on the selection made when the dialog was displayed (refer to Figure 4.12).

If the user wants to quit the filter mode, the CANCEL menu item should be clicked. When the CANCEL menu item is clicked, all the criteria selected before the click are undone. The user should click on the *filter* menu item again to enter the filter mode and filter the records, if necessary.

4.4.5 Sort (Order by)

To sort the records of a table, the user can click on the *Order by* menu item.

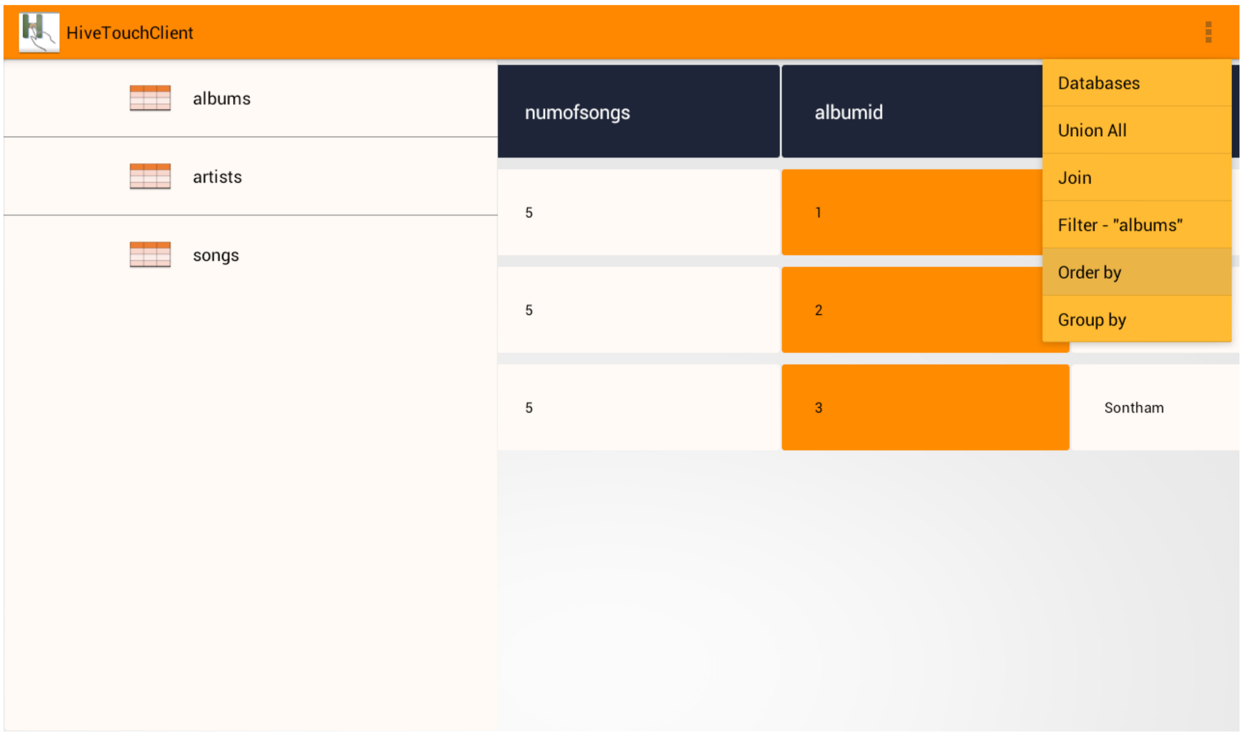


Figure 4.13: "Sort by" menu item display

Once the user clicks on the *Order by* menu item, a dialog with a list of all the column names is displayed. Each row has a switch button next to the name of the column.

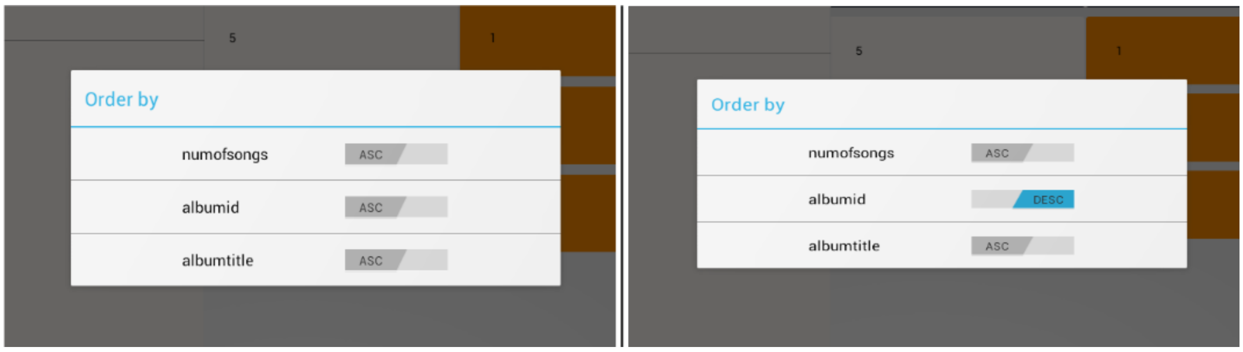
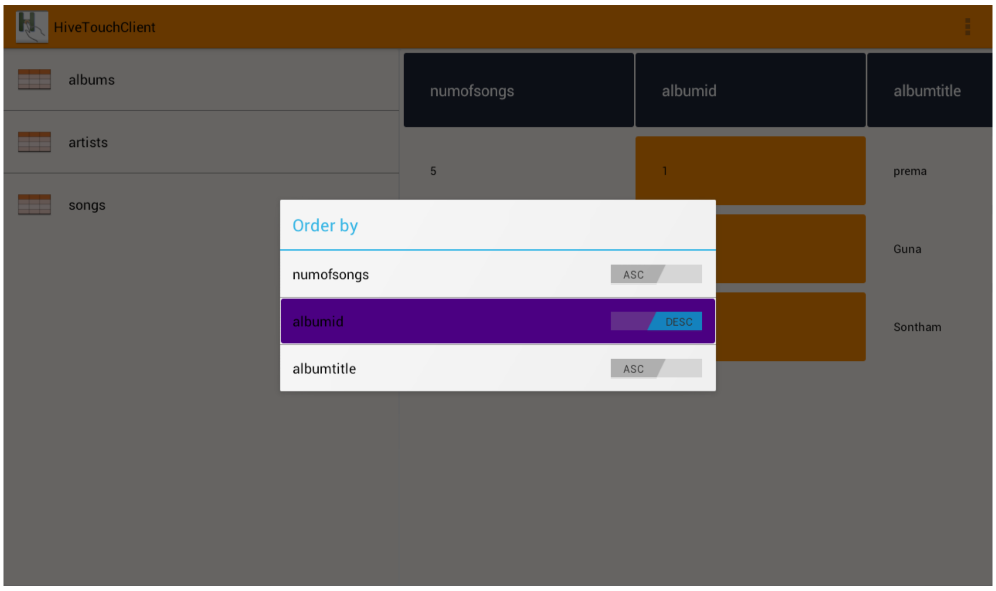


Figure 4.14: "Sort by" options dialog

As shown in Figure 4.14, the user can sort the records by ascending or descending order and set the switch button against the appropriate column name accordingly. When the column name and the order (ascending or descending) for sorting the records is determined, the user can click on the list item to finalize the selection of the criterion to sort the records.



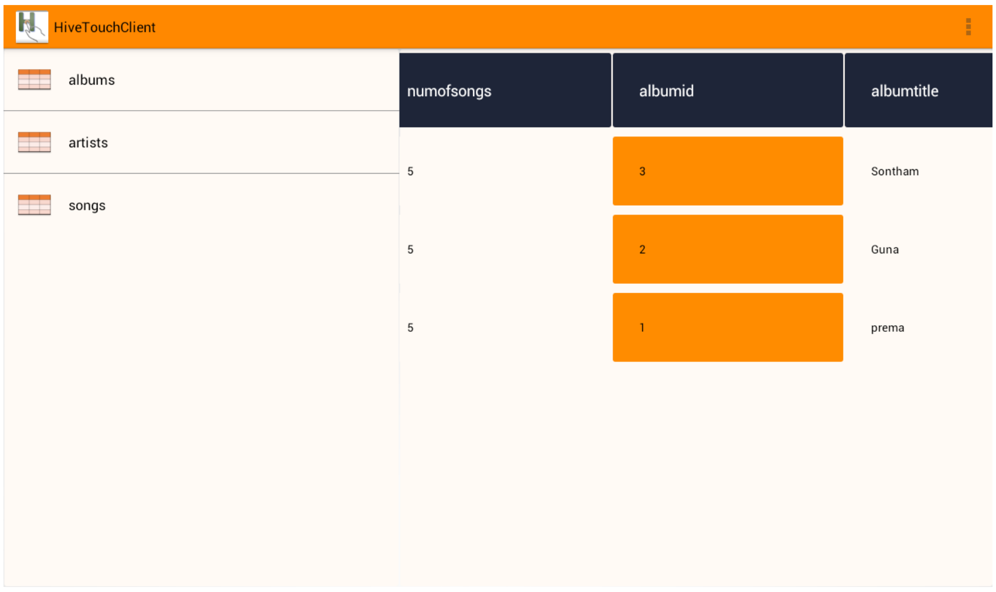
numofsongs	albumid	albumtitle
5	1	prema
		Guna
		Sontham

Order by

numofsongs	ASC
albumid	DESC
albumtitle	ASC

Figure 4.15: "Sort by" options dialog

Figure 4.15 shows that the user chose to sort the records by *albumid* in descending order. The results are then fetched again and sorted in the order selected by the user. The succeeding figure shows the results sorted by *albumid* in descending order.



numofsongs	albumid	albumtitle
5	3	Sontham
5	2	Guna
5	1	prema

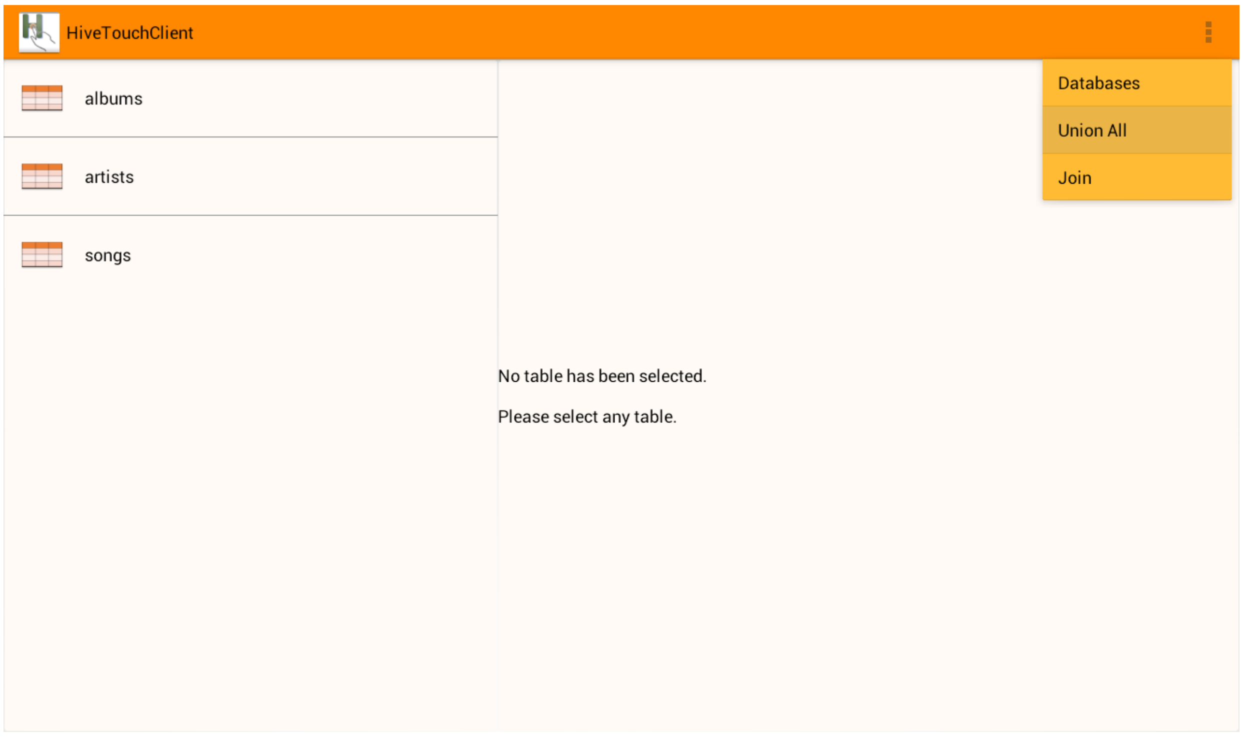
Figure 4.16: Results sorted by albumid

4.4.6 Union all

To combine the result set of 2 or more select statements, *union all* is used in both MySQL or HiveQL. MySQL has just UNION as well. *Union all* has a few requirements to be satisfied for the query to work just fine. They are

- Each select statement combined by *union all* should project the same number of columns.
- Corresponding columns of each select statement must have the same data types.
- Corresponding columns projected by select statement with similar data types should be in the same order.

In a nutshell, the projection of each select statement should have the same number of columns with the same data types in the same order. In the touch client, the user can perform *union all* of two statements by clicking on the *Union all* menu item. All the screens have this menu item except the one that handles *union all* operation. Like all the other operations, *union all* is also implemented with a feedback system that tells the user what to do next.



No table has been selected.

Please select any table.

Figure 4.17: Go to the screen that handles *union all*

When the user is directed to the screen that handles *union all* operation, a screen is displayed that is vertically divided into two sections. The section on the left shows a list of tables from which the user is expected to select the table to be retrieved as shown in Figure 4.18

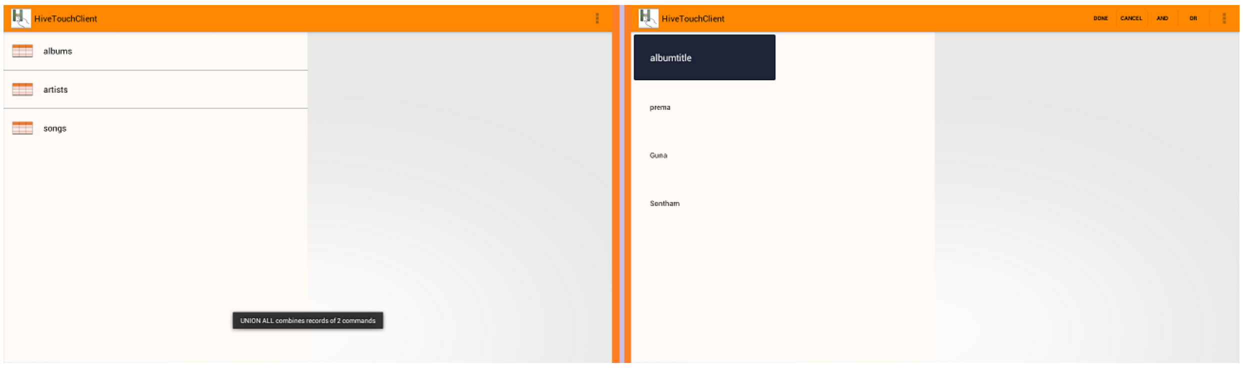


Figure 4.18: *To the left* : First table list for UNION ALL *To the right* : First table filter done

Once the user selects the table to be fetched, the section on the left is replaced by all records from the selected table, as shown in the image on the right of Figure 4.18. The user can then filter the records and click on the DONE menu item which shows up in filter mode. The user then goes through the same process of selecting the required columns to be fetched as a part of the first select statement.
The section on the right shows up with a list of tables only after the user is done with first select statement as shown in Figure 4.19. The user is expected to go through the same routine for the second select statement as well. Once the user selects a table, the records are populated on the right section where the user is allowed to filter the records. Please refer to the image on the right of Figure 4.19.

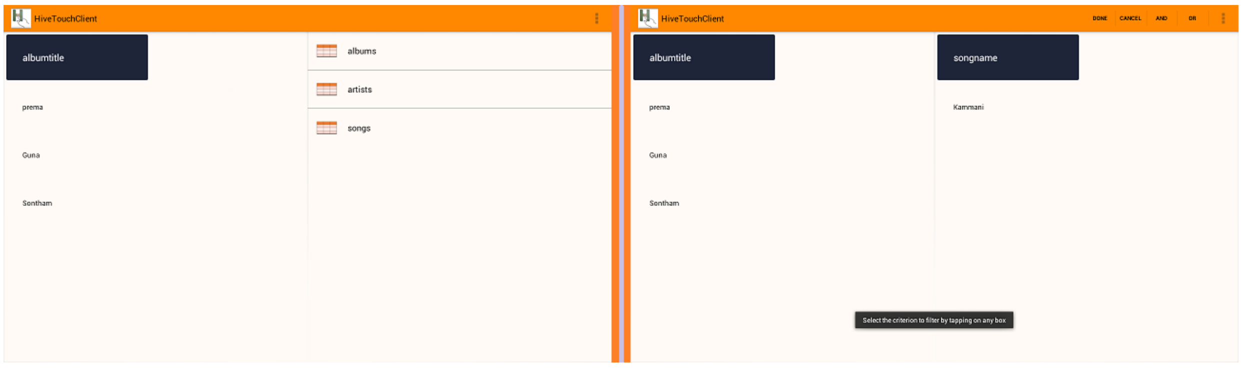
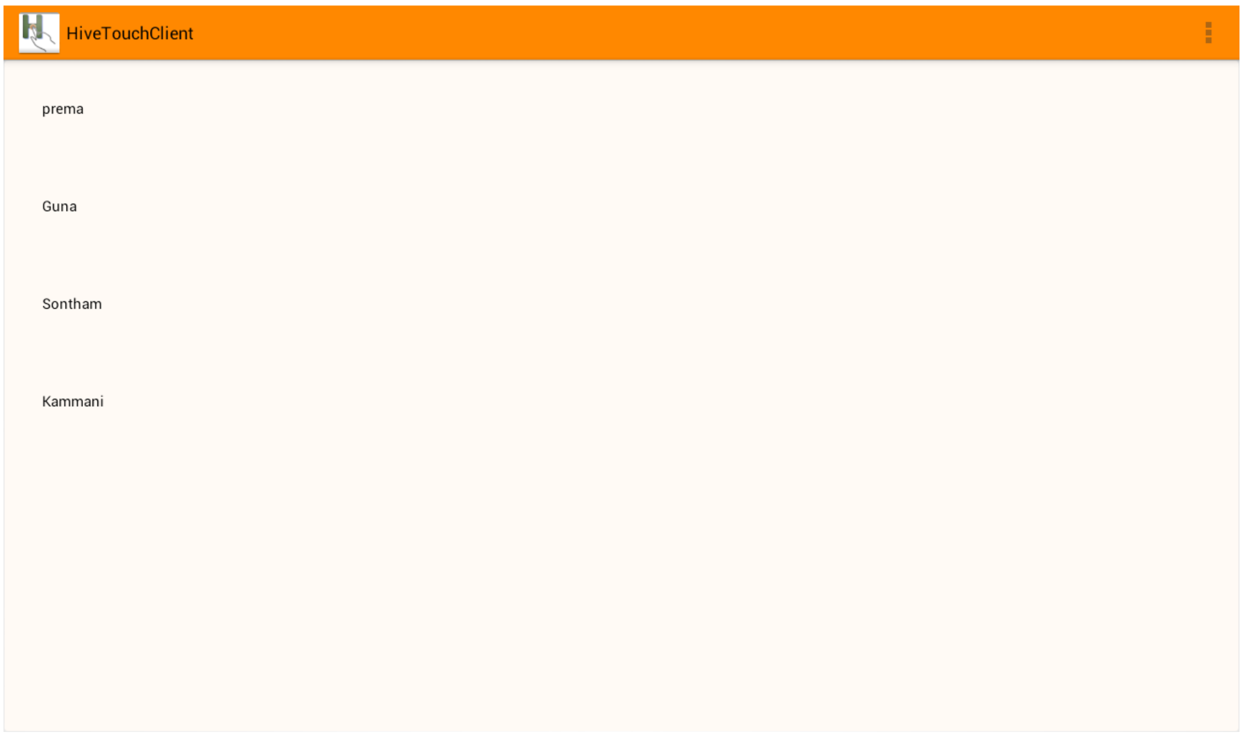


Figure 4.19: *To the left* : Second table list for UNION ALL *To the right* : Second table filter done

Once the criteria to filter the records of the table displayed in the section on the right is determined, the user can click on the DONE menu item to expect the final result set of two select statements combined by a *union all* operator. The result set will be displayed on the final screen. The results are displayed similar to the ones displayed on MySQL workbench. The results displayed on the final screen will not have column names as it is considered inappropriate in this case where the column names fetched by the two selected statements are different though they are of same data types.



prema

Guna

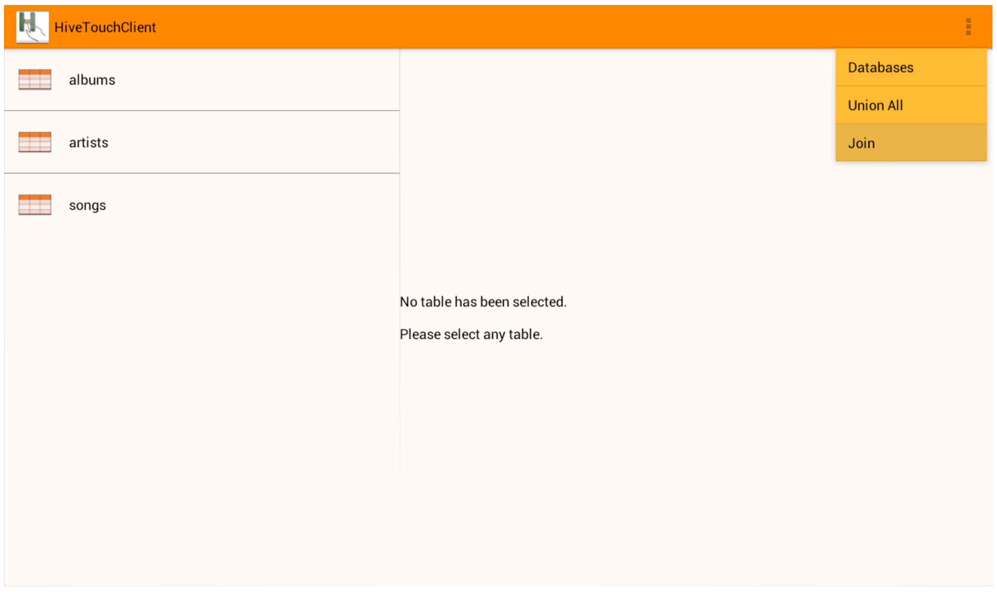
Sontham

Kammani

Figure 4.20: *Union all* query results

4.4.7 Join

It is well known that the JOINS are used to combine the rows from multiple tables (two or more) based on the common fields (or columns) between them. The join implemented in this prototype is an INNER JOIN as this is the most common type of join. The usage pattern of JOIN in touch client is very similar to the usage of *union all* to avoid a situation, where the user needs to learn how join is used. The user can perform a join by clicking on *Join* menu item as shown in the following figure. All the screens have this menu item except the one that handles the join operation. Like all the other operations, Join is also implemented with a feedback system.

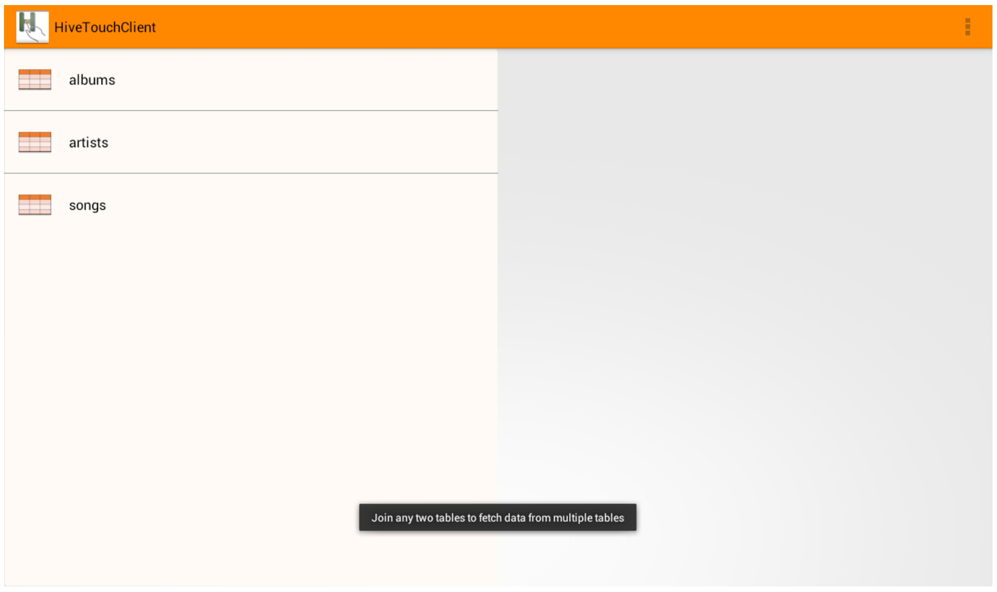


No table has been selected.

Please select any table.

Figure 4.21: Screen that handles Join operation

When the user is directed to the screen that handles the join operation, a screen is displayed which is vertically divided into two sections. The section on the left shows a list of tables and the user is expected to select the table to be fetched.



Join any two tables to fetch data from multiple tables

Figure 4.22: Select first table to join

When the user clicks on any item on the list of tables, the client displays a dialog with a list of the items (names of the columns). The user can select all or specific columns to be fetched from this table as a part of the join operation.

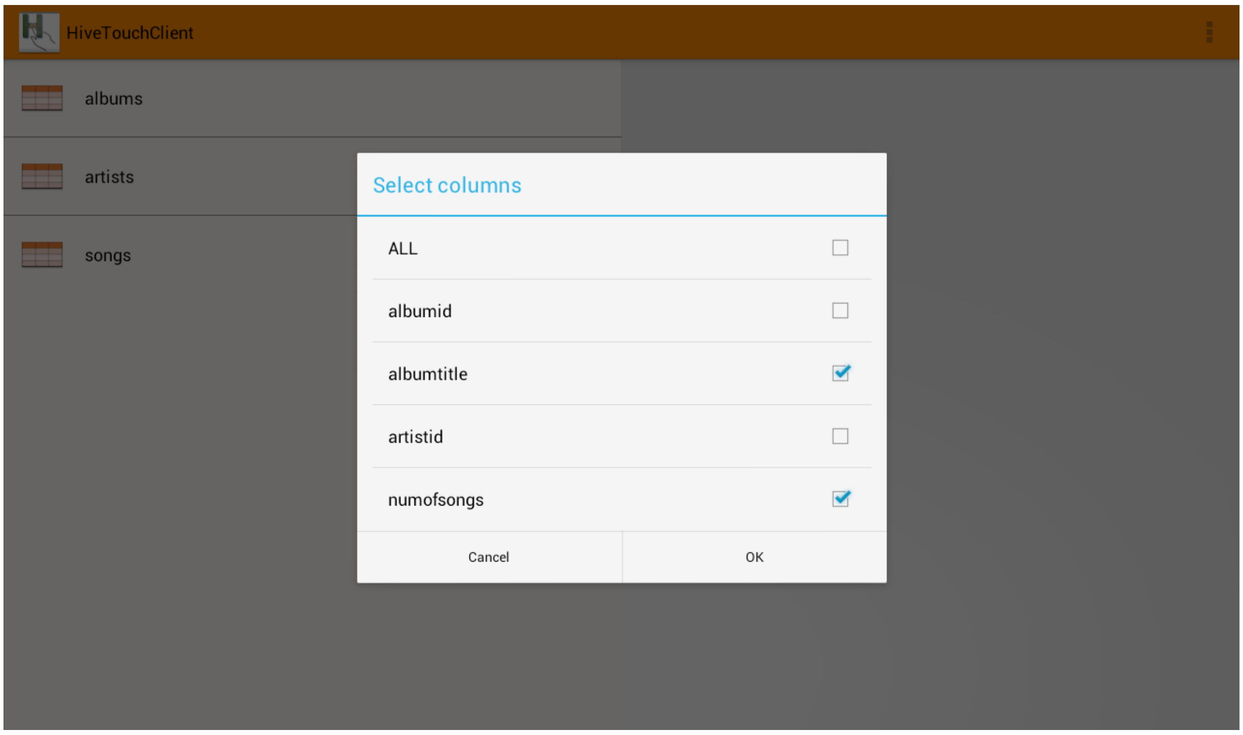
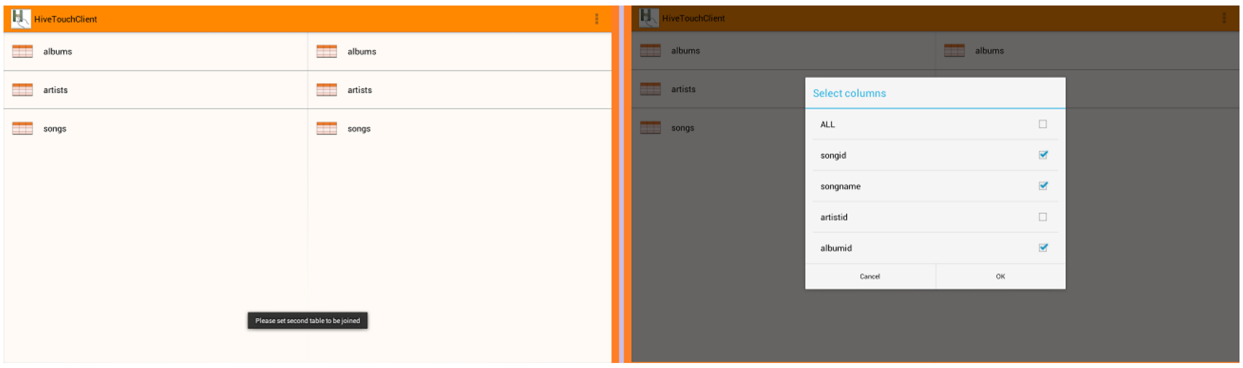


Figure 4.23: Select projection of first table for join

Once the selection is made and the user clicks the OK button on the dialog box, the section on the right of the join screen is populated with the list of tables from the database again. The feedback system asks the user to select the second table to be joined with the first table. Please observe Figure 4.24. The user goes through the same process of selecting a table from the section on the right, followed by the selection of columns to be fetched from the second table.



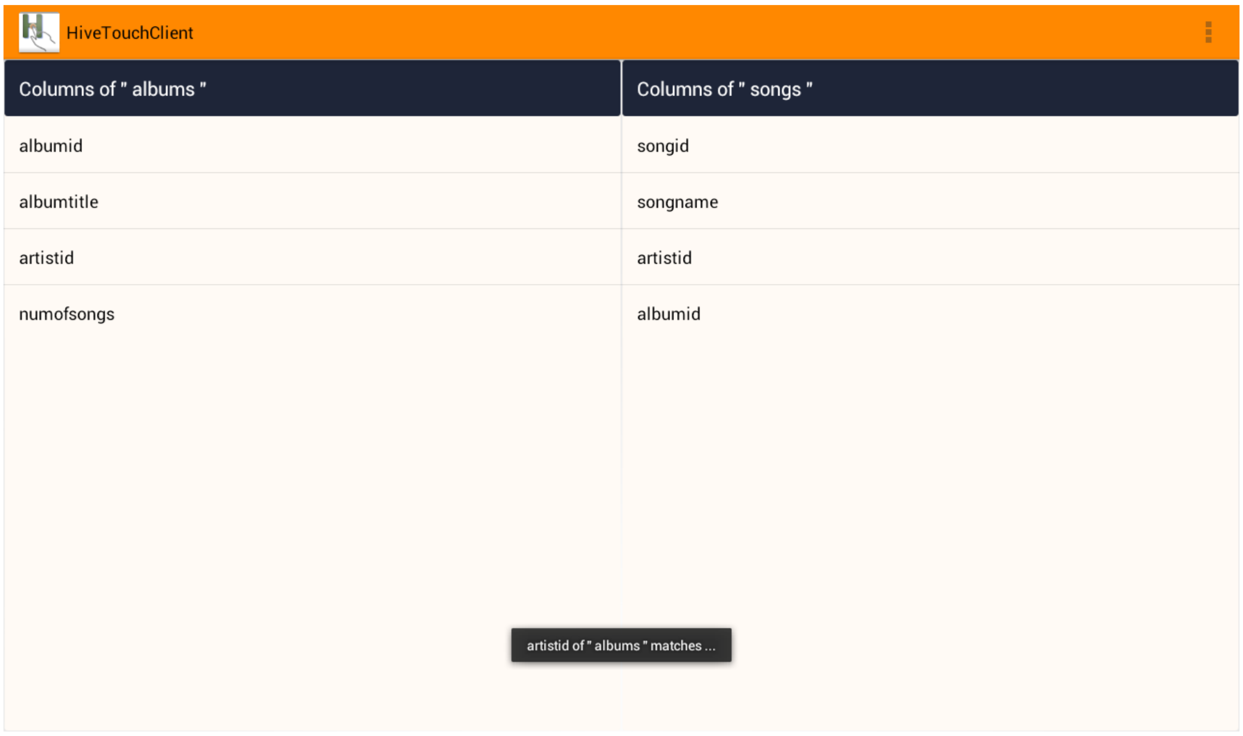
Please set second table to be joined

Select columns

- ALL ☐
- songid ✓
- songname ✓
- artistid ☐
- albumid ✓

Figure 4.24: *To the left* : Select second table on right section for join operation *To the right* : Select projection of second table for join

Once the projection for the second table is selected, the user is then directed to a screen where the task is to select the join conditions. It is to say that the column1 from the first table should match the column2 of the second table. The user can select multiple join conditions by pressing AND or OR menu items between two join conditions.

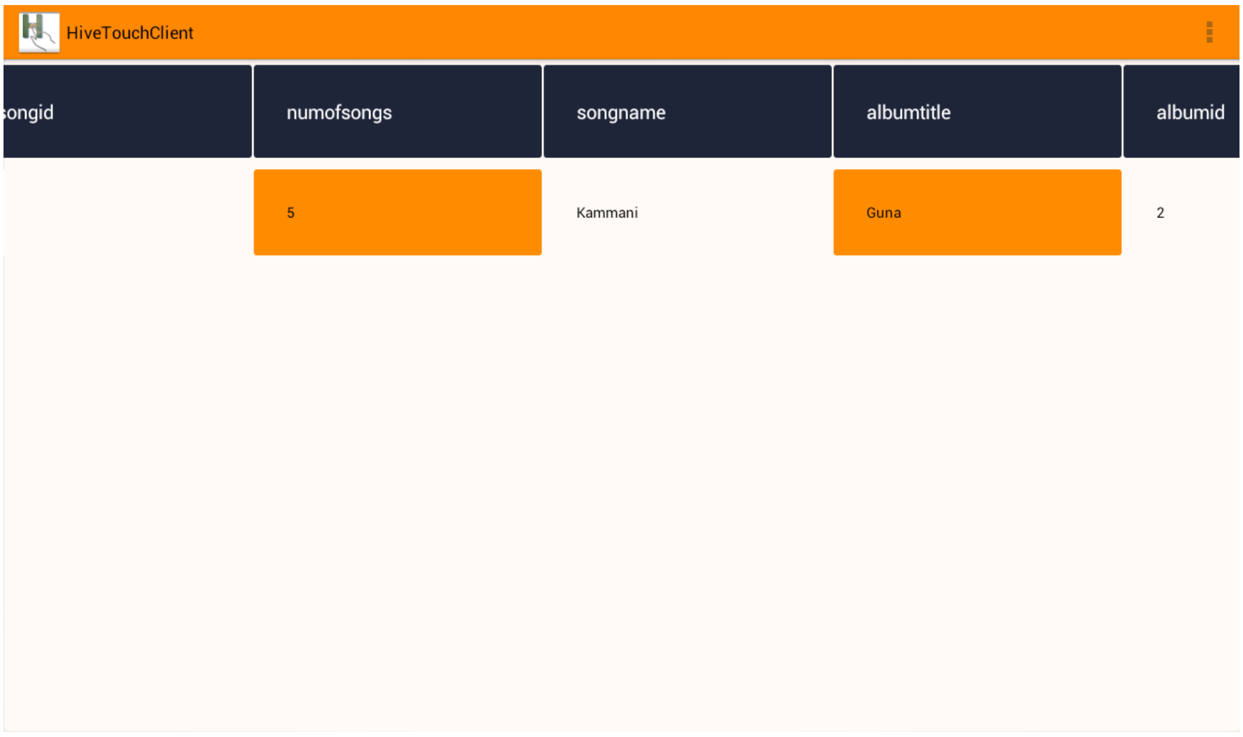


Columns of " albums "	Columns of " songs "
albumid	songid
albumtitle	songname
artistid	artistid
numofsongs	albumid

artistid of " albums " matches ...

Figure 4.25: Select join conditions i.e. which column of first table should match which column of second table

Figure 4.25 shows the selection of first join condition. So, the menu items such as AND, OR and DONE are not visible. These menu items are automatically visible once the user completes the selection of the first join condition. When the user is done with the selection of the join conditions, DONE should be clicked to fetch the result set of the final join query which looks like the following figure.



ongid	numofsongs	songname	albumtitle	albumid
	5	Kammani	Guna	2

Figure 4.26: Join query results

This way the user can use this touch client which shows the data in a more understandable way. The user is always assisted by a feedback system which further lowers the scope of confusion. The touch client allows one to switch from one query/task to another irrespective of one's current activity.

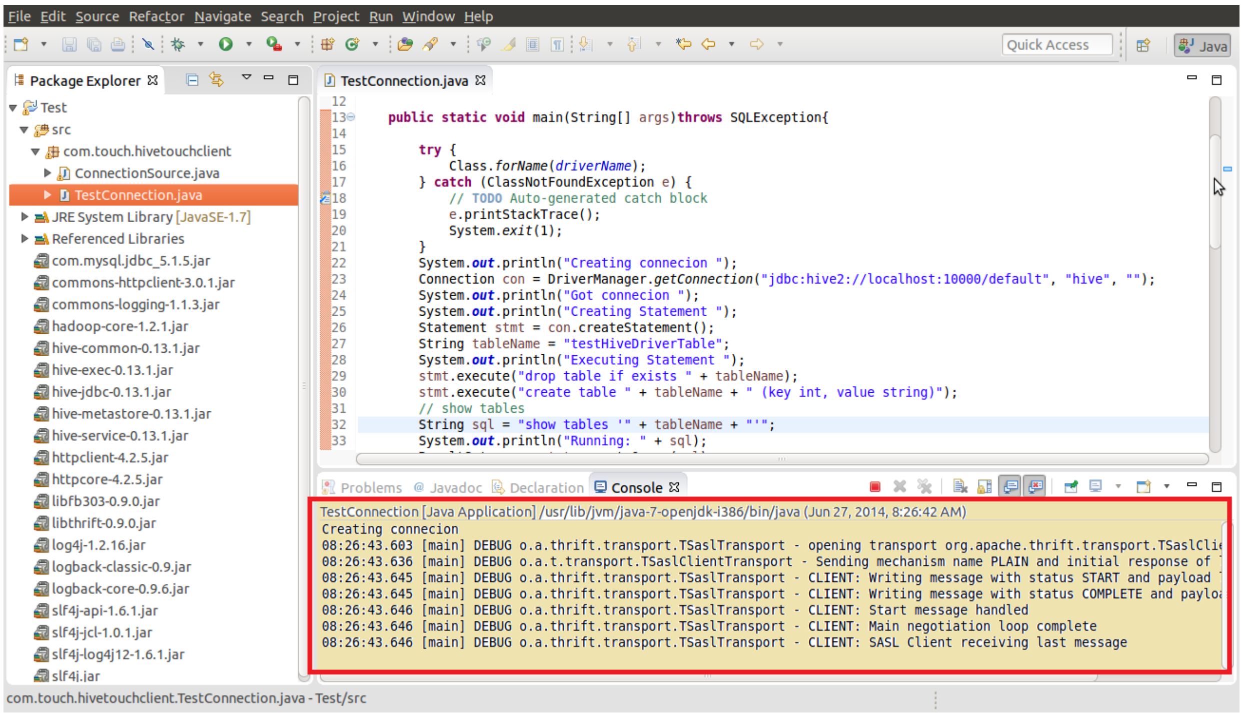
4.5 Problems faced with HiveServer

The back-end is configured on a virtual machine with Ubuntu as the guest OS. The Ubuntu has a Hadoop installed with 1 namenode and 1 datanode. Hive is also configured on the same guest OS. The sample data are placed on HDFS in the form of text files. The tables are created and populated in Hive by pushing the data from the files to the tables. These tables can be accessed via JDBC from a HiveClient written in Java with an instance of HiveServer running on the guest OS.

Initially the JDBC connection couldn't be established as the request was not forwarded to the guest OS. This was solved with the correct settings of forward porting. Though there were such minor issues, most of them were solved until one issue was faced.

4.5.1 Issue

When the client tries to establish a connection to the HiveServer, the request is forwarded to the server, but the call doesn't return to the client. Please refer to Figure 4.27. As shown in the logs, the client receives a SASL message from the server, then the client hangs at this point. The portion of the figure that displays the logs is highlighted.



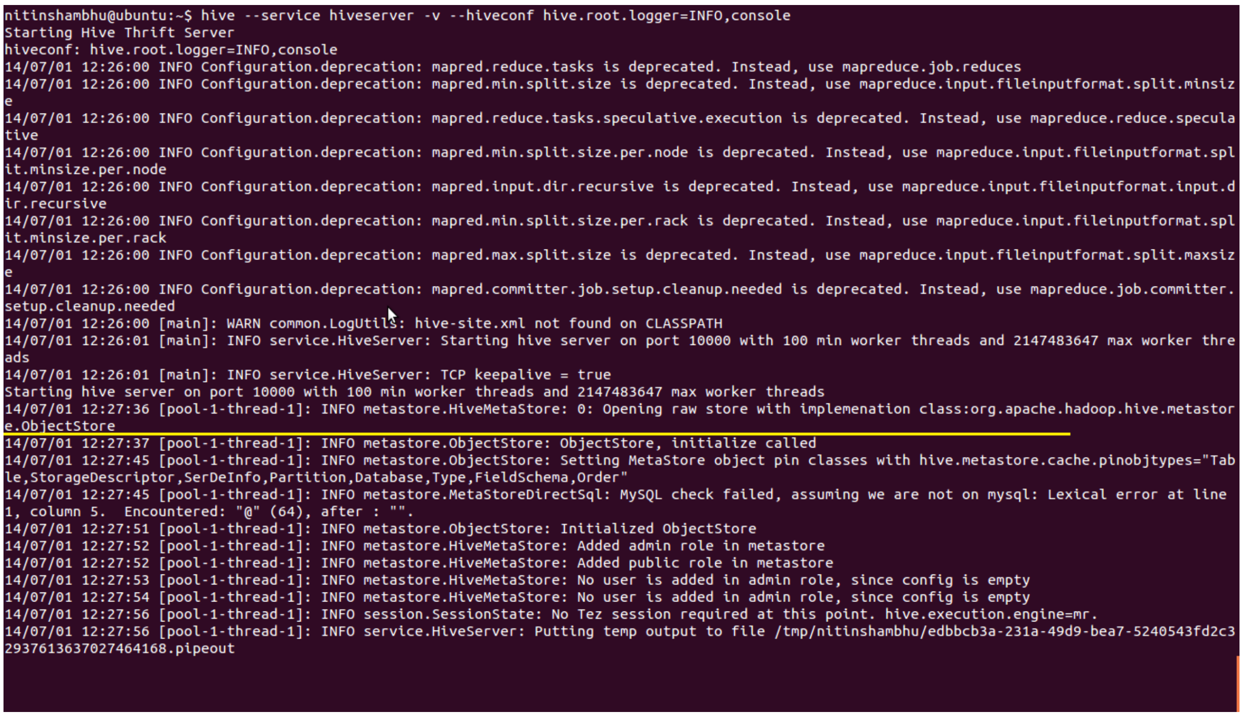
```
    public static void main(String[] args)throws SQLException{

        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            System.exit(1);
        }
        System.out.println("Creating connecion ");
        Connection con = DriverManager.getConnection("jdbc:hive2://localhost:10000/default", "hive", "");
        System.out.println("Got connecion ");
        System.out.println("Creating Statement ");
        Statement stmt = con.createStatement();
        String tableName = "testHiveDriverTable";
        System.out.println("Executing Statement ");
        stmt.execute("drop table if exists " + tableName);
        stmt.execute("create table " + tableName + " (key int, value string)");
        // show tables
        String sql = "show tables '" + tableName + "'";
        System.out.println("Running: " + sql);
```

```
TestConnection [Java Application] /usr/lib/jvm/java-7-openjdk-i386/bin/java (Jun 27, 2014, 8:26:42 AM)
Creating connecion
08:26:43.603 [main] DEBUG o.a.thrift.transport.TSaslTransport - opening transport org.apache.thrift.transport.TSaslCli
08:26:43.636 [main] DEBUG o.a.t.transport.TSaslClientTransport - Sending mechanism name PLAIN and initial response of
08:26:43.645 [main] DEBUG o.a.thrift.transport.TSaslTransport - CLIENT: Writing message with status START and payload
08:26:43.645 [main] DEBUG o.a.thrift.transport.TSaslTransport - CLIENT: Writing message with status COMPLETE and paylo
08:26:43.646 [main] DEBUG o.a.thrift.transport.TSaslTransport - CLIENT: Start message handled
08:26:43.646 [main] DEBUG o.a.thrift.transport.TSaslTransport - CLIENT: Main negotiation loop complete
08:26:43.646 [main] DEBUG o.a.thrift.transport.TSaslTransport - CLIENT: SASL Client receiving last message
```

Figure 4.27: Client logs when client tries to establish a connection

Though all the solutions available over the internet have been tried, nothing worked. One important suggestion to verify the logs of HiveServer was taken into consideration to investigate the problem further. However, even the logs of the server haven't displayed anything abnormal (such as exceptions). Please refer to the figure. All the logs below the yellow line are populated only after the client initiated the connection. As one can observe there is no abnormality observed thus leading to a deadlock.



```
nitinshambhu@ubuntu:~$ hive --service hiveserver -v --hiveconf hive.root.logger=INFO,console
Starting Hive Thrift Server
hiveconf: hive.root.logger=INFO,console
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.min.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsiz
e
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.specula
tive
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.min.split.size.per.node is deprecated. Instead, use mapreduce.input.fileinputformat.spl
it.minsize.per.node
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.input.dir.recursive is deprecated. Instead, use mapreduce.input.fileinputformat.input.d
ir.recursive
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.min.split.size.per.rack is deprecated. Instead, use mapreduce.input.fileinputformat.spl
it.minsize.per.rack
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.max.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.maxsiz
e
14/07/01 12:26:00 INFO Configuration.deprecation: mapred.committer.job.setup.cleanup.needed is deprecated. Instead, use mapreduce.job.committer.
setup.cleanup.needed
14/07/01 12:26:00 [main]: WARN common.LogUtils: hive-site.xml not found on CLASSPATH
14/07/01 12:26:01 [main]: INFO service.HiveServer: Starting hive server on port 10000 with 100 min worker threads and 2147483647 max worker thre
ads
14/07/01 12:26:01 [main]: INFO service.HiveServer: TCP keepalive = true
Starting hive server on port 10000 with 100 min worker threads and 2147483647 max worker threads
14/07/01 12:27:36 [pool-1-thread-1]: INFO metastore.HiveMetaStore: 0: Opening raw store with implemenation class:org.apache.hadoop.hive.metastor
e.ObjectStore
14/07/01 12:27:37 [pool-1-thread-1]: INFO metastore.ObjectStore: ObjectStore, initialize called
14/07/01 12:27:45 [pool-1-thread-1]: INFO metastore.ObjectStore: Setting MetaStore object pin classes with hive.metastore.cache.pinobjtypes="Tab
le,StorageDescriptor,SerDeInfo,Partition,Database,Type,FieldSchema,Order"
14/07/01 12:27:45 [pool-1-thread-1]: INFO metastore.MetaStoreDirectSql: MySQL check failed, assuming we are not on mysql: Lexical error at line
1, column 5.  Encountered: "@" (64), after : "".
14/07/01 12:27:51 [pool-1-thread-1]: INFO metastore.ObjectStore: Initialized ObjectStore
14/07/01 12:27:52 [pool-1-thread-1]: INFO metastore.HiveMetaStore: Added admin role in metastore
14/07/01 12:27:52 [pool-1-thread-1]: INFO metastore.HiveMetaStore: Added public role in metastore
14/07/01 12:27:53 [pool-1-thread-1]: INFO metastore.HiveMetaStore: No user is added in admin role, since config is empty
14/07/01 12:27:54 [pool-1-thread-1]: INFO metastore.HiveMetaStore: No user is added in admin role, since config is empty
14/07/01 12:27:56 [pool-1-thread-1]: INFO session.SessionState: No Tez session required at this point. hive.execution.engine=mr.
14/07/01 12:27:56 [pool-1-thread-1]: INFO service.HiveServer: Putting temp output to file /tmp/nitinshambhu/edbbcb3a-231a-49d9-bea7-5240543fd2c3
2937613637027464168.pipeout
```

Figure 4.28: Server logs when client tries to establish a connection

Having got stuck with this deadlock, it is decided to simulate the connectivity with MySQL Server as the back-end due to time constraints. As MySQL is very similar to HiveQL, the implementation of the prototype will have to make minimal changes to ensure a smooth functioning of the touch client. Among all the features supported for the current prototype, only Union all and join operations differ in the way they are handled in MySQL and HiveQL. The next section discusses the changes that are to be made to the existing implementation to support HiveQL, thereby explaining how this extra layer of abstraction can bring down the overhead of learning HiveQL.

4.6 Implementation with Hive

The simulation of Hive connectivity was done with MySQL as the commands of both HiveQL and MySQL are almost the same. This section explains how *union all* and *join* commands will be handled if the system has Hive server as the back-end. Only two commands are discussed in this section as the rest of the features supported in this prototype are same in both MySQL and

HiveQL. Though the commands of HiveQL and MySQL are very similar, there are notable differences between them in some cases. It is because of these differences that the touch client makes more sense in the case of HiveQL.

Consider the tables 3.1, 3.2 & 3.3 for a better understanding of the differences between MySQL and HiveQL as far as *union all* and join commands are concerned.

4.6.1 Union all

Differences between MySQL and HiveQL

Quoting the requirements of *union all* command for MySQL again.

- Each select statement combined by *union all* should project the same number of columns.
- Corresponding columns of each select statement must have the same data types.
- Corresponding columns projected by select statement with similar data types should be in the same order.

In addition to the aforementioned points, HiveQL has an extra requirement

- All select statements of *union all* command should project the same columns i.e. columns with the same name and type.

Consider the following simple command

```
Select numofsongs from album UNION ALL Select songid from songs;
```

The above command is a legitimate MySQL command as both the select statements project single column of same data type INT (Order of projection doesn't apply here as a single column is selected). If this command is executed, the results would be displayed without any problem. But the same command is illegitimate for Hive as both the select statements are projecting different columns i.e first select statement projects *numofsongs* and second select statement projects *songid*. An exception will be thrown by Hive that notifies the user that *numofsongs* column doesn't exist in the second table. So, the projection of both select statements should be same as shown in the following command where both the select statements are projecting same column (i.e. *albumid*).

```
Select albumid from albums UNION ALL Select albumid from songs;
```

While projection is a bit more restrictive in HiveQL, there is a difference in the structure of the query as well. Consider the following command.

```
Select albumid from albums UNION ALL Select albumid from songs;
```

Though this command is legitimate in the Hive, it doesn't execute as Hive requires the aforementioned command to be wrapped in one more select statement (underlined in the following command).

select albumid from (*select albumid from albums union all select albumid from songs*) final;

Execution

The touch client currently makes sure that the number of columns, their data types and their order of projection are the same. If the Hive is the back-end, an extra check will be added to make sure that all the select statements project the same columns.
Once the user is done performing *union all* by clicking on the DONE menu item, the touch client will wrap the resulting query with a select statement that projects same columns as shown in the above command. The current implementation will have two additions.

- A check to make sure the column names projected by both the select statements are same. Once the user is done with the first select statement for *union all*, the list of tables displayed to the user to select the second select statement will be populated only with the names of only those tables that have the column names projected in the first select query.
  For example, if the first table projects *albumname* from the table albums, then the user should be displayed with a list of tables which have *albumname* as a column for selection of second select statement.

- Once the user clicks on the DONE menu item, wrap the resulting query as shown in the above command before executing it through a background task.

4.6.2 Join

Differences between MySQL and HiveQL

Join makes a very compelling case to justify why touch client for Hive is a good idea. Join command in MySQL is a read operation. When a join command is executed, it joins the tables and returns the results. But, join command in HiveQL is a write operation and returns nothing. A simple join command in Hive looks like this.

```
INSERT OVERWRITE TABLE join_results SELECT songs.songname , albums.albumname
    , albums.numofsongs FROM albums JOIN songs ON (songs.artistid = albums.
    artistid);
```

It is obvious from the above command that the results are written into a table called *join_results*. For this to happen, a table with name *join_results* should exist already. It doesn't automatically create *join_results* table if it doesn't exist. Before performing a join operation, the user must create a table to store the results of join command. This can be done with the help of the following command.

```
CREATE TABLE join_results (songname STRING, albumname STRING, numofsongs INT)
    ;
```

Once the table is created and join command is implemented, the results are written to *join_results* table. The user can see the results with the following command.

```
Select * from join_results;
```

Execution

In a nutshell, it takes three commands in HiveQL to do what a single join command in MySQL does. This is precisely where touch client has an advantage of not forcing the user to go through all these commands every time a join operation is performed. Once the user clicks on the DONE menu item after performing the join operation, the task that executes the query extracts the projection of the join query and performs all the three queries for the user in the background in the same order.

1. `CREATE TABLE join_results (songname STRING, albumname STRING, numofsongs INT);`

2. `INSERT OVERWRITE TABLE join_results SELECT songs.songname , albums.albumname, albums.numofsongs FROM albums JOIN songs ON (songs.artistid = albums.artistid);`

3. `Select * from join_results;`

Imagine a user creating a lot of tables to store the results of different join commands. It is not always possible to remember what each table is meant for. But, the touch client doesn't let the user get into such situation. A table *join_results* will be temporarily created to handle join operations. Once the user is done with join operation (the task), the table is dropped to avoid accumulation of tables that are created with every join operation. The command used is

```
DROP TABLE join_results;
```

As an extra layer of abstraction, touch client handles all these complexities of executing a query and it brings down the learning curve of the user.

4.7 Design Changes

The prototype implements all the features as discussed in the preceding chapter. However, the implementation digresses from the design in one aspect. Ideally, the back-end should have been HiveServer. Due to some problems faced (as discussed in section 4.5) during the implementation, the prototype simulates the Hive connectivity through a connection to MySQL taking the advantage of the similarity between MySQL and HiveQL. However, section 4.6 discusses the changes required for the prototype to support HiveServer as a back-end. The overall alteration required to handle the change of back-end to HiveServer is minimal. At a high level, diversion of the implementation of the design is minimal.

4.8 Effect of changes on user experience

As stated already, the back-end of the system is changed due to the technical problems faced. Though there are notable changes between SQL and HiveQL, it doesn't require a lot of alterations to the current implementation to handle HiveQL. While these minimal changes are totally related to back-end, the front-end requires absolutely no changes. Thus, the change in the back-end of the system has absolutely no effect for the user. So, the user doesn't know the change in the back-end and hence, needs no learning to get used to different back-end. This extra layer of abstraction offered by the touch based client makes it flexible and gives the advantage of keeping a uniform behavior irrespective of the back-end, thereby avoiding a situation where the user needs time to familiarize with the front-end every time there is a change in back-end.

4.9 Conclusion

As MySQL is very similar to HiveQL, the implementation of prototype begins with MySQL as the back-end. As 4 out of 5 features that are listed as a part of design are almost the same in both MySQL and HiveQL, the entire front-end was developed with MySQL as a back-end. Once the development of the front-end reached to a satisfactory level, the back-end was meant to be changed. When a change in back-end was attempted, a few minor issues were faced. Despite solving these minor issues, the attempt to change the back-end wasn't successful because of the problem which is already discussed in detail in section 4.5. Except for the change in back-end, which has a minimal impact on user experience, the prototype was implemented as per the design and as per the schedule. The implementation behaves exactly as per the design. The advantage of showing the operations as a menu as opposed to the way operations are handled by Gesture Query is that the touch client leaves no scope for confusion when a user chooses to perform a query. This lack of confusion makes the feedback system more effective. The design was aimed at making it easy for users to get familiar with the system easily which is the key to make the touch interface effective. The implementation achieves exactly that and the same is manifested by the results obtained from evaluation.

4.10 Summary

In this chapter, the overview of the functional architecture is explained. It also discusses how the implementation takes advantage of the technology of choice (Android) to support various screen sizes without having to implement everything specific to each screen size. Then, the hardwares used for the development are introduced. This is followed by explaining how to use the application. After that, the chapter discusses the reasons behind retaining MySQL server as the back-end in detail. It explains the problems faced in the process of changing the back-end. It emphasizes on how the change in design doesn't have an impact on the user experience. The chapter is concluded by analyzing how good the implementation is in comparison with the related work.

Chapter 5

Evaluation

This chapter has three sections. First section discusses the research methodology used for the evaluation. Second section discusses the results that obtained after the evaluation. This section describes the results obtained, but not the reasons behind the results and what they signify. Third section discusses in-depth the factors that contributed to the results and what they signify.

5.1 Research Methodology

As stated previously, this project aims at observing the effect of analyzing the data on a mobile device, as opposed to the conventional way of doing it through desktop (i.e. Command Line Interface). The research involved performing common tasks on both Command Line Interface (CLI) and touch client. Five databases related tasks were picked out for the volunteers to perform. The volunteers may or may not be from computer science background. The volunteers need not have expertise in MySQL but it required them to be familiar with it. All the volunteers of this research were those who are familiar with MySQL or experts in it.

The setup of the research had the touch client already connected to the MySQL Server. The volunteers were provided a concise overview of the system prior to usage and were also given a demonstration of how to use the touch client. Then, the volunteers were given some sample tasks to familiarize them with the touch client. Once the volunteers confirmed their familiarity, they were requested to fill the survey.

The survey had four sections

1. Background related questions
2. Tasks
3. General questions
4. Questions for System usability scale

At first, the volunteers answered the background related questions in the online survey. Subsequently, the list of tasks was displayed to the volunteer and were bespoken to perform each task on both CLI and touch client. The tasks were arranged in the increasing order of their complexities.

The survey called for the details such as time taken to perform each task and rate the ease of those tasks on a desktop and the touch client. The volunteers were assisted to calculate the time taken to perform the task so that they could focus on the task without any digression. The survey also called for the preference of a volunteer between keyboard based interface and touch based interface after the tasks were executed. The volunteers had the option to justify their preference if they wanted to. All the volunteers availed this option.

5.2 Results

The aim of this project was to compare the usability of the data query interface on a mobile (through touch client) against the conventional approach (such as command line interface). As stated previously in Chapter 4, five tasks were picked out for a volunteer to perform. Each volunteer was requested to record the time taken to perform the task and rate the ease of running the task on both command line interface and touch client. This chapter shows the results obtained in the form of graphs comparing the time taken to complete a task (image on the left) and the rating given by the volunteer for each task (image on the right) on mobile and desktop. Please note that CLI referred to by the following graphs stands for Command Line Interface and represents the desktop approach while touch client represents a mobile based approach. A scale from one to ten is used to rate the ease of tasks on both approaches where one signifies "very tough" and ten signifies "very easy"

5.2.1 Task1 : Display the list of tables

Task : Please select "Sample" database and display the list of tables

The results of this task show that the time taken in the case of desktop based approach was more than the time taken in the case of mobile based approach. This can be observed from the image on the left of Figure 5.1. For this task, this trend was consistent across all the volunteers. The image on the right of Figure 5.1 shows that performing this task on mobile based approach was easier compared to desktop based approach. Except for volunteer 3, the rest of the volunteers perceived the task to be easy on mobile and therefore rated 10.

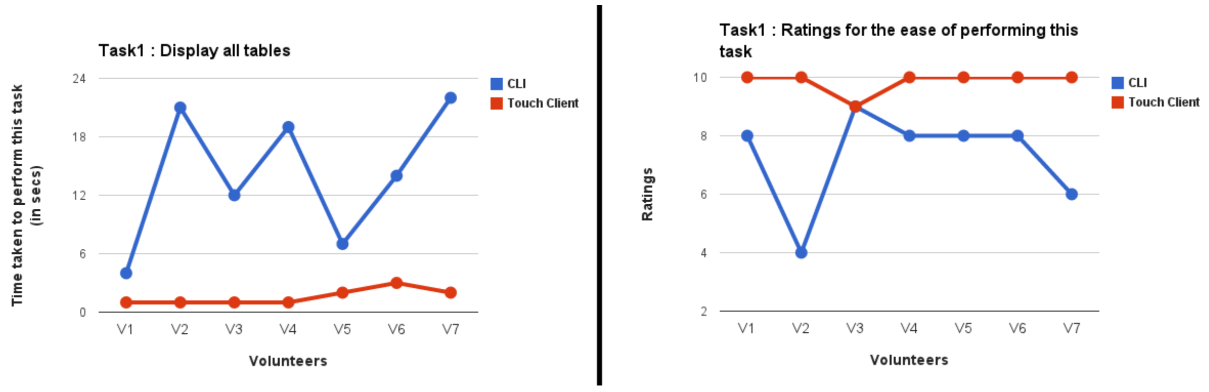


Figure 5.1: *To the left* : Comparison of time taken to perform Task1 *To the right*: Comparison of ratings given for the ease to perform Task1

5.2.2 Task2 : Fetch specific columns

Task : Fetch *songid*. *songname* and *albumid* from the table SONGS

The results of this task indicate that there was a difference between the time taken to perform this task on both approaches. The time taken in the case of mobile based approach is less. This can be observed from the image on the left of Figure 5.2. However, the difference in the time recorded was not as high as it was in the case of Task1. For any given volunteer, the time taken in the case of desktop based approach (on CLI) was greater than the time taken in the case of mobile based approach. The results also indicated that mobile based approach not only let the volunteer complete the task in lesser time, but also made it easy to execute the task. However, two volunteers (V3 and V6) experienced the same ease with respect to both approaches. This can be observed from the image on the right of Figure 5.2.

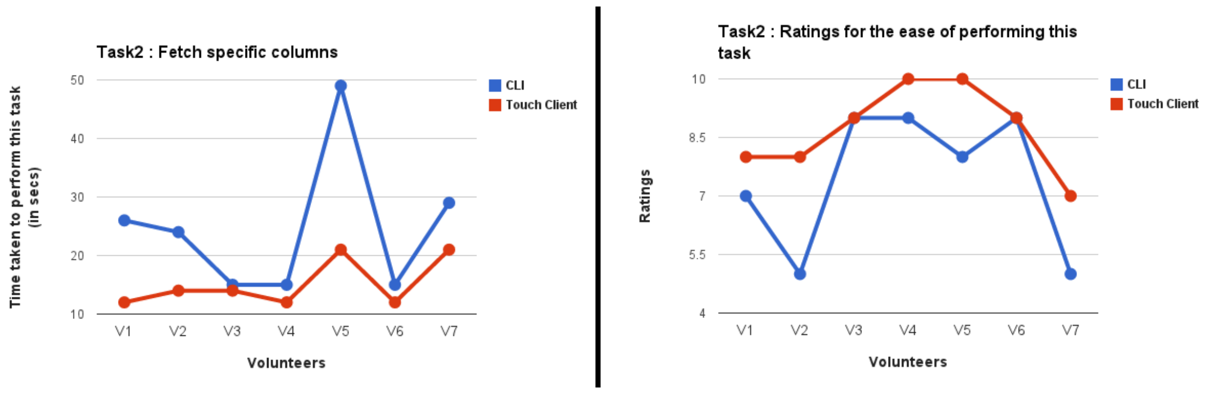


Figure 5.2: *To the left* : Comparison of time taken to perform Task2 *To the right*: Comparison of ratings given for the ease to perform Task2

5.2.3 Task3 : Filter operation

Task : Fetch all the records of the table SONGS from the album "Teenage Dream"

Unlike the preceding tasks which were straightforward, this task required the volunteer to filter the records based on albumid which could be obtained from a different table (ALBUMS). This extra step increased the complexity of the task by a bit. Though this task was executed differently by different volunteers, the results indicated that mobile based approach takes lesser time to complete the task. As one can observe from the image on the left of Figure 5.3, this aforesaid trend is not observed in the case of volunteers 5 and 6 who took less time in the case of desktop approach, yet both volunteers found that it was easy to perform the task on mobile based approach. This can be observed from the image on the right of Figure 5.3. The ratings of all the volunteers indicated that they found it easier to perform the task on mobile. The reasons for this anomaly will be discussed in the next chapter.

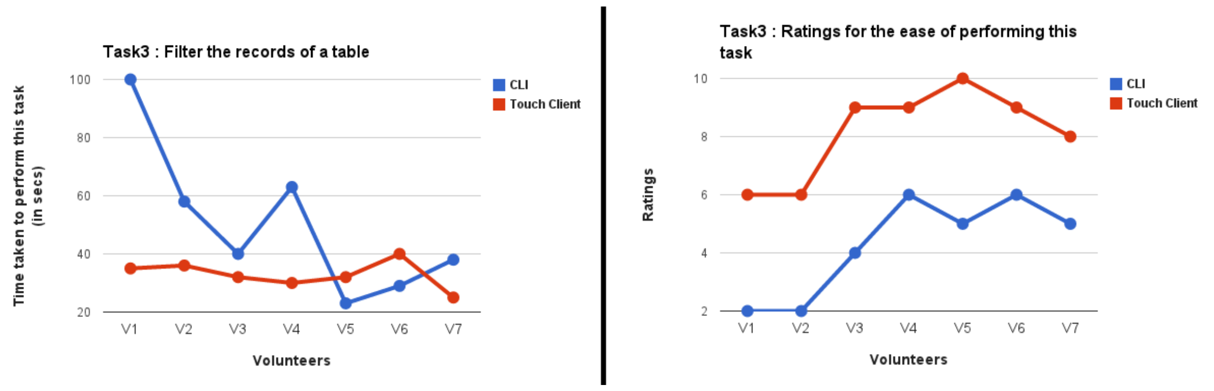


Figure 5.3: *To the left* : Comparison of time taken to perform Task3 *To the right*: Comparison of ratings given for the ease to perform Task3

5.2.4 Task4 : Union All operation

Task : Perform a union all of the following queries

- Fetch *albumtitles* of all the ALBUMS of Katy Perry
- Fetch *songnames* of all the SONGS of Katy Perry

This task was to perform a UNION ALL operation on both interfaces. The results indicate that the time taken to perform the task on mobile is lesser. This can be observed from the image on the left of Figure 5.4. In the case of volunteer 5, the time difference was no more than 3 seconds. Though the time taken in the case of desktop based approach was more in all cases, except one, there was a change in trend when it came to rating the ease of performing the tasks. This can be observed from the image on the right of Figure 5.4.

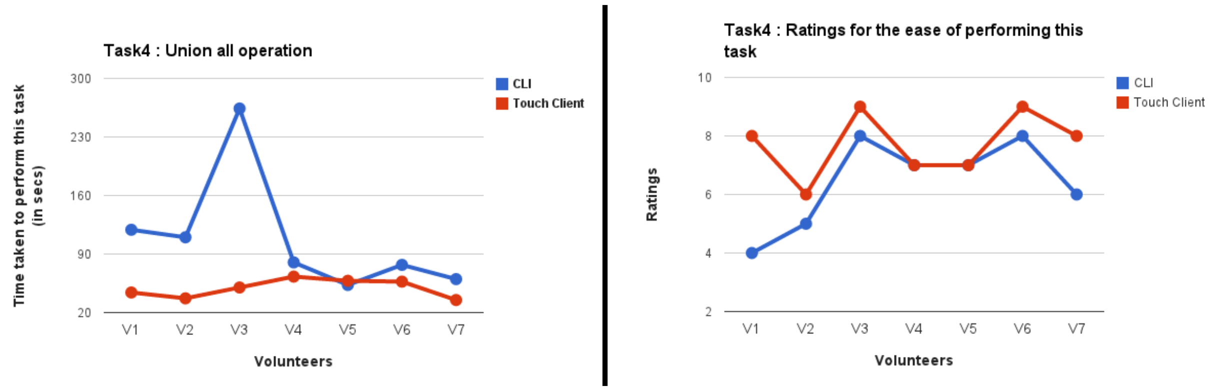


Figure 5.4: *To the left* : Comparison of time taken to perform Task4 *To the right*: Comparison of ratings given for the ease to perform Task4

5.2.5 Task5 : Join operation

Task : Execute a JOIN between tables ALBUMS and SONGS on artistId and fetch *songname*, *albumtitle*, *numofsongs*

This task was to perform a JOIN operation on both interfaces. The results indicate that the time taken to perform the task on mobile is less than the time taken on desktop consistently. This can be observed from the image on the left of Figure 5.5. Unlike the preceding task, this consistency in the trend is maintained even in the case of ratings for the ease of performing the tasks. This can be observed from the image on the right of Figure 5.5.

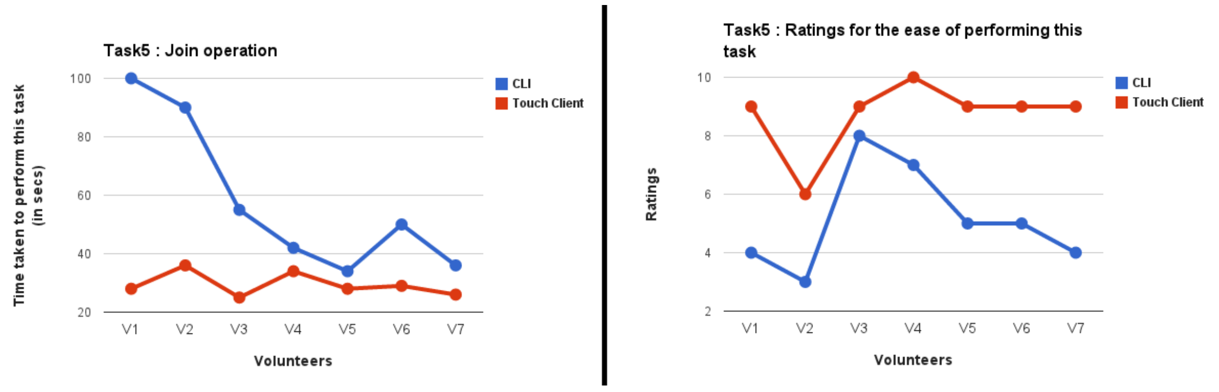


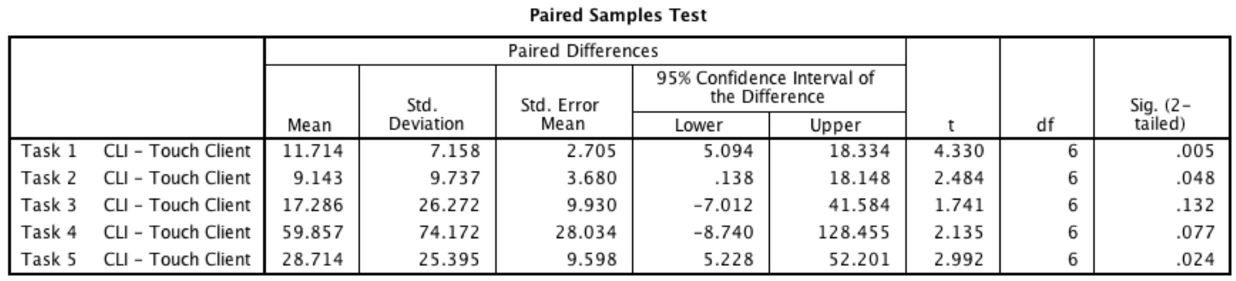
Figure 5.5: *To the left* : Comparison of time taken to perform Task5 *To the right*: Comparison of ratings given for the ease to perform Task5

5.2.6 Paired Sample T-Test

T-test for comparing the time taken to perform each task on CLI and the touch client

H0 *(Null Hypothesis)* : There is no significant difference between the means of time taken to perform the tasks on CLI in comparison with the time taken to perform the same on the touch interface

H1 *(Alternative Hypothesis)*: There is a significant difference between the means of time taken to perform the tasks on CLI in comparison with the time taken to perform the same on the touch interface



Paired Samples Test

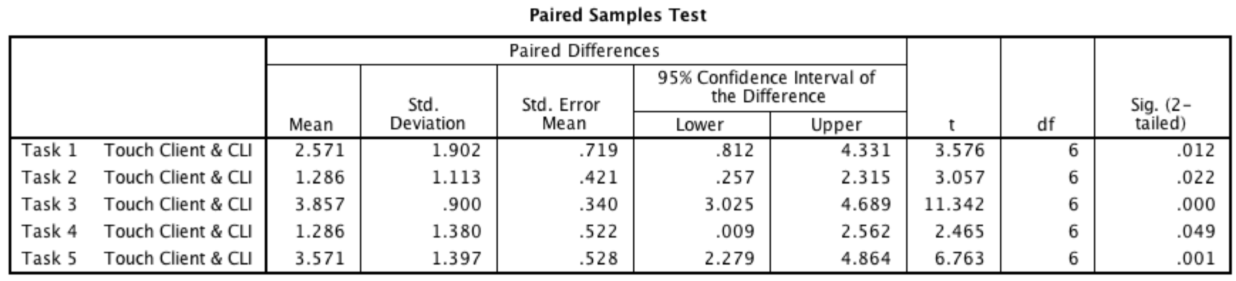
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Task 1	CLI - Touch Client	11.714	7.158	2.705	5.094	18.334	4.330	6	.005
Task 2	CLI - Touch Client	9.143	9.737	3.680	.138	18.148	2.484	6	.048
Task 3	CLI - Touch Client	17.286	26.272	9.930	-7.012	41.584	1.741	6	.132
Task 4	CLI - Touch Client	59.857	74.172	28.034	-8.740	128.455	2.135	6	.077
Task 5	CLI - Touch Client	28.714	25.395	9.598	5.228	52.201	2.992	6	.024

Figure 5.6: Paired sample t-test for the time taken to perform each task on CLI and the touch client

T values of all the tasks from the preceding table are > 0.0005. So, the results indicate that there is statistically significant reduction in the time taken to perform all the tasks on touch client as compared to performing them on CLI.

T-test for comparing the ratings given by volunteers to for the ease of performing each task on CLI and the touch client

H0 *(Null Hypothesis)* : There is no significant difference between the means of the ratings given by the user for the ease of performing each task on CLI in comparison with touch interface
H1 *(Alternative Hypothesis)*: There is a significant difference between the means of the ratings given by the user for the ease of performing each task on CLI in comparison with touch interface



Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Task 1	Touch Client & CLI	2.571	1.902	.719	.812	4.331	3.576	6	.012
Task 2	Touch Client & CLI	1.286	1.113	.421	.257	2.315	3.057	6	.022
Task 3	Touch Client & CLI	3.857	.900	.340	3.025	4.689	11.342	6	.000
Task 4	Touch Client & CLI	1.286	1.380	.522	.009	2.562	2.465	6	.049
Task 5	Touch Client & CLI	3.571	1.397	.528	2.279	4.864	6.763	6	.001

Figure 5.7: Paired sample t-test for the ratings given by the volunteers for the ease of performing each task on CLI and the touch client

So, the results indicate that there is statistically significant improvement in the experience of executing queries on touch client. The volunteers found it easier to perform the tasks on the tough client irrespective of the time taken to execute the tasks.

5.3 Discussion

A console based query interface is not so easy to use for non-experts or a new user, especially in circumstances where query language, the schema, structure of tables, underlying data etc. are unfamiliar. The problem is that there is no significant measure of work towards making databases of query languages like Hive more usable to end users. Though the query language has a console based interface, it is a keyboard and mouse based and provides no feedback to help guide the user towards the intended query. This thesis proposes the idea of replacing keyboard based client (or interfaces) with a touch based client that enables users to analyze big data on a mobile assisted by a feedback system that helps the user build a query.

As a part of this research, the implementation of the touch client was done for an Android device with database server as a back-end. The touch client was connected to the back-end using JDBC. As a technical problem was faced while establishing the JDBC to HiveServer, the database server used for this research was a MySQL server as SQL is very similar to HiveQL. Subsequently, to assess the usability of the system, a prelim user study was done by comparing the performance of touch client against the command line interface which is a conventional way. For this study, five tasks were picked for volunteers to perform. The tasks were executed in the order of their complexity and their results are already published in the preceding chapter.

The results of the user study indicate the preference of the volunteers for touch client for its visualization of data, low learning curve and its ease to switch from one task to another. For all the tasks, the rating for the ease of performing the task on touch client is either better than or same as command line interface.

5.3.1 Task1 : Display the list of tables

This task required a volunteer to select a database and display all the tables in the database. This requires just one click on a database in the case of touch client, whereas it requires two commands in the case of command line interface. Upon clicking a database, the touch client by default displays all the tables in the database. The commands to be executed on command line interface are

1. `use sample;`

2. `show tables;`

Figure 5.1 shows that the time taken for this task in the case of command line interface ranges from 4 secs to 21 secs across different volunteers. This shows how syntax errors can make a simple task take longer time than it should. Touch client avoids all this inconvenience as it doesn't require a volunteer to type anything. This difference in the ease of performing this task on both interfaces is reflected in the image on the right of Figure 5.1, where the graphs show that all the volunteers, except one, rated 10 for the ease of performing this task on touch client.

5.3.2 Task2 : Fetch specific columns

This task required a volunteer to fetch and display the specific set of columns of a table. This required 3 steps in the case of touch client and they are

- Select a table
- Filter the table for columns
- Click “OK” on the dialog

However, it requires only one command in the case of command line interface. The command to be executed on command line interface was

```
select songid, songname and albumid from songs;
```

Despite having more number of steps in the case of touch client, image on the left of Figure 5.2 shows that it took more time to perform the task on command line interface. The reasons that contributed to the time taken in the case of command line interface are not just syntax errors in commands but typing errors in the column names as well. The lack of such errors in the case of touch client ensures a smooth flow, thereby making it easy to execute the tasks. The ratings for the ease of performing this task also reverberate the same.

5.3.3 Task3 : Filter operation

Unlike the preceding tasks which were straightforward, this task required a volunteer to filter the records of an album “Teenage Dream”, whose albumid could be obtained from a different table. This extra step increased the complexity of the task by a bit. It is important to note that this task can be performed in different ways on command line interface.

1. `select albumid, albumtitle from ALBUMS;`

Fetch albumid and then execute

```
select * from SONGS where albumid=<obtained albumid>;
```

2.
```
select albumid from ALBUMS where albumtitle = "Teenage Dream";
```

   fetch albumid and execute

```
select * from SONGS where albumid=<obtained albumid>;
```

3.
```
select * from SONGS where albumid=(select albumid from ALBUMS where
    albumtitle = "Teenage Dream");
```

Volunteers executed this task in different ways. Despite having different options, most of the volunteers took lesser time to perform the task on mobile. As this task was the first one where the volunteer had to filter the records based on the column values, two volunteers took more time to execute the task on touch client due to the confusion caused by the lack of experience. This can be observed in the image on the left of Figure 5.3. However, the ratings of the same volunteers indicate that they found it easier to execute the tasks on touch client. The same can be observed in the image on the right of Figure 5.3. This indicates that the time taken by the two volunteers will be lesser if they were to perform the task again. This nevertheless doesn't signify a weakness in the system because the same **Filter** operation has to be performed twice to execute the next task i.e. Union all operation. The same volunteers faced no problem while executing the Union all operation. Figure 5.4 reveals that the volunteers have taken less time performing the task on the touch client, thereby proving that the discomfort observed in the case of two volunteers for this task was a one-time occurrence.

5.3.4 Task4 : Union All operation

This task was to perform a UNION ALL operation on both interfaces. Like the preceding task, the two queries to be combined were not direct. They required the volunteer to know the *artistid* of the artist whose *albumtitle*s from ALBUMS and *songname*s from SONGS should be fetched.

This task also could be performed in different ways. One such option is using subquery as shown in the third example of the previous task. Most of the volunteers took more time to perform the task on command line interface. Recalling the syntax of the query actually contributed to the time taken to complete the task in this case. The image on the right of Figure 5.4 indicates that all volunteers took almost the same time to perform the task on mobile and red line in the graph reflects this. However, the case is not same with command line interface. The blue line in the graph indicates a sudden rise in the time taken by volunteer 3. In this case, though the subqueries were used to perform the task, due to various syntax mistakes in command and typing errors in column names, the volunteer took around 264 seconds to complete the task. This shows how known tasks can take more time because of errors. This can be avoided if a volunteer is always assisted by a feedback system. The image on the right of Figure 5.4 indicates that it is either easier or just as easy to perform this task on mobile as compared to performing the same task on command line interface.

5.3.5 Task5 : Join operation

This task was to perform a JOIN operation on both interfaces. All the volunteers took more time to perform the task on command line interface. There is a notable and constant decrease in the time taken to perform the task on command line interface from volunteer 1 to volunteer 5. This is mainly attributed to recalling the syntax of the command. After volunteer 1, other volunteers had to go through the syntax on the internet before they could perform this task. Inspite of this, a considerable amount of time was taken to perform this task on command line interface by each volunteer. Factors like recalling syntax, avoiding typing errors in syntax, and in column names have always contributed to the discomfort of a volunteer in executing the task.
Overall, the factors that influenced the experience of the volunteers are

- Typographical errors in column names and syntax on command line interface
- Recalling syntax of the command to execute the query on command line interface
- Usage of wrong syntax of the command on command line interface
- Presentation of the data on touch client

The feedback system makes sense, especially in the case of complex queries like UNION ALL and JOIN where the user has the basic knowledge of MySQL and yet doesn't recall the syntax at will. The presentation of data backed by the feedback system made it easy for the volunteers to execute the tasks on the touch client.

Despite having basic knowledge of MySQL and years of experience of command line interface, the volunteers found it easier to perform the tasks on touch client. Though HiveQL is very similar to MySQL, imagine the kind of annoyance switching to HiveQL can cause given the slight changes to the rules of its syntax. This research makes more sense in the case of HiveQL.

5.4 Summary

This chapter comprises three sections. The first section discusses the methodology that was used to evaluate the performance of touch based interface against the command line interface. It explains the complete procedure followed to obtain the results. The second section displays the results obtained from the evaluation. The results are presented in a graphical form for a better understanding. This section describes the results, but not the reasons behind those results. The third section discusses the results of every task on both interfaces and the reasons that contributed to those results. It also discusses what those results meant and what they signify. This section also discusses the results of paired sample t-tests and its significance.

Chapter 6

Conclusion and Future Work

This chapter comprises of four sections. Firstly, it demonstrates how the research objectives have been met and challenges have been overcome. It addresses the extent to which the research objectives (described in Chapter one) have been met. Secondly, it summarizes the contribution of this research. Thirdly, it discusses general conclusion of this thesis. Finally, it discusses the future work of the research.

6.1 Objectives Assessment

6.1.1 Lower the Learning Curve

Research Objective : Lower the learning curve

The challenge to achieve this objective was to keep the UI simple to lower the learning curve. Gesture Query highly depends on gestures like pinch in, pinch out, two finger swipe etc. to identify the operation. The speed at which the gesture occurs has to be correctly interpreted. The gesture may not always be intended. For example, the user may want to swipe with a single finger and accidentally had the second finger touch the interface during the swipe. This will lead the gesture engine to interpret this gesture incorrectly as a two finger swipe.

This challenge was overcome by keeping the touch interface simple with the support of a feedback system to help user perform a query without any confusion. Unlike Gesture Query, touch client handles all the operations in the form of a menu written in text. This lets the user select the query without having to remember the mapping of gestures and operations. This avoids the scope for errors in recognizing the intended operation. As the touch recognizes the intended operation, the feedback system always provides the correct prompting.

6.1.2 Easy Switching

Research Objective : Make it easy to switch from technologies like MySQL to HiveQL

The challenge was to make an easy switch from MySQL to HiveQL. The Section 4.8 addresses how the challenge was overcome. The system of the prototype is divided into three layers. UI layer, Task layer and Database Server. Any change to the UI layer impacts the user as it mandates the user to be aware of the changes to the UI. The prototype for this research is developed in such a way that the UI layer takes the results from the task layer and displays it to the user. It doesn't interact with the database server directly. So, any change in the database server technology will be handled by the task layer, thereby avoiding any changes to the UI layer. As the UI doesn't change with the change in back-end, the user who familiarizes with touch interface once need not learn anything new for a different back-end.

6.1.3 Identification of Tasks for Evaluation

Research Objective : Identifying key tasks for the evaluation of the touch interface

The challenge to achieve this objective was to identify the tasks. This challenge was overcome with the help of Chapter Two. It began by reviewing the related work such as Gesture Query and dbTouch. With dbTouch being very complex to conceptualize, Gesture Query was studied thoroughly to gauge its advantages and challenges. Gesture Query considered two complex tasks like Union all and Join operations for its evaluation. With these tasks already being accepted by an International Conference, they have been added to the list of tasks considered for the evaluation of this research. So, the related work was taken into consideration to identify key tasks.

6.1.4 Evaluation of the Prototype

Research Objective : Evaluating the performance of the analysis on touch client (mobile based) against desktop (keyboard based) to get the right assessment of how effective touch based interface is

The challenge was to chalk out a plan to evaluate the performance of the prototype against the conventional interfaces. The final list of five tasks comprised three simple tasks and two complex tasks. It was important to have a right mix of complexities to get a better understanding of how the touch client performs in comparison with command line interface while executing simple and complex tasks. To get a better understanding of performance, each task was executed on both interfaces and parameters such as user rating for the ease of performing a task on each interface and time taken to perform a task on each interface are recorded. In this manner, the performance of the touch client is evaluated against the command line interface.

6.1.5 Visualizing Results on Mobile Device

Research Objective : Visualizing the analysis results on a mobile device

One challenge was to display the records in a very understandable way. This challenge was overcome by displaying the results on the touch interface in a tabular form. This helps the user to correctly visualize the records of a database. The command line interface of HiveQL doesn't display the records along with their column names. Just by looking at the results displayed on the console, one cannot relate with the information.

The Second challenge was the execution of the concept that helped overcome the first challenge. This concept was successfully implemented by achieving the aforementioned objectives.

6.2 Summary of Contribution

The contributions of this research are as follows

- A detailed literature review is provided about

    1. The query tools available in the field of Hadoop
    2. Commonly used mobile platforms for the development of touch based applications

3. The work that is done in the field of databases in an attempt to make a touch based interface more usable for querying databases

- A modular implementation of touch based interface to the databases in general. With a customized implementation of task layer, the touch interface can be adapted to other query languages. This ensures the ease of the user to switch from one query language to another
- An insight into the possible problems a user faces while executing queries on a conventional interface, despite being familiar with it for years. It also manifests that this long familiarity may still not help a user recall the syntax of the queries. It also shows how often users commit typing mistakes, even while executing the simplest of the queries
- Finally, the results from the evaluation demonstrate that the touch based interface is more usable in comparison with a keyboard based conventional interface. It also showcases that the touch based interface is more effective in the case of complex queries as it helps the user visualize the querying that is going to be executed, unlike the case of command line interface where the user executes the query and then realizes the mistakes in the query when a syntax error is displayed

6.3 General Conclusion

This research makes an attempt to explore the role of mobile devices in the context of big data. The aim of this project is to compare the usability of the data query interface on a mobile (through touch client) against the conventional approach (such as command line interface). As a part of this attempt, this research implemented a prototype of touch client on an Android device. This touch client allows the user to forge queries more easily and quickly than conventional interfaces such as command line interface while providing a constant feedback system. The client is easy to use as it doesn't require a user to type queries, thereby avoiding unnecessary inconvenience while accessing data. This usability of the system is manifested by SUS score of 79.375. In spite of getting used to command line interface for years, this prototype manifests the improvement in the experience of a user in querying the MySQL server. This research also shows how the same client can handle HiveQL with minor changes to the already existing implementation thus demonstrating how touch based client can be used as front-end to different database servers.

This research simulates Hive connection with MySQL connection. The Hive server supports only the command line interface and not the graphical user interface. Despite having the graphical user interface, the touch client is compared to the command line interface of MySQL.

6.4 Future Work

Our work has been to put together a prototype for demonstrating the possibility of replacing the keyboard based client with touch based client. To make this prototype more effective, improvements would have to be made in a number of areas

- **Search**
  A search feature can be added so that the user can fetch the row based on a search criteria. For example, if the user wants to fetch an albumid based on the album name, the user need not scroll through hundreds of records to locate the appropriate record.

- **Paging**
  If the fetched number of records is very large, the records can be displayed on pages with a set number of records per page. This avoids time and memory consumption while loading a very large number of records as it doesn't require to fetch all the records at the same time.

- **Support more data types**
  The prototype only handles simple data types. But, Hive supports complex types like Maps, Struct, and Arrays. This avoids the users from using very complex commands. While SQL doesn't support these kind of data types, it doesn't require to say that the users will need assistance here.

- **Write based operations**
  This prototype focuses more on the read based operations. This should be extended to write based operations as well. The interface should be extended a way which shows all the files on the HDFS. A file manager like application should be developed that lets user create a table and populate it from a file which would not be possible without using load command.

Bibliography

[1] J. Heggestuen. (2013, Dec.) Big data will drive the next phase of innovation in mobile computing. [Online]. Available: http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

[2] T. Culture-ist. (2013, May) More than 2 billion people use the internet. [Online]. Available: http://www.thecultureist.com/2013/05/09/how-many-people-use-the-internet-more-than-2-billion-infographic/

[3] L. Mearian. (2011) World's data will grow by 50x in next decade, idc study predicts. [Online]. Available: http://www.computerworld.com/s/article/9217988/

[4] R. K. B.Gerhardt, K. Griffin, ``Unlocking value in the fragmented world of big data analytics,'' 2012.

[5] C. Tankard, ``Big data security,'' *Network security*, vol. 2012, no. 7, pp. 5--8, 2012.

[6] S. Singh and N. Singh, ``Big data analytics,'' in *Communication, Information & Computing Technology (ICCICT), 2012 International Conference on*. IEEE, 2012, pp. 1--4.

[7] S. Sagiroglu and D. Sinanc, ``Big data: A review,'' in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. IEEE, 2013, pp. 42--47.

[8] [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/Tutorial#Tutorial-HiveTutorial

[9] S. Chen, ``Cheetah: a high performance, custom data warehouse on top of mapreduce,'' *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1459--1468, 2010.

[10] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Eltabakh, C.-C. Kanne, F. Ozcan, and E. J. Shekita, ``Jaql: A scripting language for large scale semistructured data analysis,'' in *Proceedings of VLDB Conference*, 2011.

[11] C. Sauer and T. Haerder, ``Compilation of query languages into mapreduce,'' *Datenbank-Spektrum*, vol. 13, no. 1, pp. 5--15, 2013.

[12] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, ``Building a high-level dataflow system on top of map-reduce: the pig experience,'' *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1414--1425, 2009.

[13] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, ``Hive-a petabyte scale data warehouse using hadoop,'' in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 996--1005.

[14] R. J. Stewart, P. W. Trinder, and H.-W. Loidl, ``Comparing high level mapreduce query languages,'' in *Advanced Parallel Processing Technologies*. Springer, 2011, pp. 58--72.

[15] A. Gates. (2010, Aug.) Pig and hive at yahoo! [Online]. Available: http://developer.yahoo.com/blogs/hadoop/pig-hive-yahoo-464.html

[16] D. Gavalas and D. Economou, ``Development platforms for mobile applications: Status and trends,'' *Software, IEEE*, vol. 28, no. 1, pp. 77--86, 2011.

[17] E. Liarou and S. Idreos, ``dbtouch in action: Database kernels for touch-based data exploration,'' in *Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE)*, 2014.

[18] L. Jiang, M. Mandel, and A. Nandi, ``Gesturequery: a multitouch database query interface,'' *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1342--1345, 2013.