HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

# “Relational Data Access on Big Data”

## Bachelor Thesis

Department of Computer Science
University of Applied Science Rapperswil

Fall Term 2013

| | |
|---|---|
| Authors: | Christof Büchi, Susanne Mathys |
| Advisor: | Prof. Josef M. Joller, HSR |
| External Co-Examiner: | Romeo Kienzler, IBM Innovation Center Zurich |
| Internal Co-Examiner: | Prof. Hansjörg Huser, HSR |

# Abstract

Big Data is an expanding topic in information technology based on the huge amounts of data which are generated by IT systems. The ability to store structured and unstructured data at petabyte scale will improve the efforts of business intelligence and widely expand the type of questions which can be answered with these systems. To address changes in data management for large data warehouses (DWH) several new products are released on top of the Hadoop ecosystem. We focus on storing data for business analytics concerned questions and running queries against it. Where the Hadoop distributed filesystem (HDFS) has its strengths at parallel processing and reading files in a sequential access pattern, we are interested in getting answers to business related questions based on aggregations. Answering these questions calls for range queries accessing subsets of data. The problem of dealing with files in Hadoop needs knowledge of the Map-Reduce principle and different languages are used to access the data. Enterprises have to invest time and money in new systems and know-how for Big Data systems. A new way is to provide the well known ANSI SQL standard for querying data on Hadoop platforms. Already done investment on the analytics side could be reused on the new platform.

We have analyzed the capabilities of IBM BigSQL. To compare performance and semantic differences between IBM BigSQL and a traditional relational DWH solution, we built up a distributed IBM DB2 cluster based on IBM Database Partitioning Feature (DPF) technology and used an IBM BigInsights cluster to execute analytical queries. We have selected the TPC-H benchmark simulating a typical OLAP workload as use case. We have measured performance of query execution as well as scaling out characteristics.

The results show us that in some cases processing an analytical query on a Big Data platform could be more efficient than on relational DWH systems. The Hadoop ecosystem owns lot of potential but brings along lot of drawbacks as well. But with these in mind, a cost-efficient DWH can be established. Beside the established traditional way many hadoop-based solutions are in development to avoid the negative aspects of the Map-Reduce principle. After our research and measurements during this thesis we came to the conclusion that all of the investigated products could be used in a production environment. A recommendation which product could be used for analytical queries depends of many different factors like costs, query execution time and diversity.

# Declaration

We, Christof Büchi and Susanne Mathys declare,
that the work we present here in form of this bachelor thesis,
is completely developed and written by ourselves.
Any external sources or means are cited as such.
This bachelor thesis was not used in any other
academic grading context or is being published elsewhere.

Rapperswil, December 19, 2013

Christof Büchi

Susanne Mathys

# Acknowledgement

We, the authors thank ...

... Romeo Kienzler for inspiring and supporting us during this thesis and offer us an helping hand.

... Prof. Dr. Josef M. Joller for giving us an open scope on this work.

... Jürgen Götz for discussing with us about performance aspects of IBM DB2 DPF.

... Simon Harris for providing us performance related information about IBM BigSQL and Apache Hive.

... IBM Switzerland and Patrick Cadonau, which provide us the technical cluster environment and the offering to host meetings and presentation.

... to all cross readers of this thesis, which had shared time and expertise.

# Contents

# 1. Management Summary

## 1.1. Problem Definition

In this thesis we want to point out the semantic differences between processing Big Data with traditional relational database management system (RDBMS) and hadoop-based NoSQL datastores. On one side we will build a distributed IBM DB2 cluster, on the other side we will use a Hadoop cluster based on IBM BigSQL and the Apache Hive database. Predefined analytical queries will be executed on both systems to measure and compare the performance of those two different architecture systems. We want to take a look at the IBM BigSQL solution and if it suits the actual business requirements in data warehouse questions.

## 1.2. Solution Proposal

The traditional relational part of our comparison will be a distributed IBM DB2 cluster. Based on the preconditions we have to choose which product/technology best matches our requirements. We decide between building an IBM Database Partitioning Feature (DPF) or IBM pureScale cluster. The Big Data part of our work is covered by Apache Hive and IBM BigSQL on an IBM BigInsights cluster.

## 1.3. Sample Use case

One of the available TPC benchmarks has to be chosen and implemented on our system. Our problem domain will be analytics of data for Business Intelligence (BI) purposes.

## 1.4. Experiments

Based on the given use case both clusters have to be adjusted for best performance. Performance will be evaluated by processing time of the given queries. We do not focus on time for building up the database or load the tables. Experiments will be done by scaling out from 5 to 10 nodes.

## 1.5. Results

Summarized the IBM DB2 cluster delivers the fastest processing times. Although Hive is designed for scaling out to hundreds of nodes, in our case - with using only 10 nodes - query execution times are admissible for this workload. Compared to the other two systems IBM BigSQL has no immense outliers in our measurement series. With IBM BigSQL, IBM shows the possible capabilities of the Hadoop platform. A recommendation which system should be used for analytical queries depends of different factors, for example costs and diversity of workload.

## 1.6. Future Work

After gathering insights to execute Big Data analytic concerned queries in a Big Data system, a future project could be to execute a real world use case with a Business Intelligence software like IBM Cognos BI and compare it with its existing relational solution. Another extension to our work would be the measurements with Presto or its competitors, like Cloudera Impala.

# 2. Introduction

Information is today one of the most essential values of an enterprise as business decisions are mostly based on information [29]. Time to market and innovation of new products are nowadays the key factors for enterprises. Data warehouses and BI support the process of making business decisions. These instruments allow the discovery of hidden pattern like asking unknown relations between certain facts or entities. This causes an important change: in the past, the question which has been run against the system was already known at the time of collecting the data, today it is common practice to catch all the data to hold it for questions which will be asked in the future [28]. That is the reason why Big Data is a hot growing topic in information science.

Databases - which are the fundamental of data warehouses - adapt these new business requirements, which is a challenge for a system that was invented decades ago. The systems reach their limits, because the increase of data is enormous. As an additional difficulty beside the rising of data, systems are more and more complex to handle as long as workload is distributed across multiple servers on clusters. Vast amount of data is actually handled by database vendors in tree different approaches:

– First traditional relational databases are tuned to scale out with partitioning of tables and support for some semi structured data types such as XML or JSON [8] [34] [32] [31].

– Second in-memory databases like SAP HANA [1] are one of the leading database products on the market. They provide faster query processing times.

– Third way is to store and process data on a Hadoop system [38].

Enterprises which already hold lots of data, like search engines and social networks, running queries in fast execution times over their huge collection of data with hadoop-based technologies. Hadoop allows data distribution and parallelized data access in an efficient way. Disadvantages are the necessity of learning the new concept and attaining experience during real-world tests. That is why many enterprises show much respect to this new approach.

Recently providing SQL ANSI query language for data access on the Hadoop platform is getting more popular. IBM BigSQL brings the benefit of executing SQL queries on their Big Data platform IBM BigInsights.

We analyze the ability of IBM BigSQL. In addition, we want to point out the semantic differences of it and a traditional relational data warehouse solution. Performance measurements are done by scale out a workload on a distributed IBM DB2 system and on an IBM BigInsights cluster. With this work, we want to provide recommendations about solving a specific use case. For a common use case, we selected the TPC-H benchmark which is known as very stressful, but realistic [33]. During our study we want to explore the benefits and disadvantages of the less known Big Data system and describe what should be considered while planning to implement such a system.

# 3. System and Methods

Our study is divided into two parts: working on an relational database system and using a Hadoop system. In this section, we first give an overview about basic terms. As second, we present all products and systems for building up our IBM DB2 distributed cluster. Third, we give an introduction into the Hadoop ecosystem and used products for our Hadoop cluster. The last part of this chapter explains the TPC-H benchmark environment.

### 3.0.1. Term Definition

**Shared nothing Architecture**

It is a common idea that a database server has local storage, which can be structured best for needed requirements. Also in an usual distributed database environment, a server has an exclusive access to its storage, in form of local harddisks. This architecture is represented in our use case by IBM DB2 DPF.

**Shared Disk Architecture**

The shared disk architecture is formerly known as clustered file system. A clustered filesystem can be mounted on multiple servers, while all of those can access the disk simultaneously. If a server need access to data which is not at his local storage, network traffic is generated to access the data from one of the other nodes in the clustered filesystem, which have a lot impact on the disk access time. For relational access on data, the disk access time is the most important factor.

**Schema on write**

With schema on write, which is used in relational DBMS, at table creation time the structure of data in the table is defined. The RDBMS writes the data in that specified structure. Consistency is verified at the moment the DBMS writes records. When data has to be read out of the tables, the structure is already defined and consistency is guaranteed. This approach performs best at relationships in complex data warehouses, like multi dimension clustering. It is also very space efficient.

**Schema on read**

When schema on read is used, data is written in records and often records with a separator character are used. At writing time the DBMS simply writes down the records and does not verify the data against a defined structure. But when data is read, all the information about the structure of the records is needed. The schema of a table is used while reading the records from that table, values are verified only at reading time. While the schema on write performs best, the schema on read approach scales best, and provides the most possible flexibility. New generated data can be directly written into the filesystem - the availability of new data is in realtime, while querying and building aggregations on unstructured or loosely bound data generates much more I\O.

## 3.1. IBM DB2

IBM DB2 is a well established RDBMS. It is deployed in many reliable environments, at many large enterprises. IBM DB2 is a leading product from IBM, initially built a few decades ago. To support todays needs, IBM DB2 has support for object relational features and non-relational data structures. In the latest implementation, IBM provides a feature namely "BLU Acceleration" which uses a NoSQL-Strategy (column-store) and In-Memory technology behind the scenes. With this functionality IBM DB2 BLU offers a relational DBMS with the benefits of a column store. This can be defined on table level within the same database. At the moment, this feature is not supporting multi node configurations, which is the reason it is not comparable to the Hadoop system.
Since the data will be bigger and bigger, IBM DB2 has to support distributed strategies.
We provide a short overview of possibilities to operate with heavy cpu or I\O-intensive workloads in a distributed system.

### 3.1.1. IBM DB2 DPF

IBM DB2 uses a shared-nothing architecture within the Database Partitioning Feature (DPF). DPF allows you to set up a cluster of multiple nodes, each of it stores a partition of the database. Distribution of tables is provided with a distribution key (a column you are specifying). Rows of a table are horizontally partitioned over a hashing function of the specified distribution key column. Within the database, different partition groups can be used to spread tables across different nodes. Adding or removing nodes with redistribution of data is provided by IBM DB2 commands, which is really straight forward. The following figure 3.1 shows the architecture of an IBM DB2 DPF environment:

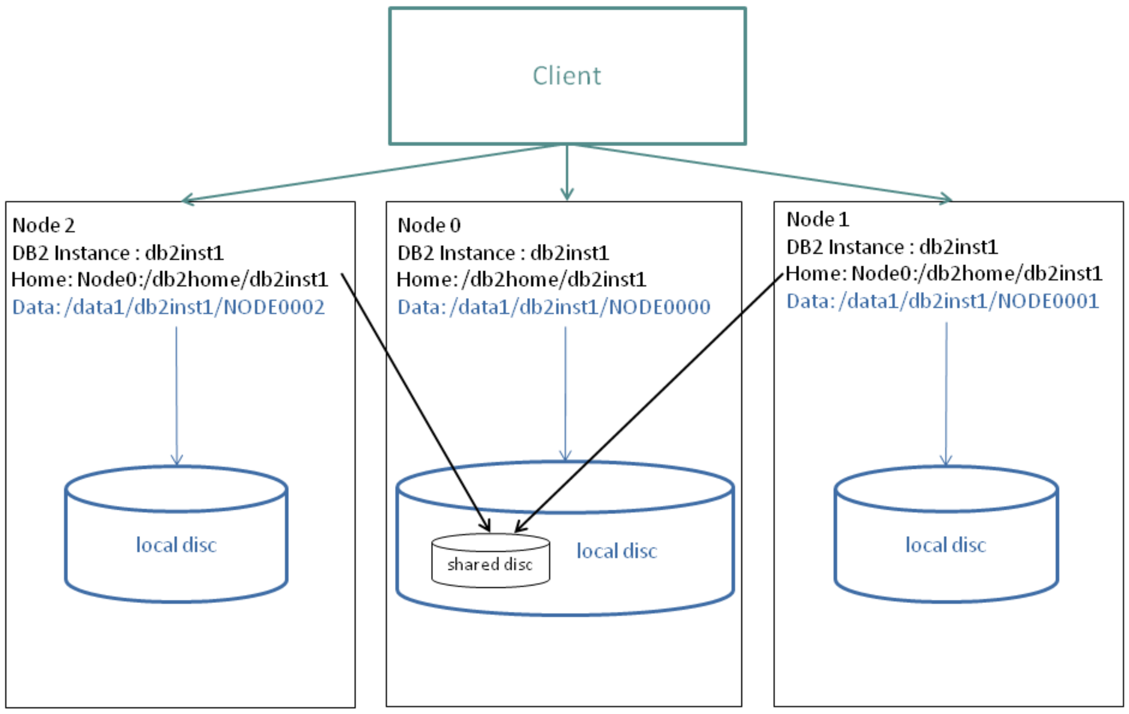


Figure 3.1.: IBM DB2 DPF

### 3.1.2. IBM DB2 MDC

Multi Dimension Clustering (MDC) [5] is used to store rows with same values over different dimensions physically together - better known as storing the data in OLAP cubes. This method will reduce I\O cost for analytical queries. MDC provides performance improvement, when all the queries and the whole database are created and designed for data warehouse workloads. However, when scaling out such a system the performance does not really benefit from the additional computing resources added to the cluster.

### 3.1.3. IBM DB2 PureScale

IBM DB2 pureScale [6] uses a shared-storage architecture (similar to Hadoop) and presents a fault-tolerance system. Every single node in the pureScale cluster has a connection to services of the pureScale server. The high available (HA) pureScale server acts as a master node and controls requests from clients, manages locking and stores a centralized cache. The member nodes can read and write the database in shared-storage mode. The pureScale server stores information about which pages are processed by a member node and stores dirty committed pages from that member in its centralized buffer pool. Fail-over of one member node does not impact the other members and the failed node can be recovered from the data of the buffer pool. IBM DB2 pureScale is a solution for a scale-out high available database system which points more to OLTP workloads.

The following figure 3.2 illustrates the architecture of an IBM pureScale environment:

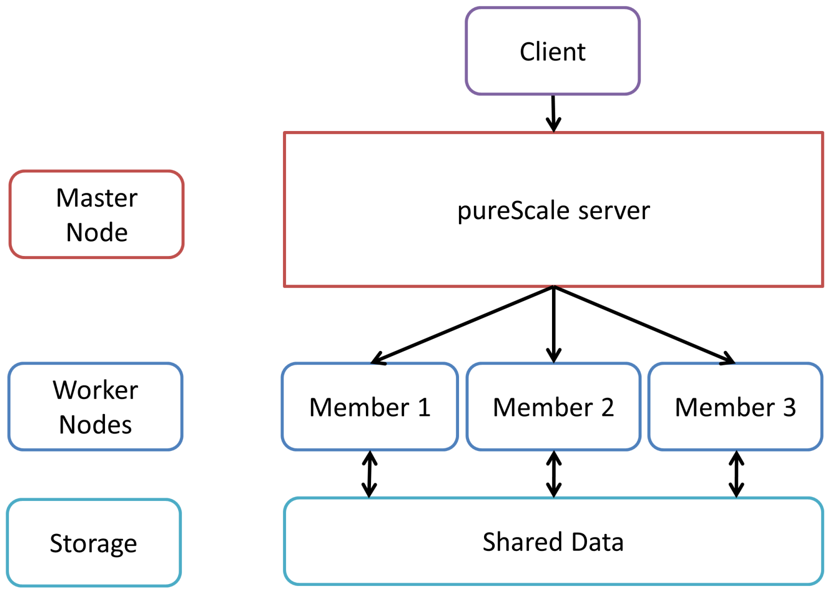


Figure 3.2.: IBM DB2 pureScale

### Selected Distribution Strategy

In the context of the TPC-H use case, facing the need to provide a realistic OLAP database system we decided to use a IBM DB2 DPF cluster. We neglected IBM pureScale and MDC to achieve a scalable distributed database environment with an architecture that is similar to the Hadoop ecosystem. IBM pureScale uses only one host per query which means it cannot execute one query over different nodes in parallel manner. IBM DB2 DPF uses all available computation resources for one query at a time. By choosing IBM DB2 DPF we are accepting that our IBM DB2 system is not fault tolerant.

## 3.2. Hadoop

Hadoop is the open-source implemenation of the Map-Reduce framework. There are other products behind this topic, but none of them represent the notion of Big Data as much as Hadoop. In the following paragraph we give an overview about the framework and its ecosystem and describe why Hadoop is important in this topic.
Hadoop allows to split an amount of work into many pieces and enables to send this pieces of work to worker units. Those worker units could be very primitive computation engines based on cheap commodity hardware. In an ideal situation they have some direct-attached storage to minimize the network bottleneck.
Usually there exists one jobtracker and one namenode. The jobtracker receives the job as a big amount of work and split it to many tasks. On the same node, there should be also a namenode installed. The namenode holds a list of all available datanodes, and which file is located where, which also includes replication of files. In summary, they know which worker has which computation

resources and which files are located on that specific worker. Computation resources are measured in map and reduce capacity. With that information, the task could be sent to the node with the local data, which avoids unnecessary network traffic and valuable task setup time.
The following figure 3.3 illustrates the architecture of Hadoop and its ecosystem.

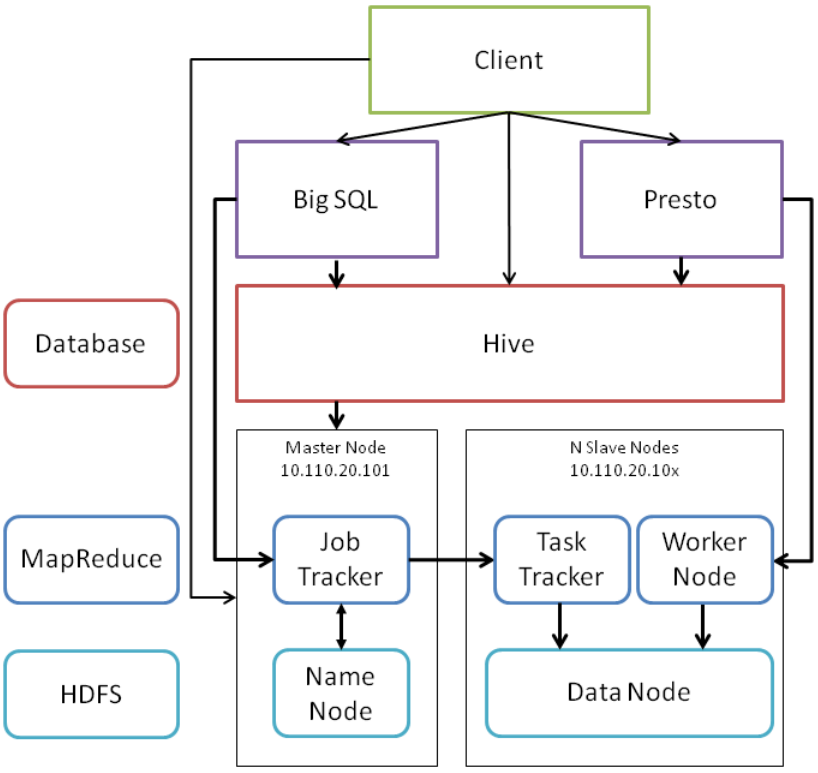


Figure 3.3.: Hadoop Ecosystem

A client software can access directly to Hive, queries are compiled and are executed by running Map-Reduce jobs. Information about the tables are stored in Hive metastore. IBM BigSQL and Presto take benefit of Hive metastore to compile the query. IBM BigSQL is using the Map-Reduce framework and HDFS of the Hadoop system for executing queries. While Presto is only using the HDFS. A client application can connect to Hive, IBM BigSQL, Presto or directly to the Hadoop system interface.

### 3.2.1. Hadoop Ecosystem

Hadoop has a wide ecosystem, which contains many products. There are commercial and non-commercial products for different purposes. For this work, we use only a small part of the Hadoop-related projects. Components of the Hadoop ecosystems are:

- NoSQL databases like Apache Hive [18], Apache HBase [16] or Cloudera Impala [2].
- Metastores like Apache HCatalog [17] which acts like a system catalog of relational databases.
- Workflow scheduler like Apache Oozie [20].

- Machine learning libraries like Apache Mahout [19].
- Tools for loading and collecting data into the Hadoop system like Apache Sqoop [22] and Apache Flume [15].
- High-level language like Apache HiveQL [25], Apache Pig [21], IBM BigInsights Jaql [4] and Twitter Scalding [37] for writing analytical programs.
- Software for managing distributed servers such as Apache ZooKeeper [23].

### 3.2.2. IBM BigInsights

IBM BigInsights contributes an easy to use webconsole, web-installer and many analytic and developer-related tools. It contains Apache Hadoop with associated products for a ready-to-use environment. Furthermore, it bundles a specific version of Hadoop and its components such as Hive, Flume, Oozie, Hbase and others within one package. The components can be selected during the installation. The following list shows the benefits [3] of using IBM BigInsights:

- Easy to use installer
- Realtime-view of cluster status and fine graded perspective on tasks and jobs
- Distribution of configuration-files over all nodes
- Scripts for cluster-management (adding nodes, cluster healthcheck, start and stop of individual components)
- Enhanced security with LDAP-access
- Flexible job scheduling mechanism
- Developer related functions such as eclipse plugin for linking references and automatic upload of code fragments
- Simple deployment and distribution of custom Java applications on all nodes
- Already pre-deployed sample applications, such as wordcount
- Application lifecycle management for data processing applications

IBM has packaged the Hadoop ecosystem in one product, called IBM BigInsights. The most recent is released in version 2.1 which contains following version of Hadoop and its ecosystem:

| component | version |
|---|---|
| Avro | 1.7.2 |
| Flume | 1.3.0 |
| Hadoop | 1.1.1 |
| HBase | 0.94.3 |
| HCatalog | 0.4.0 |
| Hive | 0.9.0 |
| Oozie | 3.2.0 |
| Pig | 0.10.0 |
| Sqoop | 1.4.2 |
| ZooKeeper | 3.4.5 |

Table 3.1.: BigInsights 2.1 components version

There are other vendors which also built packages based on Hadoop. IBM has added various additional features to their package. One, which is important for us, is IBM BigSQL. But also their General Parallel File System (GPFS) [7] and namenode high availability are notable.

### 3.2.3. HDFS

HDFS is the Hadoop Distributed File System [35]. HDFS plays an important role in the entire Hadoop cluster, for this reason it holds all the data of the whole cluster. It scales well, provides fault tolerance through inter-node replication and handles filesystem errors transparent. The HDFS saves the files into blocks of a predefined size. It is common to set this size around 64 or 128 megabytes. For a cluster environment with many terabytes this size could be a poor configuration, it generates a remarkable amount of blocks, which have to be managed by the namenode. In addition, for an optimal data locality on the workernodes, the jobtracker generates one map task per block. A map task is a valuable unit, because it requires an expensive setup of the task on the workernode. With that in mind, setting an optimal blocksize is an important configuration. It is also an important size for a non-splittable compression algorithm. The coordinator of the HDFS is the namenode, which lacks in fault-tolerance. If the node with the namenode-image (the meta-information of the HDFS) fails, the complete HDFS is unavailable. Because of this, there are other filesystem, like the IBM GPFS which could also be used with IBM BigInsights. HDFS works best with read access, which means it is optimal for data warehouses.

### 3.2.4. Apache Hive

Hive [18] is an important part of the Hadoop ecosystem. Primary built from Facebook, Hive has been further developed by developers from the Apache foundation. Hive fills-in between two universes. On one side there is the already well-known relational part of the databases, on the other side there is the new Big Data approach. Hive closes the gap. It supports SQL-like queries, written in a separate language which can be compiled into Map-Reduce jobs [14]. For a better performance, the meta-data will be saved into the Hive metastore, for faster access. At Facebook, Hive contains several hundred terabytes of data for reporting and ad-hoc analysis [36]. Hive is mostly designed for offline OLAP queries, as used in common data warehouses. The data are saved as files and folders in HDFS. Hive gives the user the possibilities to write own de/serialization methods, which for example can be used for enabling compression.

### 3.2.5. HiveQL

HiveQL[25] is a high level query language for Map-Reduce provided by Hive to define and manipulate data stored in Hive. The ANSI SQL-92 standard is not fully implemented, but the language is similar to well known SQL language. The language supports the most primitive data types, collections like arrays and maps and user-defined types. Data Definition Language(DDL) is provided by creating tables statements. Further indexes could be defined and data could be loaded in tables by LOAD and INSERT statements. There is no UPDATE oder DELETE statement. The SELECT statement implements almost all SQL features:

- Group by, Order by and built-in functions
- Joins
- Union all
- Sub queries
- Lateral Views and User-Defined functions (UDF)
- Windowing and analytics functions

There is also an explain tool which shows how the query will be divided into different stages that includes map and reduce jobs.
Below, based on an example of query 5, we illustrate the differences between SQL and the HiveQL syntax:

Listing 3.1: SQL Query 5

```
select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from customer, orders, lineitem, supplier, nation, region
where c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and l_suppkey = s_suppkey
 and c_nationkey = s_nationkey
 and s_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and r_name = 'ASIA'
 and o_orderdate >= date('1994-01-01')
 and o_orderdate < date('1995-01-01')
group by n_name
order by revenue desc;
```

Listing 3.2: HiveQL Query 5

```
select
  n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from
  customer c join
    ( select n_name, l_extendedprice, l_discount, s_nationkey, o_custkey from
        orders o join
      ( select n_name, l_extendedprice, l_discount, l_orderkey, s_nationkey from
          lineitem l join
        ( select n_name, s_suppkey, s_nationkey from supplier s join
          ( select n_name, n_nationkey
            from nation n join region r
            on n.n_regionkey = r.r_regionkey and r.r_name = 'ASIA'
          ) n1 on s.s_nationkey = n1.n_nationkey
        ) s1 on l.l_suppkey = s1.s_suppkey
      ) l1 on l1.l_orderkey = o.o_orderkey and o.o_orderdate >= '1994-01-01'
              and o.o_orderdate < '1995-01-01'
) o1
on c.c_nationkey = o1.s_nationkey and c.c_custkey = o1.o_custkey
group by n_name
order by revenue desc;
```

In HiveQL joins between tables can only be written in the form:
table1 join table2 on key1=key2.
That is the main difference between the SQL query statement and results in a different query structure:

- ◇ The innermost join is first a projection of table Region with the filter predicate r_name = 'ASIA', and then a join to table Nation
- ◇ Now table after table is joined for the end result which is a join between tables Region, Nation, Supplier, Lineitem, Orders and Customer.
- ◇ Filter predicates are used at the time of the join.

### 3.2.6. IBM BigSQL

As mentioned before, writing Map-Reduce jobs to get results of every short query is time-consuming and complex. All existing SQL queries of your existing system have to be rewritten to be executed on a Hadoop system. With IBM BigSQL a new layer is build on top of the Hadoop framework. So that using SQL queries is provided for Hbase and Hive databases.

### 3.2.7. Presto

During our work, Facebook released a successor of Hive. They presented it with the announcement, that it should be 10 times faster then Hive, which is a big promise. In background, it uses Apache Hive as the data metastore. The metastore contains information of the location of the filesystem blocks. It uses the HDFS, but does not generate Map-Reduce jobs. The building process of the Map-Reduce jobs requires a lot of time. As a result, Presto has its own workers and uses the data from HDFS directly, as illustrated by figure 3.3 Hadoop on page 9. It has also its own directory service, which distribute the work to the workers. The directory service connects to the Hive metastore to get the HDFS block location. With that information, an optimal placement of work and data can be made which results in fewest network traffic as possible.

## 3.3. TPC-H Benchmark

To simulate a data warehouse system environment, we decided to execute the TPC-H benchmark [9]. The benchmark provides us a typical database schema and some predefined BI concerned queries, as well as tools for generating data for load and DDL-statements. Transaction Processing Performance Council (TPC) benchmarks are well known, established and supported by all the important database vendors [10].
The schema of the database and the ad-hoc queries were designed to get support the following characteristics: having a high degree of complexity, use a variety of access, process large amount of the stored data and differing from each other.
We use the eight tables in the given schema and the defined 22 queries with scale factor 1000 to create a data warehouse system in a Big Data use case. We run this scenario to measure the performance of our cluster. Extracted data of an operational database had been initially loaded into our system. We operate on this snapshot of the business data and do not update or refresh any of the gathered information.
It was not our ambition to best tune our environment to get the best results for benchmarking. But rather simulate a real world business problem in different database architectures. We were focusing on tuning some database parameters and on query execution optimization. This includes index creation, data partitioning, table replication and table data compression.

Figure 3.4 shows the relationship between the tables, their table specifications and cardinalities. Detailed DDL-statements are found in Appendix A.1 (page 46)

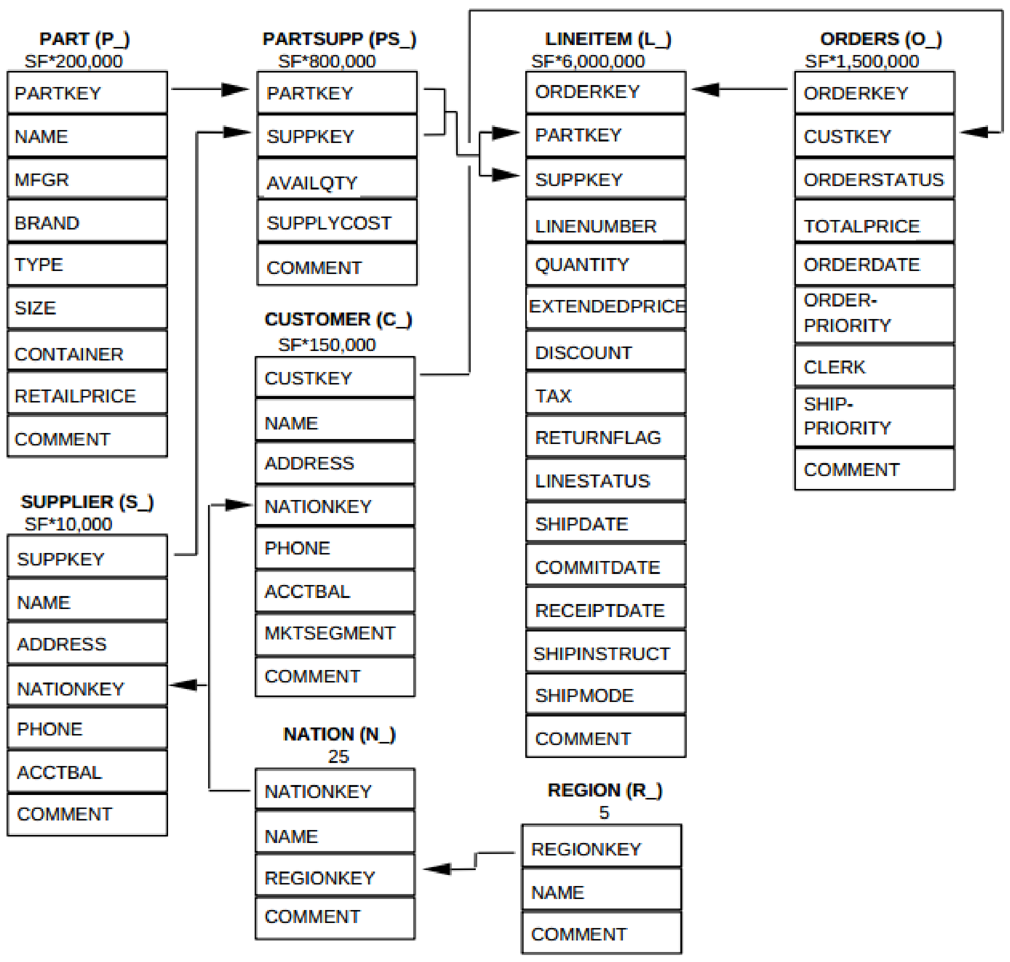


Figure 3.4.: TPC-H - Database Schema [11]

Legend [11]:

– The parentheses following each table name contain the prefix of the column names for that table;

– The arrows point in the direction of the one-to-many relationships between tables;

– The number/formula below each table name represents the cardinality (number of rows) of the table. Some are factored by SF, the scale-factor, to obtain the chosen database size. The cardinality for the Lineitem table is approximate;

TPC provides the dbgen tool to generate load data for the tables. The tool implements several scale-factors (for example: 300,1000,3000) for sizing table data. We took the Scale-Factor 1000, this brings us up to 8.66 billion (8'659'989'739) records in the Lineitem table.

| Table Name | Number of Rows |
|---|---:|
| Customer | 150,000,000 |
| Lineitem | 5,999,989,709 |
| Nation | 25 |
| Orders | 1,500,000,000 |
| Part | 200,000,000 |
| Partsupp | 800,000,000 |
| Region | 5 |
| Supplier | 10,000,000 |

Table 3.2.: Cardinality of Tables

The SQL query statements are provided by the TPC-H Benchmark tool suite [12], also HiveQL query statements can be copied from the Hive Jira Issue 600 [24], which contains the TPC-H Benchmark for Hive. Each of the 22 queries deals with questions relating to business facts about pricing and promotions, supply and demand management, profit, customer satisfaction, market share and shipping management. Depending on the needed information, the queries contain multiple joins, aggregate functions, group by clauses and sub-select statements. The detailed SQL statements are listed in Appendix C (page 60)
The table 3.3 on page 17 summarize the characteristics of each query.

Table 3.3.: TPC-H query characteristics

| No. | Tables | No. of results rows | Aggregation Function | Group by | Sub-select |
|---|---|---|---|---|---|
| Q1 | LineItem | 5 | sum+avg | yes | |
| Q2 | Nation, Part, Partsupp, Region, Supplier | 100 | | | yes |
| Q3 | Customer, Lineitem, Order | 10 | | yes | |
| Q4 | Lineitem, Orders | 5 | | yes | yes |
| Q5 | Customer, Lineitem, Nation, Orders, Region, Supplier | 5 | | yes | |
| Q6 | Lineitem | 1 | sum | | |
| Q7 | Customer, Lineitem, Nation, Orders, Supplier | 4 | | yes | |
| Q8 | Customer, Lineitem, Nation, Orders, Part, Region, Supplier | 2 | sum | yes | yes |
| Q9 | Lineitem, Nation, Orders, Part, Partsupp, Supplier | 175 | sum | yes | yes |
| Q10 | Customer, Lineitem, Nation, Orders | 20 | sum | yes | |
| Q11 | Nation, Partsupp, Supplier | 1048 | sum | yes | yes |
| Q12 | Lineitem, Orders | 2 | sum | yes | |
| Q13 | Customer, Orders | 42 | count | yes | yes |
| Q14 | Lineitem, Part | 1 | sum | | |
| Q15 | Supplier, View_revenue | 1 | | | |
| Q16 | Part, Partsupp | 18314 | | yes | yes |
| Q17 | Lineitem, Part | 1 | sum+avg | | yes |
| Q18 | Customer, Lineitem, Orders | 57 | sum | yes | yes |
| Q19 | Lineitem, Part | 1 | sum | | |
| Q20 | Lineitem, Nation, Partsupp, Supplier | 204 | sum | | yes |
| Q21 | Lineitem, Nation, Orders, Supplier | 100 | | yes | yes |
| Q22 | Customer, Orders | 7 | sum | yes | |

# 4. Experiments

In this chapter we first list which hardware we used for our experiments in the environment section. Followed by a section for our settings and configuration of the relational platform and finally a section with explanation of Big Data cluster setup.

## 4.1. Environment

Thanks to IBM Switzerland we executed our experiments on an IBM bladecenter hardware cluster.

Figure 4.1.: Test Environment: Hardware cluster located at IBM Switzerland

### 4.1.1. Cluster Hardware Information

For the performance measurements, we used blade-servers inside the bladecenter. The ten worker-nodes offer following resources:

| | |
|---|---|
| **Server model** | IBM eServer BladeCenter HS21 -[8853L4G]- |
| **CPU** | Intel(R) 2x Xeon(R) CPU 5140 - total 4 Cores |
| **Memory** | 12 GB |
| **Network** | NetXtreme II BCM5708S Gigabit Ethernet |
| **Disk** | 1x 900 GB SAS IBM-ESXS ST9900805SS - Logical block size: 512 bytes |
| **OS** | RHEL 6.4 X86-64, GNU/Linux 2.6.32 |

Table 4.1.: Hardware Information

## 4.2. IBM DB2 Cluster

We used all 10 nodes of the bladecenter for our calculations. IBM DB2 DPF works best on a homogeneous clustered system. It is the reason, why we only selected 10 nodes.

### 4.2.1. Configurations

A IBM DB2 DPF cluster build up from a normal IBM DB2 system. There are a few additional tasks which have to be done. For simplicity, the IBM DB2 home directory is shared with Network File System (NFS) to the other cluster-nodes. It contains the needed information about the database instance. In a IBM DB2 DPF cluster, a partition is the smallest unit of computational power. It is possible to map one cpu-core to one logical partition. That removes the data copying effort from one core to another core during computation. But for our scenario, one partition is one worker-node, and has one logical node. It means that every hardware system represents one node, with one logical node on it. Our db2nodes.cfg lists the 10 nodes with each one partition as follow:

| partition | hostname | logical node |
|---|---|---|
| 0 | sa-biginsights-110-20-101-rh6 | 0 |
| 1 | sa-biginsights-110-20-102-rh6 | 0 |
| 2 | sa-biginsights-110-20-103-rh6 | 0 |
| 3 | sa-biginsights-110-20-104-rh6 | 0 |
| 4 | sa-biginsights-110-20-105-rh6 | 0 |
| 5 | sa-biginsights-110-20-106-rh6 | 0 |
| 6 | sa-biginsights-110-20-107-rh6 | 0 |
| 7 | sa-biginsights-110-20-108-rh6 | 0 |
| 8 | sa-biginsights-110-20-109-rh6 | 0 |
| 9 | sa-biginsights-110-20-110-rh6 | 0 |

Table 4.2.: db2nodes.cfg-file

In addition to this, our cluster has to communicate from node to node. While we use an own subnet we could neglect security perspectives and focus our cluster for performance. We use the well known rsh-toolkit for remote shell-commands. An other option would be ssh, which uses more cpu-power for encryption.

### 4.2.2. Database Organization

We decided to organize our database with the following settings:

– 2 Partition Groups: one large group over all nodes for the big tables and one group for the two small tables Nation and Region which are stored only on one node and will be replicated over the other nodes.

– Every partition group has its own tablespace and bufferpool

The following figure 4.2 shows the organization of our TPC-H database on the IBM DB2 DPF system.

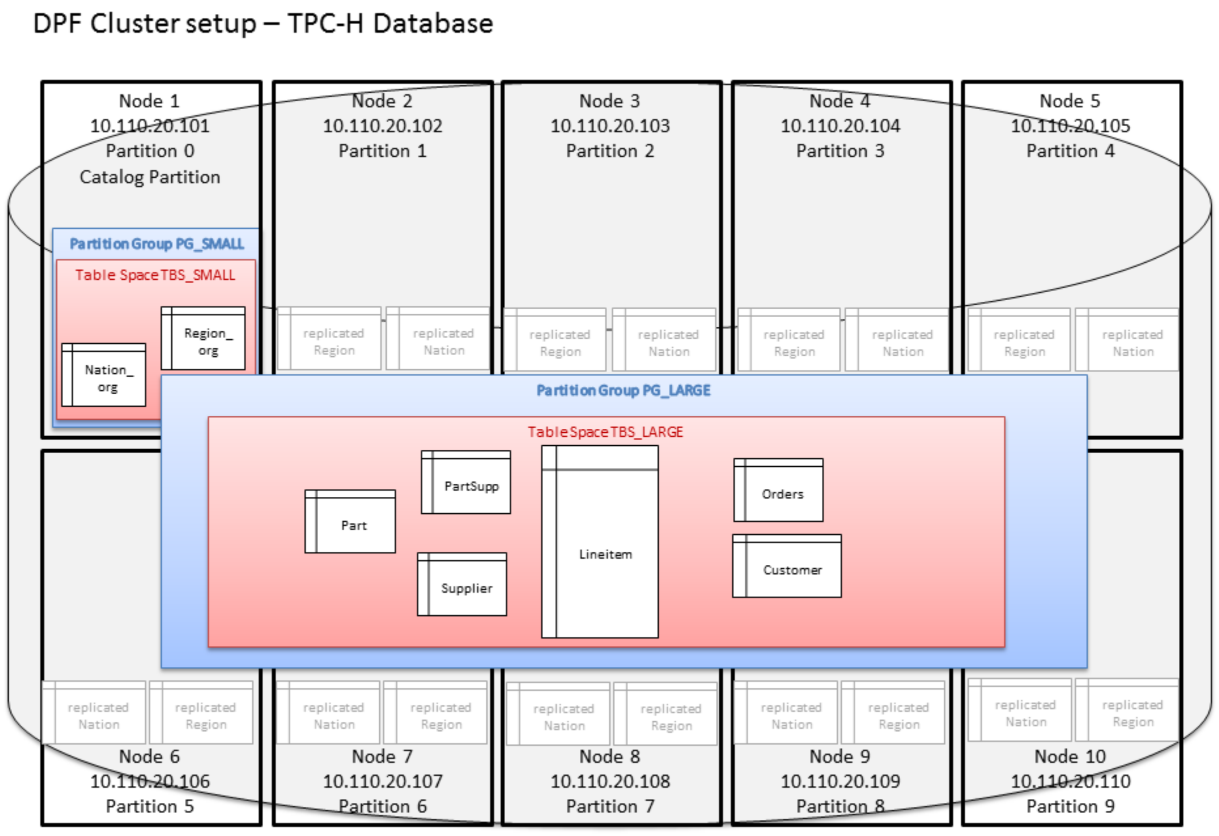


Figure 4.2.: DPF Cluster Organization

The main focus of our database design was to determinate the distribution keys of all the large six tables of the TPC-H benchmark. Rows of the table are partitioned by a hash function of the specified distribution key column. Data skew is an important factor at partitioning data across multiple nodes. Data skew means that distribution of data should be equally with the same amount of data on each node. To avoid data skew on the nodes the used columns have to provide many distinct values which leads to the primary key columns of a table. On the other side similar values of data should reside on the same node for joining queries. Typically the queries which will be executed over the database are using different selections and functions over columns of all the tables, which denies the satisfaction of both before mentioned requirements in a DPF cluster setting.
We chose the primary keys of the tables Supplier, Part, Partsupp and Customer as distribution key and we consider co-location of same values for tables Lineitem and Orders with setting of distribution key L_ORDERKEY and O_ORDERKEY.
We discussed this configuration with Jürgen Götz [1], in a meeting about the table schema of TPC-H benchmark. He confirmed our thoughts about replicate the small tables and divide the entire database into two partition groups.

### 4.2.3. Query Execution

In a DPF system the execution of the whole query is split over all the nodes which participate in the cluster. Via the hash value of the distribution key of the row, the system knows which rows are stored on which node. The system coordinates the parallel processing of the query by sending the work to nodes which store the desired row. Each node processes the step on his own, based on his own result sets and stores it in a temporary table. These temporary results are then sent to the next node which executes the next step of the query. For fast query processing, sending data from node to node has to be minimal. That means in the query plan, Directed Table Queue (DTQ) and Broadcast Table Queue (BTQ), should occur very limited.
BTQ will occur, if a node sends all data to the other nodes. In worst case, this broadcast could be done from all nodes simultaneously, which is called a Merge Broadcast Table Queue (MBTQ). At the other side, a DTQ is the most efficient way to transfer data inside the IBM DB2 DPF cluster. It sends the data from one node direct to an other node.
Data is not broadcasted over network, when same values of the table keys reside on the same node. With our TPC-H use case we could not avoid data skew and look for same values of keys on same node at the same time. Here we have made our compromise by storing rows with values of L_ORDERKEY of tables Lineitem and Orders on the same node, but distribute all other table rows equally on all available nodes.

We want to point out the compromise which has to be done in a DPF environment by providing some detailed information about the query access plan for query number 5.

[1] Jürgen Götz, IBM DB2 specialist, IBM Germany

Listing 4.1: the SQL Statement for Query 5

```
select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from customer, orders, lineitem, supplier, nation, region
where c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and l_suppkey = s_suppkey
 and c_nationkey = s_nationkey
 and s_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and r_name = ’ASIA’
 and o_orderdate >= date(’1994-01-01’)
 and o_orderdate < date(’1995-01-01’)
group by n_name
order by revenue desc;
```

Listing 4.2: DB2 Query Optimizer Plan of Query 5

```
                             RETURN
                              ( 1)
                               |
                             TBSCAN
                              ( 2)
                               |
                              SORT
                              ( 3)
                               |
                              GRPBY
                              ( 4)
                               |
                             TBSCAN
                              ( 5)
                               |
                              SORT
                              ( 6)
                               |
                              BTQ
                              ( 7)
                               |
                             GRPBY
                              ( 8)
                               |
                             NLJOIN
                              ( 9)
                     /------/      \---------\
                 DTQ                          *
                 (10)                         |
                  |                   Index:
                HSJOIN                IDX1311111743040
                 (11)                 SUPPLIER
        /------/      \--------\
  TBSCAN                        DTQ
   (12)                         (13)
    |                            |
 Table:                        HSJOIN
 LINEITEM                       (14)
                /-------------/      \-----------\
            DTQ                                   NLJOIN
            (15)                                   (17)
             |                             /-----/      \----------\
           IXSCAN                    NLJOIN                         *
            (16)                      (18)                          |
             |                      /      \-----\          Index:
      Index:                 IXSCAN               *         IDX1311111743470
      IDX1311111743350         (19)               |         CUSTOMER
      ORDERS                    |          Index:
                        Index:             IDX1311111743520
                        IDX1311111742470   NATION
```

```
                    REGION
```

The Query Plan is separated in 19 different Steps:

- Step 19: An index scan (IXSCAN) through table Region via index (R_NAME,R_REGIONKEY) is shown.
- Step 18: Nested Loop Join (NLJOIN) between Region and Nation is done via indexes IDX1311111742470 (R_NAME,R_REGIONKEY) and IDX1311111743520 (N_REGIONKEY, N_NAME N_NATIONKEY)
- Step 17: Result of previous operation is joined via NLJOIN with table Customer. Instead of tablescan, the index IDX1311111743470 (C_NATIONKEY, C_CUSTKEY) of table customer is used.
- Step 16: Access on Order table is performed by IXSCAN.
- Step 15: Because same values of O_ORDERKEY from table Orders and Customer not reside on the same node, the required values of O_ORDERKEY in table Orders are sent via DTQ to the node which calculates step 14.
- Step 14: Values of table Orders and the previous calculated temp result of joining Region, Nation and Customer are joined by a Hash Table Join (HSJOIN).
- Step 13: The result of step 14 is sent via DTQ to the next node which processes step 11.
- Step 12: In this step a full table scan (TBSCAN) over Lineitem table is executed.
- Step 11: Then the earlier temp result is joined by HSJOIN with table Lineitem.
- Step 10: For joining the calculated temp result with the table Supplier, this result is sent by DTQ to the node which serve the next join.
- Step 9: The result of the join on step 11 is now joined with table Supplier via IDX1311111743040 (S_NATIONKEY, S_SUPPKEY)
- Steps 8-1: Now the result is first grouped by (GRPBY) the column N_Name and the result is broadcasted over all nodes (BTQ) for sorting. Data is sorted (SORT) by the column "Revenue" and finally the result is returned back to the node, on which the SQL query was issued.

### 4.2.4. Performance Settings

Except the ones listed in Appendix A.1 (page 46), we left all database configuration parameters on default values.

Hardware performance improvements were done by enabling parallel I\O on disk access level and aligning filesystem blocksize to 4kb pagesize. The alignment is very important, because the filesystem has to match the pagesize. A harddisk is divided into sections, with a fixed level. Newer disks are using 4 kbytes, while older disks are still using a size of 512 bytes. The pagesize has to be the same or a multiple of the underlying filesystem blocksize, while the filesystem blocksize has to be the same or a multiple of the underlying harddisk sectorsize. Without a correct alignment, IBM DB2 has to read multiple blocks, while it only needs one block. This results in double I\O requests, which would generate a massive performance impact.

Second we build up some indexes which allows IBM DB2 a more direct access to the information of the used columns. This means not the whole row has to be read while query execution. Although indexes are a good implementation for tuning performance, the index are only based on certain queries. There is no way to generally decrease reads for whole rows dynamically from query to query. The "DB2 Create Indexes" section in Appendix A.2 (page 50) shows detailed information about the indexes we set.

Third aspect of tuning was enabling compression. Because our workload was very I\O intensive we had to compress our rows. Classic row compression in IBM DB2 is implemented by using a sort of Lempel-Ziv algorithm which works with dictionary codes. IBM DB2 provides easy activation of compression by altering the tables.

Listing 4.3: DB2 Alter Table For Compression

```
DB2 ALTER TABLE nation COMPRESS YES STATIC
```

The use of compression on our tables did not only decreased storage usage on the node, but also the query execution did profit of compression because more rows could be stored in the bufferpool and fewer I\O operations were needed.

## 4.3. IBM BigInsights Cluster

The following table is showing the underlying hardware we used for our masternode, which includes the namenode and the jobtracker.

| | |
|---|---|
| **Server model** | IBM eServer BladeCenter HS21 -[7995HVG]- |
| **CPU** | Intel(R) Xeon(R) CPU E5450 - quadcore |
| **Memory** | 24 GB |
| **Network** | NetXtreme II BCM5708S Gigabit Ethernet |
| **Disk** | 1x 900 GB SAS IBM-ESXS ST9900805SS - Logical block size: 512 bytes |
| **OS** | RHEL 6.4 X86-64, GNU/Linux 2.6.32 |

Table 4.3.: Hadoop master/namenode/jobtracker

### 4.3.1. Hadoop Configurations

Hadoop has its own configuration-files inside the installation directory: "/opt/ibm/biginsights/" in our case. We did configuration-changes on the following files:

**core-site.xml** lets you change core Hadoop settings, like the available compression algorithm.

**hdfs-site.xml** allows to change the HDFS blocksize and replication factor inside HDFS.

**mapred-site.xml** allows to set some Java Virtual Machine (JVM) settings, like Java heap size.

**ibm-hadoop.properties** defines the available map and reduce-capacity of a node.

During our work, we had to increase the Java heap size and we used temporary settings in Hive, to obtain the better results.

### 4.3.2. Hive Database Organization

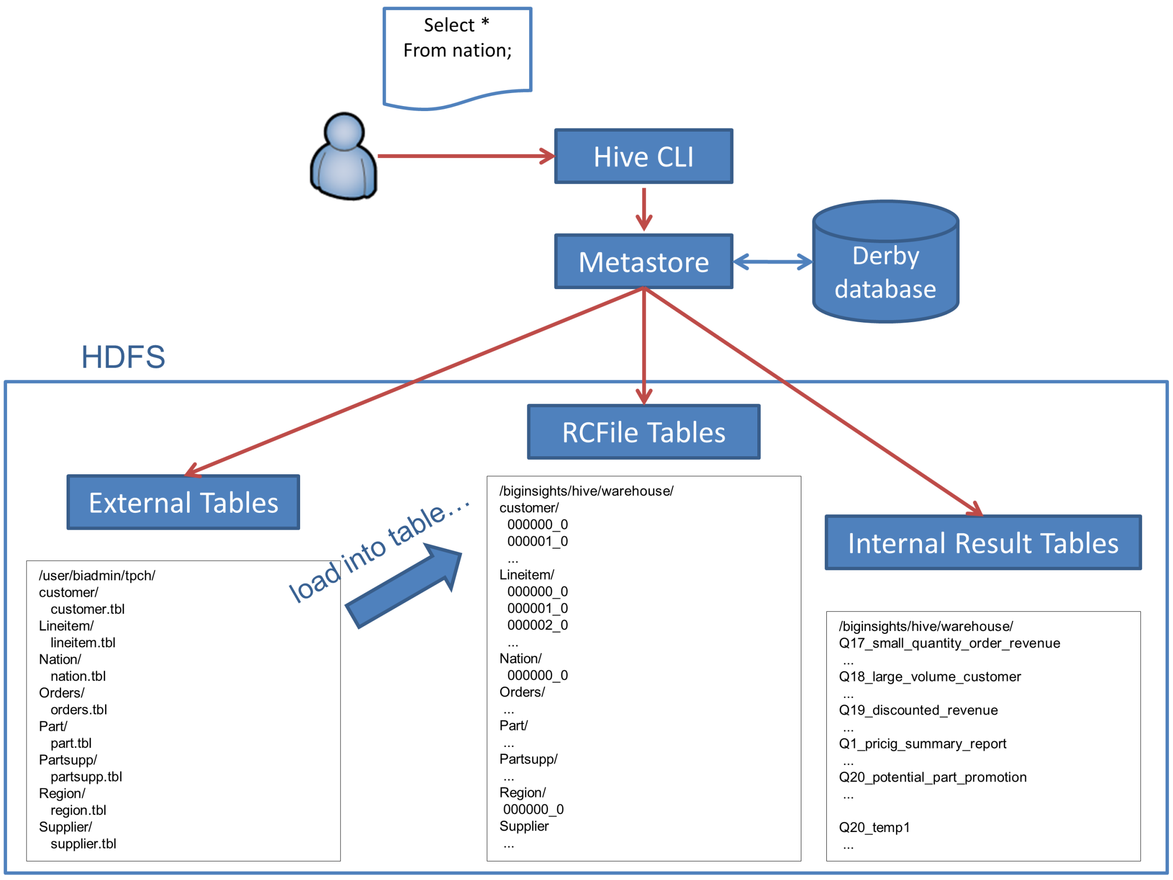


Figure 4.3.: Hive Database Organization

As seen on figure 4.3 we are using three different types of tables in Hive.

#### External Stage Tables

External tables are tables that are defined over a certain directory in the HDFS. A table structure will be lying above the directory. Advantage of this type of table is, that files can be placed into the specified directory and no action in Hive has to be done in order to populate data. It is unproblematic loading a table by simply add additional files to the directory. We load the generated table data from local filesystem into HDFS path /user/biadmin/tpch/ Each table has its separate directory.

#### Internal Result Tables

Internal Tables are mostly used for storing intermediate results of executed queries. After a select statement was processed, we put the results of that query into a Hive table. Results of the query could be displayed by executing a select * from "result table". All Hive tables are stored in the Hive home-directory /biginsights/hive/warehouse/. A table is realized as directory and the result data is stored in files under this directory.

#### RCFile Tables

As we enabled the compression with RCFile, we built new internal Hive tables with the condition "STORED AS RCFile". For each of the eight tables a new directory in the Hive home-directory was applied and with issuing the load statement, data of the staging tables were loaded with compression into the new Hive internal table directories. In this configuration is is possible to run query against the uncompressed data at the stage tables or against the compressed data at the internal tables.

### 4.3.3. Selected Compression Algorithm

SQL queries on a Hadoop clustered system generate a lot of I\O requests. Every map-task has to read the complete HDFS block. As described in the HDFS section, we changed the default block size to a bigger value (1280 megabytes space), to reduce the amount of map-tasks during one job. At the other side, the map-tasks have to read more information from disk and generates more I\O during this setup of a task. While the block is read, the cpu is mostly idle. We monitor the system with nmon during query execution. The following picture 4.4 is an example of the disk-load during an execution of a query:

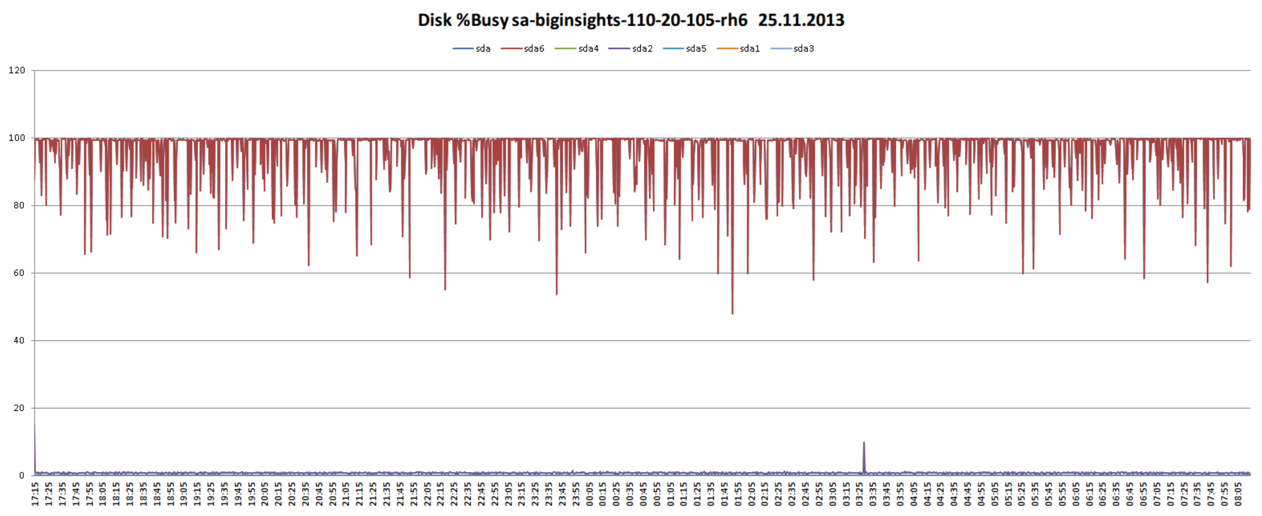


Figure 4.4.: Disk load during a query

IBM BigInsights supports the following compression algorithms:

**org.apache.hadoop.io.compress.DefaultCodec**

**org.apache.hadoop.io.compress.GzipCodec**
Not splittable compression algorithm

**org.apache.hadoop.io.compress.BZip2Codec**
Not splittable compression algorithm

**com.ibm.biginsights.compress.CmxCodec**
Unknown compression algorithm, further measurements would be necessary to recommend this codec.

**org.apache.hadoop.io.compress.SnappyCodec**
Only available in the IBM BigInsights package since version 2.1.0.1

**com.hadoop.compression.lzo.LzopCodec**
Additional effort needed in the Hadoop environment

Choosing a compression-algorithm is a trade-off between cpu-load and resulting compression efficiency. With more available cpu-power, a better compression factor could be achieved, at a predefined time-window. For the use in the Hadoop environment, the reading of a HDFS block from a specific line-number with a predefined offset is very important. That enables the jobtracker to generate more fine-grained tasks. It is extremely important for tasks which operates on a small input, like on one line, as the default input-reader of Hadoop basically works. In such use cases, the optimal codec from our perspective would the snappy codec [26], developed by Google. It generates a respective file size, at the time of reasonable cpu load. In addition, it is splittable, which means that a mapper can read from a specified offset and decompress it. A non splittable compression requires the read of the whole file. An offset can not be used to jump to a specific line, because to read one line, the header before had to be read.
In our use case, we are accessing a full range of data by one map task. A map-task has to read

HDFS block and scan this. During the read phase, we could use a lot of cpu power. The processing of a line (tablescan phase), for a specific filter condition, requires very less cpu power. With that in mind we choose the gzip-codec [13]. It is not splittable, but with the following Hive performance settings 4.3.7 in mind, that does not matter. We do not recommend this compression codec in other use cases. The not-splittable and cpu intensive characteristics could results in immense drawbacks.

### 4.3.4. Bottleneck

We are using cheap commodity disks, with 900 GB. In a reliable productive server environment more expensive disks with higher I\O-rates are used, also with raid-strategies on the disk-controller. But Hadoop has the aim to use commodity hardware. The HDFS throughput could theoretically be calculated by a aggregation of all underlying disks [30].
While we are using Hadoop and HDFS for SQL queries, we had to adapt the strategy from the query compiler. Especially join-operations within Hive or IBM BigSQL can only be done on the full available dataset. Some special exceptions (like small tables) could be done within the map-job, but usual join-operations are done in the reduce-job.
The reduce-jobs are depending on the map-jobs and could only be fully completed, when the map-jobs are completed. We need a high transfer rate within the map-tasks for reading the data, and a high transfer rate while writing the data in internal tables from reduce-jobs.
Also queries which are very selective needs a full table scan, which means the full table have to be read. Our experience shows generating many thousands map-tasks for reading data in small parts is less efficient with our workload. We decided to use a large HDFS blocksize, which causes less map task during query execution and we used compression for gathering more data values at reading one block.

### 4.3.5. Record Columnar File (RCFile)

With the growth of NoSQL databases, the solution of using columnar stores shines as they show better performance for analytical workloads. Disadvantage is, that building the entire row costs more than in a row format orientated database. Facebook did a trade off between storing data in rows and storing data in columns and introduced a structure called RCFile [27]. An important difference is, that the compressed data is not decompressed at every read access.
The following figure 4.5 shows the structure of RCFiles:

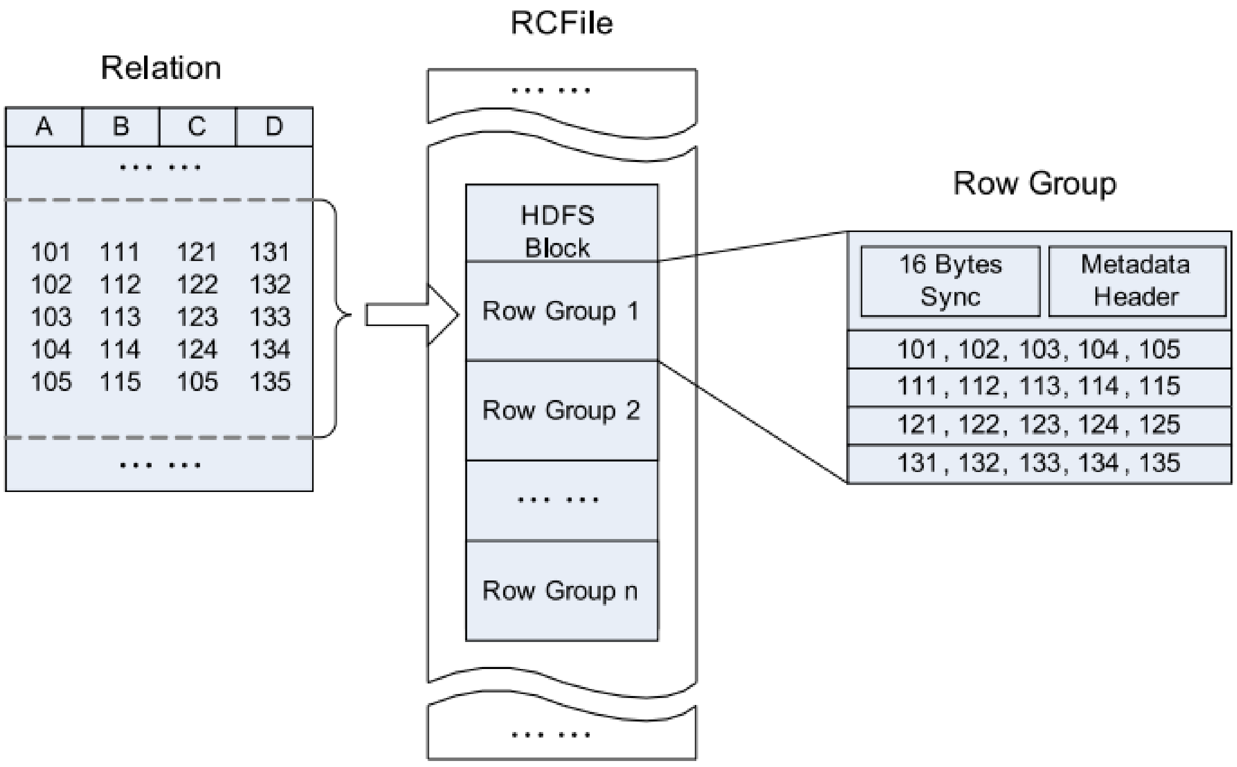


Figure 4.5.: Structure of RCFile [27]

A relation is stored in multiple HDFS blocks. A HDFS block has multiple row groups, that means the records of a table are spread into certain row groups. Ever row group has e preamble with meta information and the table data. In the table data the values of a column are stored as a record like in columnar databases.
By storing meta information about the stored values in the row groups, dynamically for each query the query compiler can decide which values are needed for the calculation and has not to load the entire row into memory for executing the query operations.

### 4.3.6. Query Execution

As mentioned before, the Hive query statement is transformed in multiple Map-Reduce jobs. The query plan shows different stages and how they depend on each other. As an example we present a part of the query plan for Query 5, first the HiveQL query statement:

Listing 4.4: HiveQL Query 5

```
select
  n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from
  customer c join
    ( select n_name, l_extendedprice, l_discount, s_nationkey, o_custkey from
        orders o join
```

```
      ( select n_name, l_extendedprice, l_discount, l_orderkey, s_nationkey from
          lineitem l join
        ( select n_name, s_suppkey, s_nationkey from supplier s join
          ( select n_name, n_nationkey
            from nation n join region r
            on n.n_regionkey = r.r_regionkey and r.r_name = 'ASIA'
          ) n1 on s.s_nationkey = n1.n_nationkey
        ) s1 on l.l_suppkey = s1.s_suppkey
      ) l1 on l1.l_orderkey = o.o_orderkey and o.o_orderdate >= '1994-01-01'
              and o.o_orderdate < '1995-01-01'
) o1
on c.c_nationkey = o1.s_nationkey and c.c_custkey = o1.o_custkey
group by n_name
order by revenue desc;
```

and a part of the Q5 query access plan:

Listing 4.5: Hive Query Plan of Query 5

```
...

STAGE DEPENDENCIES:
  Stage-8 is a root stage
  Stage-5 depends on stages: Stage-8
  Stage-6 depends on stages: Stage-5
  Stage-7 depends on stages: Stage-6
  Stage-1 depends on stages: Stage-7
  Stage-2 depends on stages: Stage-1
  Stage-3 depends on stages: Stage-2
  Stage-0 depends on stages: Stage-3
  Stage-4 depends on stages: Stage-0

STAGE PLANS:
  Stage: Stage-8
    Map Reduce
      Alias -> Map Operator Tree:
        o1:l1:s1:n1:n
          TableScan
            alias: n
            Reduce Output Operator
              key expressions:
                    expr: n_regionkey
                    type: int
              sort order: +
              Map-reduce partition columns:
                    expr: n_regionkey
                    type: int
              tag: 0
              value expressions:
                    expr: n_nationkey
                    type: int
                    expr: n_name
                    type: string
        o1:l1:s1:n1:r
          TableScan
            alias: r
            Filter Operator
              predicate:
                  expr: (r_name = 'ASIA')
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: r_regionkey
                      type: int
                sort order: +
                Map-reduce partition columns:
                      expr: r_regionkey
```

```
                   type: int
             tag: 1
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1
          handleSkewJoin: false
          outputColumnNames: _col0, _col1
          Select Operator
            expressions:
                  expr: _col1
                  type: string
                  expr: _col0
                  type: int
            outputColumnNames: _col0, _col1
            File Output Operator
              compressed: false
              GlobalTableId: 0
              table:
                  input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
...
```

◇ A list of all the stages that will be generated is shown in the section STAGE DEPENDENCIES. This section also illustrate the dependencies among the stages.

◇ Each of the eight given stages contains a simple Map-Reduce-Job. Whereas stage 8 is the root stage.

◇ Stage 8 builds the following part of the Hive query:
select n_name , n_nationkey
from nation n join region r
on n.n_regionkey = r.r_regionkey and r.r_name = ' ASIA '

in the explain plan we see these operations:

– Stage 8 Map Operator Tree:
On table Nation a reduce output operator selects n_nationkey and n_name field and place them as value columns under the key of n_regionkey. And on table Region there is a filter operator on field r_name = 'ASIA'. That means if the rows contains the value 'ASIA' in r_name field the value of r_regionkey is written to an integer field.

– Stage 8 Reduce Operator Tree :
An inner join is executed with the output of a temp file that has 2 fields _col10 as key value and _col1 as value field.

◇ After all these map and reduce operations, at the end of this stage, a temp file is written which is structured into key field with value regionkey and value field name. It contains all rows of table Region which full fit name = 'ASIA' and value of regionkey is existing in both tables Nation and Region.

### 4.3.7. Hive Performance Settings

Hive has a lot of configuration-options. With those, Hive can be optimally adapted to a certain workload. For a benchmark the optimal settings has to be found. We tried multiple variations, especially with compression.
The following statements were set in the Hive-shell during the setup of internal tables in Hive.

**hive.exec.compress.output=true**
: Tells the Hive engine to compress the output.

**mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec**
: Used codec-algorithm for the compression inside the RCFile.

**mapred.output.compress=true**
: Compress the output of Map-Reduce job.

**hive.merge.mapredfiles=true**
: Allows Hive to merge the generated files of the internal table.

The following configuration-options are special in our use case. Because we are using a not-splittable compression (section: Selected Compression Algorithm), we had to use a fix filesize. The blocksize of HDFS and the compressed blocks had to match best possible.

**mapred.min.split.size=1024000000**
: Set the minimum split size for a Map-Reduce task.

**mapred.max.split.size=1024000000**
: Set the maximum split size for a Map-Reduce task. The parameter before and this is used to assign a fix value for the split size of the Map-Reduce task.

**hive.merge.smallfiles.avgsize=1024000000**
: Defines the average file-size of the merged files.

**hive.merge.size.per.task=1024000000**
: Set the threshold to the same size as smallfiles.avgsize. Because of the compression, it is very important to process this file-size per task.

**mapred.min.split.size.per.node=1024000000**
: The minimum per Map-Reduce-task per node, has to be the same size as desired splitsize.

**mapred.min.split.size.per.rack=1024000000**
: The minimum per Map-Reduce-task per rack, has to be the same size as desired splitsize.

The following statements were set in the Hive-shell during the execution of the various queries. All those options which start with "hive" could also be set in the Hive-config file.
/opt/ibm/biginsights/hive/conf/hive-site.xml

**hive.optimize.autoindex=true**
: Allows Hive to create index, to optimize the table scan strategy.

**hive.exec.parallel=true**

Enables to execute multiple stages in parallel, which could increase the cluster utilization.

**hive.auto.convert.join=true**
Allows the Map-task to execute join-operations. Useful for join within multiple tables, when one of them is smaller.

**hive.auto.convert.join.noconditionaltask = true**
Enables to execute joins of non conditional tasks in parallel.

**hive.auto.convert.join.noconditionaltask.size = 10000**
Sets a limit of the size of a table to join inside a map task.

**mapred.child.java.opts = -Xmx2048m**
Sets the Java heap-size.

### 4.3.8. IBM BigSQL Performance Settings

IBM BigSQL is designed to support existing queries, with fewest modifications as possible. It supports all applications with JDBC or ODBC drivers, which is a big benefit for existing applications. With that in mind, IBM has the aim to reduce the configuration parameters while choosing the best parameters by itself, which means there are no necessary performance settings for us to configure.

IBM BigSQL divides the execution into multiple parts. Those parts can be executed in parallel on the Map-Reduce cluster. But its analyzer is clever enough to not split every task, because some tasks are faster at execution on a single-node, called as local query execution. It uses the Hadoop cluster, but not as excessive as Hive. In addition, the query optimizer imply much information of the Hive metastore to determine a efficient data access. If not all information are available for this process, the user can define some helping hints, like execution strategy, or join method, as an example.

The following listing shows an overview about the settings which are automatically used during the query execution:

**bigsql.memoryjoin.size=10485760**
Enables joins in map-tasks for input smaller than 10 MB.

**bigsql.reducers.max=999**
Sets the maximum reducers.

**bigsql.reducers.autocalculate=true**
Calculates the optimal amount of reducers for a job.

**bigsql.reducers.bytes.per.reducer=1073741824**
This value represent the size of the input of a reducer. It calculates implicit the amount of reduce-tasks

**bigsql.localmode.size=209715200**
Sets a threshold for generating a local job. A local job is done inside the IBM BigSQL server. Beyond this size a Map-Reduce job is generated.

## 4.4. Presto

Since Presto is only used with the Cloudera Distribution for Hadoop, we run against problem while we are trying to connect to an IBM BigInsights test-cluster. We get response with no content from the presto-server. With the same installation, a query was possible to the virtual image of Cloudera server. We fetched the newest source from github (0.55-snapshot) which was available at this time and compiled it by ourselves. For that reason, we could not measure any comparable query execution times on our physical IBM BigInsights cluster.

# 5. Results

## 5.1. Comparison of Query Execution Time

We executed the TPC-H queries in sequential mode at experiments over all three products. The following table 5.1 shows an overview of the query execution measurements:

Table 5.1.: 10 Nodes Query Execution Times

| Query | IBM DB2 | Apache Hive | IBM BigSQL |
|---|---|---|---|
| Q1 | 829.54 | 1382.78 | 2219.58 |
| Q2 | 355.29 | 1196.42 | 4697.85 |
| Q3 | 1301.77 | 15349.96 | 2628.39 |
| Q4 | 489.06 | 1351.50 | 1766.90 |
| Q5 | 754.92 | 10261.20 | 4106.66 |
| Q6 | 389.71 | 420.66 | 510.61 |
| Q7 | 11260.57 | 14038.21 | 4075.17 |
| Q8 | 571.15 | 24606.98 | 3998.29 |
| Q9 | 5496.56 | 23727.50 | 6247.77 |
| Q10 | 886.39 | 7654.54 | 3536.00 |
| Q11 | 80.10 | 1363.17 | 3029.58 |
| Q12 | 508.64 | 4551.25 | 1328.54 |
| Q13 | 470.81 | 1579.21 | 4405.03 |
| Q14 | 426.14 | 931.16 | 1700.88 |
| Q15 | 426.14 | 974.31 | 222.76 |
| Q16 | 165.37 | 2098.35 | 0.00* |
| Q17 | 728.10 | 7591.87 | 10484.52 |
| Q18 | 1193.20 | 30807.01 | 9517.80 |
| Q19 | 399.07 | 5711.35 | 979.12 |
| Q20 | 52500.54 | 2523.51 | 4193.19 |
| Q21 | 29075.22 | 33513.27 | 12066.05 |
| Q22 | 2602.27 | 1670.95 | 2543.30 |

*refer to section 5.3 Limitations for further details

Figrue 5.1 illustrates the query execution times of all three used products:

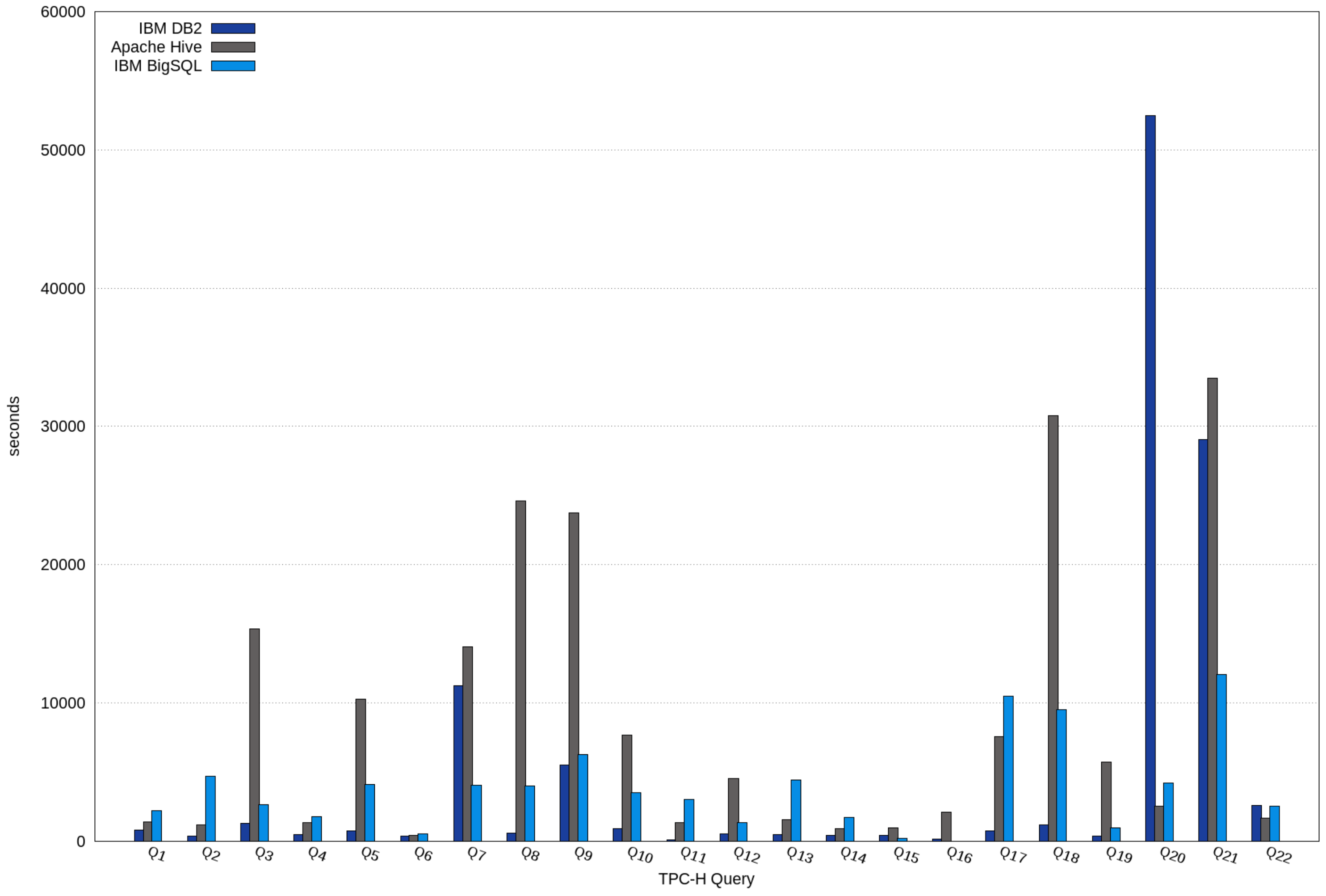


Figure 5.1.: Overview of 10 Nodes Measurements

In summary the IBM DB2 cluster delivers the highest throughput. Q20 is identifiable as a distinctive outlier value of the entire measurements series.
The Q20 contains a lot of subqueries, which results broadcasting temp-tables from every node to every node. Totally $10^9$ tables queues are sent over the network. This generates a lot of traffic inside the cluster. This has a massive impact to the query execution time. We recognize the same behavior during testing on our separate virtual machine cluster.
At least 3/4 of queries on IBM DB2 were executed in a substantial shorter time in comparison to Hive and IBM BigSQL.

Altough Hive is designed for scaling out to hundreds of nodes, in our case - with using only 10 nodes - query execution times are admissible for this workload. We expect, that Hive is mostly used in asynchronous tasks.

The figure shows that IBM BigSQL has no immense outliers in our measurement series. IBM is experienced in compiling SQL queries and has a deep knowledge of executing query plans in parallel. With IBM BigSQL, IBM shows the possible capabilities of the Hadoop platform and its components like Hive.

After gathering results of Hive measurements it is surprising to see, that queries with IBM BigSQL are notable fast in comparison to Hive.

## 5.2. Scale-out Measurements

We describe the scale out characteristics for each of the particular systems. In database environments linear scale out behavior for query execution is extremely uncommen. Because indexes are implemented as B-Trees it is known that an index scan goes with the complexity of O(log(n)).Accessing data through index is the fastest method at query execution. Also the running times of common join algorithms are known:

**Nested Loop Join (NLJoin):** O(n log m)

**Merge Join (MSJoin):** O(n+m)

**Hash Join (HSJoin):** O(N* hc +m *hm+J)

Summarized executing a query with double cluster size can not benefit of the entire additional resources with linear scale out. A linear scale out would result in running times with O(n log n) complexity. We estimate that our queries are running with more than O(n log n) complexity.

## IBM DB2

The following measurements show the results of IBM DB2 DPF:

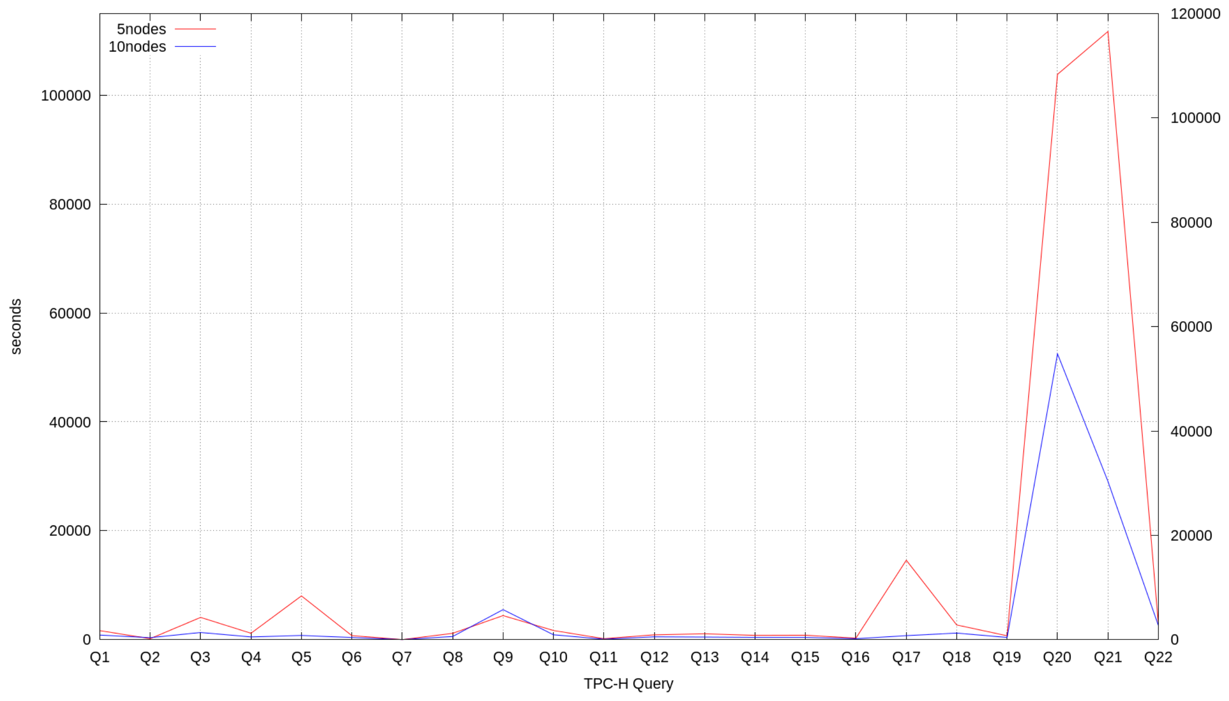
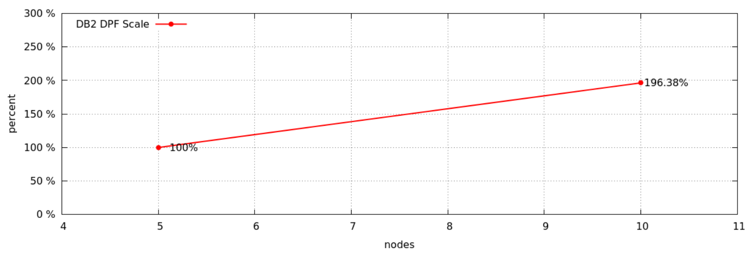


Figure 5.2.: IBM DB2 Measurements

A good scale out behavior for queries Q5, Q17, Q20 and Q21 is obtained. With using 10 nodes each node has half of the data stored on it as with 5 nodes. As a result all query operations on the nodes are issued over half of the data than with 5 nodes. Disadvantage is that within a broadcast over all nodes, there are more messages sent across the network. Query Q5, Q17 and Q20 benefit the most of the available additional resources with 10 nodes. A deeper look to their query access plans shows that they contain lot of sorting and HSJoin operations. As their complexity goes with faster times than O(n log n), we assume that the used sorting algorithm of IBM DB2 is an incremental sorting algorithm which runs on average with O($n^2$). This means that the sorting algorithm is more than two times faster on 10 nodes than on 5 nodes, because each node has half the amount of rows to sort during query execution.

Performance increase is illustrated by the following figure 5.3:



| Nodes | total time | percent |
|---|---|---|
| 5 | 137165.52 | 100% |
| 10 | 69846.66 | 196.38% |

Figure 5.3.: IBM DB2 scale-out

We identified Q17 and Q20 as outlier values, because their performance gain is more than two times better with 10 nodes than with 5. As a result of this we did not involve the execution times in the calculation of performance gain. While the measurements with 5 nodes represent 100 percent performance, the measurements with 10 nodes shows the performance gain, in percent. 200 percent would represent a linear scale out. A value below this 200 percent would represent a non-linear scaling. A value above this 200 percent represents a performance gain. We achieve a performance gain of 196 percent.

## Hive

The following values are the measurement times during the query execution in Hive:

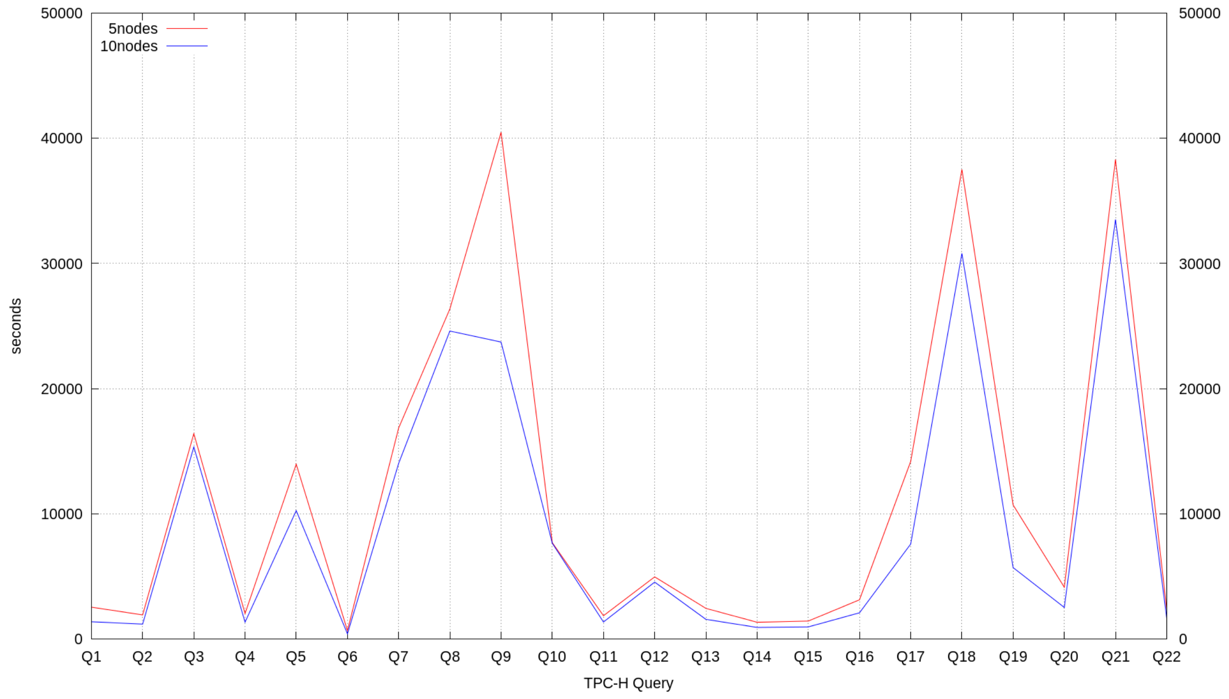


Figure 5.4.: Hive Measurements

Queries in our Hive experiments have not take much advantage of using more nodes for query execution. The scale out is not as good as expected. In addition the following figure shows the performance increase:

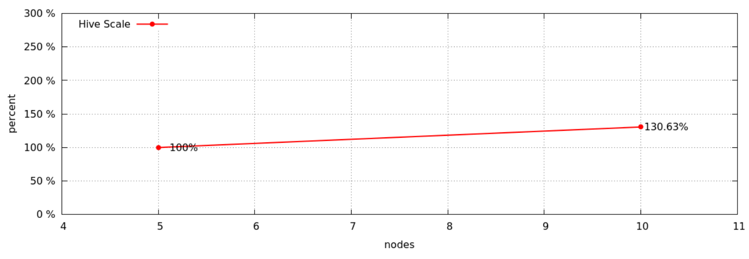


Figure 5.5.: Hive scale-out

| Nodes | total time | percent |
|---|---|---|
| 5 | 231449.50 | 100 % |
| 10 | 177168.60 | 130.63 % |

With 130 percent performance on 10 nodes, in comparison to 5 nodes, which represent 100 percent, we did not achieve a good-looking result. The Hive compiler generates multiple stages per query, which depend on each other. A following stage has to wait, until the previous ends. While all results of map tasks have to be merged, we observe many periods where only a handful reduce-tasks are merging the data. During this time, the cluster was no completely in use. The result of 130 percent gain could be better with an optimal tuning.

## IBM BigSQL

We get following measurements from IBM BigSQL:

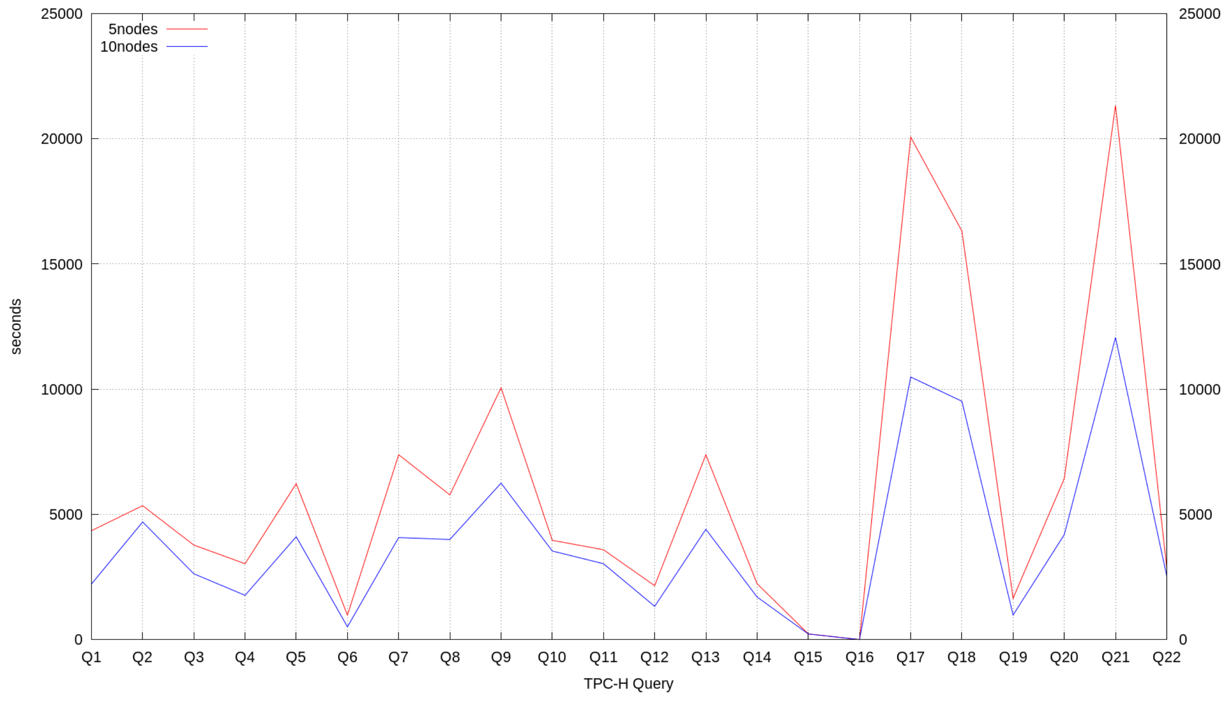
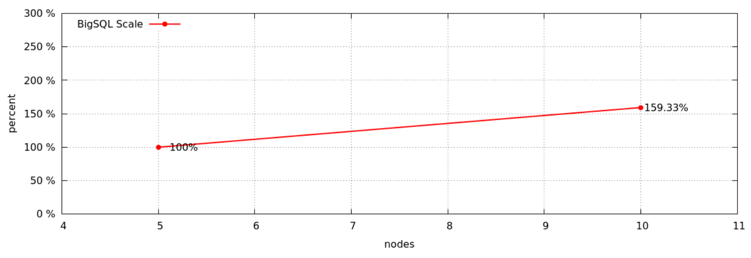


Figure 5.6.: BigSQL Measurements

In comparison to Hive, we see a much better resource usage while scaling out in IBM BigSQL. The query compiler is able to provide a better scale-out behavior. Long running queries scale great, while short running queries do not have much potential to scale. In addition the following figure shows the performance increase:



| Nodes | total time | percent |
|---|---|---|
| 5 | 127757.19 | 100 % |
| 10 | 80182.82 | 159.33 % |

Figure 5.7.: IBM BigSQL scale-out

Other than the figure 5.6 looks like, the performance scale-out is not that conspicuous. With around 160 percent, we do not reach a linear scale-out.

## 5.3. Limitations

Experiments were limited by time because this thesis lasts 14 weeks.
Numerous query tuning options exists for IBM DB2 database, we selected only a few of them to get better performance on query execution time. Despite the fact we spent a lot of time for index creation, for Q7 on IBM DB2 cluster with 5 nodes we did not get a result in a feasible time. The created indexes did not speed up query execution time for Q7 on 5 nodes. The IBM DB2 advise tool suggested another index with a different sort order than existing indexes on the tables. As we had already finished measurements with 10 nodes, we could not enable an additional index for executions on the 5 node cluster. As a result we did not involve Q7 in our measurement experiments, although IBM DB2 on 10 nodes could process the result of the query.
By executing the queries on IBM BigSQL we could not obtain results for Q16. In the short time left we did not find a suitable solution to solve the problems of using too much Java heap size during map or reduce task execution. Therefore we did not include values for executing Q16 on all three systems. As Q17 execution time was remarkable longer than execution in Hive, we analyzed the SQL statement and did a rewrite for this query. As the Hive implementation was faster, we tried to adapt the structure of the HiveQL statement in SQL. We had additional help from query files of the benchmark testing of Simon Harris [2]. Through rewrite of the query, we achieve a faster execution time of Q17.
Measurements were only done by cluster size, we could not increase the dataset size for the table data. Next higher scale-factor value of TPC-H benchmark is 3000. This value generate too much data for storing on our physical disks. In IBM DB2 environment the load of data sample with this factor will take too long, reorganization of table Lineitem could not be possible, because the reorg command usually is not done in-place (table data is copied during reorg command). As we store the table data in Hive into different types of tables, size of data could increase to more than two times, which is certainly too high for our physical disks. Next smaller value of scale factor is 300 which results in too small dataset sizes especially on the Hadoop system.

[2]Simon Harris, IBM Australia

# 6. Discussion

## 6.1. IBM DB2

### Strengths

IBM DB2 is a well known product and lot of best practices are available. During setting up and executing queries we got lot of assistance and support from the community. IBM DB2 also brings along a lot of administration tools. Partitioning tables simply result in better performance.

### Cautions

As already mentioned, IBM DB2 DPF can only perform best with co-located joins, design of database, selecting distribution keys and generating indexes could only be done with a known workload. There is no way of dynamic adjustment or tuning possibilities for unknown queries. DPF is also not fault tolerant. The IBM DB2 environment is more expensive and has to be installed on a fault-tolerant hardware environment. We also disregard the fact that IBM shows too less insights about the implementation of DB2.

## 6.2. Apache Hive

### Strengths

The Hive solution benefits from using different file-structures, which can be defined for the records of a table. With RCFile, it provides an approach for handle reads on files dynamic according to the used values for the query execution. Another import element of Hive is the collection of meta information in the Hive metastore. Hive writes the data into files and directories into the HDFS, which provides a clear view of where data is stored.

### Cautions

Tuning Hive queries is not as simple as one is used with RDBMS. Although Hive provides flexibility for diverse workloads, one first has to be familiar with the tuning opportunities. The reference and also community lack of best practices information about setting values of amount of map or reduce tasks. Hive generates Map-Reduce jobs with depending stages, which causes Hive to wait on completion of previous tasks.

## 6.3. IBM BigSQL

### Strengths

IBM BigSQL shines at fast executing point queries and fully implementation of the SQL Select statement. At measurement BigSQL, queries had no remarkable outliers and processing was really fast without use of additional tuning options. With the IBM BigInsights platform, IBM has released an entire package for extending the existing date warehouse solution for processing Big Data.

### Cautions

As disadvantage we criticize that IBM BigSQL is only available within the IBM BigInsights distribution. At the moment it is not possible to take IBM BigSQL and configure it in another Hadoop distribution. Also the dependencies of underlying components can be a problem, because when new releases of the components appear, IBM take some time to integrate them into their platform.

## 6.4. Presto

Presto is developed by Facebook, and is only tested against the Cloudera Hadoop Distribution (CDH). For that reason Presto was not working on our IBM BigInsights cluster in version 0.55-snapshot and we could only test it on the free available and fully configured virtual machine from Cloudera, CDH4. That results in measurements which are not comparable to our hardware cluster.

### Strengths

With Presto, Facebook avoids using the Map-Reduce-part of Hadoop. The setup of a Map-Reduce-job, mostly starting the JVM, needs a lot of time which is not necessary. Presto uses parts of the Hadoop ecosystem, but beware of generating Map-Reduce jobs. Other vendors, like Cloudera, are also working on products which beware the Map-Reduce jobs, called Impala at Cloudera. Presto has their own query execution engine (formerly known as worker) which is located on the HDFS datanodes, to avoid network bottlenecks. It is the essential way to be faster with query execution time.

## 6.5. Conclusion

We started with IBM DB2, because it was unknown for us, but we knew there would be a great community, with deep know-how. IBM DB2 provides a lot of tuning options, for various of use cases. IBM DB2 could be adjusted on a specific use case perfectly, but it has negative impacts in a diverse use case environment. That is the environment where IBM BigSQL scores.
Hive made the basic steps to enable SQL-selects on the Hadoop system and provides the metastore, which is used by the other systems. With all the possible tuning options, Hive demonstrates the ability of the Hadoop environment to adapt a workload. It provides support for additional

ideas around compression and file structure. As the negative part of Hive shows, it generates too much overhead while working off, the self defined dependent query-stages. At this point, we see an improvement, which is used by Presto, by using its own workers apart of the Map-Reduce environment. Also tuning the queries at Hive is time consuming and asks for deeper knowledge of adjusting amount of map and reduce tasks and also of setting correct HDFS block size.
IBM BigSQL has an impressive query compiler, and could create a good strategy for executing the work on query execution time. This forces a better overall executing time of diverse workload, other than in IBM DB2 environments where designing of database and tuning option depend on the known workload.
With that background, we think it is worth build a data warehouse based on the Hadoop platform as an extension to an existing DWH. Data for realtime queries performs best on traditional RDBMS, while a Hadoop based DWH could be used as a secondary store with maximum ability to scale and with an enormous flexibility because of schema on read strategy.

# A. DB2 Database Configuration

## A.1. DB2 Registry Settings

```
DB2_PARRALEL_IO=*
LOFGILSIZ=16384
LOGPRIMARY=20
LOGSECOND=20
```

## A.2. DB2 Database Creation Script

Listing A.1: DB2 Create Tables

```
-- This CLP file was created using DB2LOOK Version "10.5"
-- Timestamp: Fri 15 Nov 2013 11:03:07 AM CET
-- Database Name: TPCH
-- Database Manager Version: DB2/LINUXX8664 Version 10.5.1
-- Database Codepage: 1208
-- Database Collating Sequence is: IDENTITY


CONNECT TO TPCH;

------------------------------------------------
-- DDL Statements for Schemas
------------------------------------------------

-- Running the DDL below will explicitly create a schema in the
-- new database that corresponds to an implicitly created schema
-- in the original database.

CREATE SCHEMA "DB2INST1";

------------------------------------------------
-- DDL Statements for Table "DB2INST1"."NATION_ORG"
------------------------------------------------

CREATE TABLE "DB2INST1"."NATION_ORG"  (
       "N_NATIONKEY" INTEGER NOT NULL ,
       "N_NAME" CHAR(25 OCTETS) ,
       "N_REGIONKEY" INTEGER NOT NULL ,
       "N_COMMENT" VARCHAR(152 OCTETS) )
      IN "TBS_SMALL" NOT LOGGED INITIALLY
      ORGANIZE BY ROW;


-- DDL Statements for Primary Key on Table "DB2INST1"."NATION_ORG"

ALTER TABLE "DB2INST1"."NATION_ORG"
```

```
  ADD PRIMARY KEY
    ("N_NATIONKEY");

------------------------------------------------
-- DDL Statements for Table "DB2INST1"."REGION_ORG"
------------------------------------------------

CREATE TABLE "DB2INST1"."REGION_ORG"  (
      "R_REGIONKEY" INTEGER NOT NULL ,
      "R_NAME" CHAR(25 OCTETS) ,
      "R_COMMENT" VARCHAR(152 OCTETS) )
     IN "TBS_SMALL" NOT LOGGED INITIALLY
     ORGANIZE BY ROW;


-- DDL Statements for Primary Key on Table "DB2INST1"."REGION_ORG"

ALTER TABLE "DB2INST1"."REGION_ORG"
  ADD PRIMARY KEY
    ("R_REGIONKEY");


------------------------------------------------
-- DDL Statements for Table "DB2INST1"."NATION"
------------------------------------------------
SET CURRENT SCHEMA = "DB2INST1";
SET CURRENT PATH = "SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","DB2INST1";

CREATE TABLE NATION AS ( SELECT * FROM NATION_org) DATA INITIALLY DEFERRED REFRESH
   DEFERRED ENABLE QUERY OPTIMIZATION MAINTAINED BY SYSTEM DISTRIBUTE BY
   REPLICATION IN "TBS_LARGE"     ORGANIZE BY ROW;


ALTER TABLE "DB2INST1"."NATION" DEACTIVATE ROW ACCESS CONTROL;

REFRESH TABLE "DB2INST1"."NATION";

------------------------------------------------
-- DDL Statements for Table "DB2INST1"."REGION"
------------------------------------------------
 SET CURRENT SCHEMA = "DB2INST1";
SET CURRENT PATH = "SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","DB2INST1";

CREATE TABLE REGION AS ( SELECT * FROM REGION_org) DATA INITIALLY DEFERRED REFRESH
   DEFERRED ENABLE QUERY OPTIMIZATION MAINTAINED BY SYSTEM DISTRIBUTE BY
   REPLICATION IN "TBS_LARGE"     ORGANIZE BY ROW;


ALTER TABLE "DB2INST1"."REGION" DEACTIVATE ROW ACCESS CONTROL;

REFRESH TABLE "DB2INST1"."REGION";

------------------------------------------------
-- DDL Statements for Table "DB2INST1"."PART"
------------------------------------------------


CREATE TABLE "DB2INST1"."PART"  (
      "P_PARTKEY" BIGINT NOT NULL ,
      "P_NAME" VARCHAR(55 OCTETS) ,
      "P_MFGR" CHAR(25 OCTETS) ,
      "P_BRAND" CHAR(10 OCTETS) ,
      "P_TYPE" VARCHAR(25 OCTETS) ,
      "P_SIZE" INTEGER ,
      "P_CONTAINER" CHAR(10 OCTETS) ,
      "P_RETAILPRICE" DECIMAL(13,2) ,
      "P_COMMENT" VARCHAR(23 OCTETS) )
     DISTRIBUTE BY HASH("P_PARTKEY")
       IN "TBS_LARGE" NOT LOGGED INITIALLY
     ORGANIZE BY ROW;

------------------------------------------------
-- DDL Statements for Table "DB2INST1"."SUPPLIER"
------------------------------------------------


CREATE TABLE "DB2INST1"."SUPPLIER"  (
      "S_SUPPKEY" BIGINT NOT NULL ,
      "S_NAME" CHAR(25 OCTETS) ,
      "S_ADDRESS" VARCHAR(40 OCTETS) ,
      "S_NATIONKEY" BIGINT NOT NULL ,
      "S_PHONE" CHAR(15 OCTETS) ,
      "S_ACCTBAL" DECIMAL(13,2) ,
      "S_COMMENT" VARCHAR(101 OCTETS) )
     DISTRIBUTE BY HASH("S_SUPPKEY")
       IN "TBS_LARGE" NOT LOGGED INITIALLY
     ORGANIZE BY ROW;

------------------------------------------------
-- DDL Statements for Table "DB2INST1"."CUSTOMER"
------------------------------------------------


CREATE TABLE "DB2INST1"."CUSTOMER"  (
      "C_CUSTKEY" BIGINT NOT NULL ,
      "C_NAME" VARCHAR(25 OCTETS) ,
      "C_ADDRESS" VARCHAR(40 OCTETS) ,
      "C_NATIONKEY" BIGINT NOT NULL ,
      "C_PHONE" CHAR(15 OCTETS) ,
      "C_ACCTBAL" DECIMAL(13,2) ,
      "C_MKTSEGMENT" CHAR(10 OCTETS) ,
      "C_COMMENT" VARCHAR(117 OCTETS) )
     DISTRIBUTE BY HASH("C_CUSTKEY")
       IN "TBS_LARGE" NOT LOGGED INITIALLY
     ORGANIZE BY ROW;

------------------------------------------------
-- DDL Statements for Table "DB2INST1"."ORDERS"
------------------------------------------------


CREATE TABLE "DB2INST1"."ORDERS"  (
      "O_ORDERKEY" BIGINT NOT NULL ,
      "O_CUSTKEY" BIGINT NOT NULL ,
      "O_ORDERSTATUS" CHAR(1 OCTETS) ,
      "O_TOTALPRICE" DECIMAL(13,2) ,
      "O_ORDERDATE" DATE ,
      "O_ORDERPRIORITY" CHAR(15 OCTETS) ,
      "O_CLERK" CHAR(15 OCTETS) ,
      "O_SHIPPRIORITY" INTEGER ,
      "O_COMMENT" VARCHAR(79 OCTETS) )
     DISTRIBUTE BY HASH("O_ORDERKEY")
       IN "TBS_LARGE" NOT LOGGED INITIALLY
     ORGANIZE BY ROW;


------------------------------------------------
-- DDL Statements for Table "DB2INST1"."LINEITEM"
------------------------------------------------


CREATE TABLE "DB2INST1"."LINEITEM"  (
      "L_ORDERKEY" BIGINT NOT NULL ,
      "L_PARTKEY" BIGINT NOT NULL ,
      "L_SUPPKEY" BIGINT NOT NULL ,
      "L_LINENUMBER" INTEGER NOT NULL ,
      "L_QUANTITY" DECIMAL(13,2) ,
      "L_EXTENDEDPRICE" DECIMAL(13,2) ,
      "L_DISCOUNT" DECIMAL(13,2) ,
      "L_TAX" DECIMAL(13,2) ,
      "L_RETURNFLAG" CHAR(1 OCTETS) ,
      "L_LINESTATUS" CHAR(1 OCTETS) ,
      "L_SHIPDATE" DATE ,
      "L_COMMITDATE" DATE ,
      "L_RECEIPTDATE" DATE ,
      "L_SHIPINSTRUCT" CHAR(25 OCTETS) ,
      "L_SHIPMODE" CHAR(10 OCTETS) ,
      "L_COMMENT" VARCHAR(44 OCTETS) )
     DISTRIBUTE BY HASH("L_ORDERKEY")
       IN "TBS_LARGE" NOT LOGGED INITIALLY
     ORGANIZE BY ROW;

ALTER TABLE "DB2INST1"."NATION_ORG"
  ADD CONSTRAINT "NATION_FK1" FOREIGN KEY
    ("N_REGIONKEY")
  REFERENCES "DB2INST1"."REGION_ORG"
    ("R_REGIONKEY")
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
  ENFORCED
  ENABLE QUERY OPTIMIZATION;

----------------------------
-- DDL Statements for Views
----------------------------
SET CURRENT SCHEMA = "DB2INST1";
SET CURRENT PATH = "SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","DB2INST1";
create view c_orders(c_custkey,c_count) as select c_custkey, count(o_orderkey)
from customer left outer join orders on c_custkey = o_custkey and o_comment
not like '%special%requests%' group by c_custkey;

SET CURRENT SCHEMA = "DB2INST1";
SET CURRENT PATH = "SYSIBM","SYSFUN","SYSPROC","SYSIBMADM","DB2INST1";
create view revenue as select l_suppkey as supplier_no, sum(l_extendedprice
* (1 - l_discount)) as total_revenue from lineitem where l_shipdate >=
date ('1996-01-01') and l_shipdate < date ('1996-04-01') group by l_suppkey;

------------------------------------------------
-- Compression
------------------------------------------------
```

```
ALTER TABLE NATION_ORG COMPRESS YES STATIC;
ALTER TABLE REGION_ORG COMPRESS YES STATIC;
ALTER TABLE PART COMPRESS YES STATIC;
ALTER TABLE SUPPLIER COMPRESS YES STATIC;
ALTER TABLE PARTSUPP COMPRESS YES STATIC;
ALTER TABLE CUSTOMER COMPRESS YES STATIC;
ALTER TABLE ORDERS COMPRESS YES STATIC;
ALTER TABLE LINEITEM COMPRESS YES STATIC;

COMMIT WORK;

CONNECT RESET;

TERMINATE;
```

## A.3. DB2 Database Index Script

Listing A.2: DB2 Create Indexes

```
-- DDL Statements for Indexes on Table "DB2INST1"."NATION"

CREATE INDEX "DB2INST1"."IDX1310281446341" ON "DB2INST1"."NATION"
    ("N_NAME" ASC,  "N_NATIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."NATION"

CREATE INDEX "DB2INST1"."IDX1311111743520" ON "DB2INST1"."NATION"
    ("N_REGIONKEY" ASC, "N_NAME" ASC, "N_NATIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."NATION"

CREATE INDEX "DB2INST1"."N_NATIONKEY_INDX" ON "DB2INST1"."NATION"
    ("N_NATIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."NATION"

CREATE INDEX "DB2INST1"."N_REGIONKEY_INDX" ON "DB2INST1"."NATION"
    ("N_REGIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."REGION"

CREATE INDEX "DB2INST1"."IDX1311111742470" ON "DB2INST1"."REGION"
    ("R_NAME" ASC, "R_REGIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."REGION"

CREATE INDEX "DB2INST1"."R_REGIONKEY_INDX" ON "DB2INST1"."REGION"
    ("R_REGIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."PART"


CREATE INDEX "DB2INST1"."IDX1311081125300" ON "DB2INST1"."PART"
    ("P_NAME" ASC, "P_PARTKEY" DESC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."PART"

CREATE INDEX "DB2INST1"."PARTKEY_INDX" ON "DB2INST1"."PART"
    ("P_PARTKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."SUPPLIER"

CREATE INDEX "DB2INST1"."IDX1311081126360" ON "DB2INST1"."SUPPLIER"
    ("S_SUPPKEY" ASC, "S_NAME" ASC, "S_ADDRESS" ASC, "S_NATIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."SUPPLIER"

CREATE INDEX "DB2INST1"."IDX1311081249170" ON "DB2INST1"."SUPPLIER"
    ("S_SUPPKEY" ASC, "S_NATIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."SUPPLIER"

CREATE INDEX "DB2INST1"."IDX1311111743040" ON "DB2INST1"."SUPPLIER"
    ("S_NATIONKEY" ASC, "S_SUPPKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS YES INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."SUPPLIER"

CREATE INDEX "DB2INST1"."IDX1311111903070" ON "DB2INST1"."SUPPLIER"
    ("S_NATIONKEY" ASC, "S_SUPPKEY" DESC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS YES INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."SUPPLIER"


CREATE INDEX "DB2INST1"."S_NATIONKEY_INDX" ON "DB2INST1"."SUPPLIER"
    ("S_NATIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."SUPPLIER"

CREATE INDEX "DB2INST1"."S_SUPPKEY_INDX" ON "DB2INST1"."SUPPLIER"
    ("S_SUPPKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;
```

```
-- DDL Statements for Indexes on Table "DB2INST1"."PARTSUPP"

CREATE INDEX "DB2INST1"."IDX1311081125110" ON "DB2INST1"."PARTSUPP"
    ("PS_PARTKEY" ASC, "PS_SUPPKEY" ASC, "PS_AVAILQTY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO  INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."PARTSUPP"

CREATE INDEX "DB2INST1"."PS_PARTKEY_INDX" ON "DB2INST1"."PARTSUPP"
    ("PS_PARTKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."PARTSUPP"

CREATE INDEX "DB2INST1"."PS_SUPPKEY_INDX" ON "DB2INST1"."PARTSUPP"
    ("PS_SUPPKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."CUSTOMER"

CREATE INDEX "DB2INST1"."C_CUSTKEY_INDX" ON "DB2INST1"."CUSTOMER"
    ("C_CUSTKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."CUSTOMER"

CREATE INDEX "DB2INST1"."C_NATIONKEY_INDX" ON "DB2INST1"."CUSTOMER"
    ("C_NATIONKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."CUSTOMER"

CREATE INDEX "DB2INST1"."IDX1311111743470" ON "DB2INST1"."CUSTOMER"
    ("C_NATIONKEY" ASC, "C_CUSTKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS YES INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."ORDERS"

CREATE INDEX "DB2INST1"."IDX1311081250400" ON "DB2INST1"."ORDERS"
    ("O_ORDERKEY" ASC, "O_ORDERDATE" DESC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."ORDERS"

CREATE INDEX "DB2INST1"."IDX1311111743350" ON "DB2INST1"."ORDERS"
    ("O_ORDERDATE" ASC, "O_ORDERKEY" ASC, "O_CUSTKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS YES INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."ORDERS"

CREATE INDEX "DB2INST1"."O_CUSTKEY_INDX" ON "DB2INST1"."ORDERS"
    ("O_CUSTKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."ORDERS"

CREATE INDEX "DB2INST1"."O_ORDERKEY_INDX" ON "DB2INST1"."ORDERS"
    ("O_ORDERKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."LINEITEM"

CREATE INDEX "DB2INST1"."IDX1311081125330" ON "DB2INST1"."LINEITEM"
    ("L_PARTKEY" ASC, "L_SUPPKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."LINEITEM"

CREATE INDEX "DB2INST1"."L_ORDERKEY_INDX" ON "DB2INST1"."LINEITEM"
    ("L_ORDERKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."LINEITEM"

CREATE INDEX "DB2INST1"."L_PARTKEY_INDX" ON "DB2INST1"."LINEITEM"
    ("L_PARTKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;

-- DDL Statements for Indexes on Table "DB2INST1"."LINEITEM"

CREATE INDEX "DB2INST1"."L_SUPPKEY_INDX" ON "DB2INST1"."LINEITEM"
    ("L_SUPPKEY" ASC)
    COLLECT SAMPLED DETAILED STATISTICS
    COMPRESS NO INCLUDE NULL KEYS ALLOW REVERSE SCANS;
```

# B. Hive Database Configuration

## B.1. Database Creation Script

Listing B.1: internal Hive Tables for Query results

```
CREATE TABLE IF NOT EXISTS q1_pricing_summary_report
(L_RETURNFLAG STRING, L_LINESTATUS STRING, SUM_QTY DOUBLE,
SUM_BASE_PRICE DOUBLE, SUM_DISC_PRICE DOUBLE, SUM_CHARGE DOUBLE,
AVE_QTY DOUBLE, AVE_PRICE DOUBLE, AVE_DISC DOUBLE, COUNT_ORDER INT);

CREATE TABLE IF NOT EXISTS q2_minimum_cost_supplier_tmp1
(s_acctbal double, s_name string, n_name string, p_partkey int,
ps_supplycost double, p_mfgr string, s_address string,
s_phone string, s_comment string);

CREATE TABLE IF NOT EXISTS q2_minimum_cost_supplier_tmp2
(p_partkey int, ps_min_supplycost double);

CREATE TABLE IF NOT EXISTS q2_minimum_cost_supplier
(s_acctbal double, s_name string, n_name string, p_partkey int,
 p_mfgr string, s_address string, s_phone string, s_comment string);

CREATE TABLE IF NOT EXISTS q3_shipping_priority
(l_orderkey int, revenue double, o_orderdate string, o_shippriority int);

CREATE TABLE IF NOT EXISTS q4_order_priority_tmp
(O_ORDERKEY INT);

CREATE TABLE IF NOT EXISTS q4_order_priority
(O_ORDERPRIORITY STRING, ORDER_COUNT INT);

CREATE TABLE IF NOT EXISTS q5_local_supplier_volume
(N_NAME STRING, REVENUE DOUBLE);

CREATE TABLE IF NOT EXISTS q6_forecast_revenue_change
(revenue double);

CREATE TABLE IF NOT EXISTS q7_volume_shipping
(supp_nation string, cust_nation string, l_year int, revenue double);

CREATE TABLE IF NOT EXISTS q7_volume_shipping_tmp
(supp_nation string, cust_nation string, s_nationkey int, c_nationkey int);

CREATE TABLE IF NOT EXISTS q8_national_market_share
(o_year string, mkt_share double);

CREATE TABLE IF NOT EXISTS q9_product_type_profit
(nation string, o_year string, sum_profit double);

CREATE TABLE IF NOT EXISTS q10_returned_item
```

```
(c_custkey int, c_name string, revenue double, c_acctbal string,
 n_name string, c_address string, c_phone string, c_comment string);

CREATE TABLE IF NOT EXISTS q11_important_stock
(ps_partkey INT, value DOUBLE);

CREATE TABLE IF NOT EXISTS q11_part_tmp
(ps_partkey int, part_value double);

CREATE TABLE IF NOT EXISTS q11_sum_tmp
(total_value double);

CREATE TABLE IF NOT EXISTS q12_shipping
(l_shipmode string, high_line_count double, low_line_count double);

CREATE TABLE IF NOT EXISTS q13_customer_distribution
(c_count int, custdist int);

CREATE TABLE IF NOT EXISTS q14_promotion_effect
(promo_revenue double);

CREATE TABLE IF NOT EXISTS revenue
(supplier_no int, total_revenue double);

CREATE TABLE IF NOT EXISTS max_revenue
(max_revenue double);

CREATE TABLE IF NOT EXISTS q15_top_supplier
(s_suppkey int, s_name string, s_address string,
 s_phone string, total_revenue double);

CREATE TABLE IF NOT EXISTS q16_parts_supplier_relationship
(p_brand string, p_type string, p_size int, supplier_cnt int);

CREATE TABLE IF NOT EXISTS q16_tmp
(p_brand string, p_type string, p_size int, ps_suppkey int);

CREATE TABLE IF NOT EXISTS supplier_tmp
(s_suppkey int);

CREATE TABLE IF NOT EXISTS q17_small_quantity_order_revenue
(avg_yearly double);

CREATE TABLE IF NOT EXISTS lineitem_tmp
(t_partkey int, t_avg_quantity double);

CREATE TABLE IF NOT EXISTS q18_tmp
(l_orderkey int, t_sum_quantity double);

CREATE TABLE IF NOT EXISTS q18_large_volume_customer
(c_name string, c_custkey int, o_orderkey int,
 o_orderdate string, o_totalprice double, sum_quantity double);

CREATE TABLE IF NOT EXISTS q19_discounted_revenue
(revenue double);

CREATE TABLE IF NOT EXISTS q20_tmp1
(p_partkey int);

CREATE TABLE IF NOT EXISTS q20_tmp2
```

```
(l_partkey int, l_suppkey int, sum_quantity double);

CREATE TABLE IF NOT EXISTS q20_tmp3
(ps_suppkey int, ps_availqty int, sum_quantity double);

CREATE TABLE IF NOT EXISTS q20_tmp4
(ps_suppkey int);

CREATE TABLE IF NOT EXISTS q20_potential_part_promotion
(s_name string, s_address string);

CREATE TABLE IF NOT EXISTS q21_tmp1
(l_orderkey int, count_suppkey int, max_suppkey int);

CREATE TABLE IF NOT EXISTS q21_tmp2
(l_orderkey int, count_suppkey int, max_suppkey int);

CREATE TABLE IF NOT EXISTS q21_suppliers_who_kept_orders_waiting
(s_name string, numwait int);

CREATE TABLE IF NOT EXISTS q22_customer_tmp
(c_acctbal double, c_custkey int, cntrycode string);

CREATE TABLE IF NOT EXISTS q22_customer_tmp1
(avg_acctbal double);

CREATE TABLE IF NOT EXISTS q22_orders_tmp
(o_custkey int);

CREATE TABLE IF NOT EXISTS q22_global_sales_opportunity
(cntrycode string, numcust int, totacctbal double);
```

Listing B.2: External Stage Hive Tables

```
CREATE EXTERNAL TABLE IF NOT EXISTS stage_lineitem
(L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT,
 L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE,
 L_RETURNFLAG STRING, L_LINESTATUS STRING, L_SHIPDATE STRING, L_COMMITDATE STRING,
 L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING, L_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/lineitem';

CREATE EXTERNAL TABLE IF NOT EXISTS stage_part
(P_PARTKEY INT, P_NAME STRING, P_MFGR STRING, P_BRAND STRING, P_TYPE STRING,
 P_SIZE INT, P_CONTAINER STRING, P_RETAILPRICE DOUBLE, P_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/part';

CREATE EXTERNAL TABLE IF NOT EXISTS stage_customer
(C_CUSTKEY INT, C_NAME STRING, C_ADDRESS STRING, C_NATIONKEY INT, C_PHONE STRING,
 C_ACCTBAL DOUBLE, C_MKTSEGMENT STRING, C_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/customer';

CREATE EXTERNAL TABLE IF NOT EXISTS stage_orders
(O_ORDERKEY INT, O_CUSTKEY INT, O_ORDERSTATUS STRING, O_TOTALPRICE DOUBLE,
O_ORDERDATE STRING, O_ORDERPRIORITY STRING, O_CLERK STRING, O_SHIPPRIORITY INT,
O_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/orders';
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS stage_supplier
(S_SUPPKEY INT, S_NAME STRING, S_ADDRESS STRING, S_NATIONKEY INT, S_PHONE STRING,
S_ACCTBAL DOUBLE, S_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/supplier';

CREATE EXTERNAL TABLE IF NOT EXISTS stage_nation
(N_NATIONKEY INT, N_NAME STRING, N_REGIONKEY INT, N_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/nation';

CREATE EXTERNAL TABLE IF NOT EXISTS stage_region
(R_REGIONKEY INT, R_NAME STRING, R_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/region';

CREATE EXTERNAL TABLE IF NOT EXISTS stage_partsupp
(PS_PARTKEY INT, PS_SUPPKEY INT, PS_AVAILQTY INT,
PS_SUPPLYCOST DOUBLE, PS_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE LOCATION '/user/biadmin/tpch/partsupp';
```

Listing B.3: RCFile Hive Tables

```
CREATE TABLE IF NOT EXISTS lineitem
(L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT,
 L_LINENUMBER INT, L_QUANTITY DOUBLE,
 L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE,
 L_RETURNFLAG STRING, L_LINESTATUS STRING, L_SHIPDATE STRING,
 L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING,
 L_SHIPMODE STRING, L_COMMENT STRING) STORED AS RCFile;

CREATE TABLE IF NOT EXISTS part
(P_PARTKEY INT, P_NAME STRING, P_MFGR STRING, P_BRAND STRING,
 P_TYPE STRING, P_SIZE INT, P_CONTAINER STRING, P_RETAILPRICE DOUBLE,
 P_COMMENT STRING) STORED AS RCFile;

CREATE TABLE IF NOT EXISTS customer
(C_CUSTKEY INT, C_NAME STRING, C_ADDRESS STRING, C_NATIONKEY INT,
 C_PHONE STRING, C_ACCTBAL DOUBLE, C_MKTSEGMENT STRING,
 C_COMMENT STRING) STORED AS RCFile;

CREATE TABLE IF NOT EXISTS orders
(O_ORDERKEY INT, O_CUSTKEY INT, O_ORDERSTATUS STRING, O_TOTALPRICE DOUBLE,
 O_ORDERDATE STRING, O_ORDERPRIORITY STRING, O_CLERK STRING,
 O_SHIPPRIORITY INT, O_COMMENT STRING) STORED AS RCFile;

CREATE TABLE IF NOT EXISTS supplier
(S_SUPPKEY INT, S_NAME STRING, S_ADDRESS STRING, S_NATIONKEY INT,
 S_PHONE STRING, S_ACCTBAL DOUBLE, S_COMMENT STRING) STORED AS RCFile;

CREATE TABLE IF NOT EXISTS nation
(N_NATIONKEY INT, N_NAME STRING,
 N_REGIONKEY INT, N_COMMENT STRING) STORED AS RCFile;

CREATE TABLE IF NOT EXISTS region
(R_REGIONKEY INT, R_NAME STRING,
 R_COMMENT STRING) STORED AS RCFile;
```

```
CREATE TABLE IF NOT EXISTS partsupp
PS_PARTKEY INT, PS_SUPPKEY INT, PS_AVAILQTY INT,
PS_SUPPLYCOST DOUBLE, PS_COMMENT STRING) STORED AS RCfile;
```

Listing B.4: Load RCFile Hive Tables

```
--Part
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE part SELECT * FROM stage_part;

--Partsupp
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE partsupp SELECT * FROM stage_partsupp;

--lineitem
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE lineitem SELECT * FROM stage_lineitem;

--orders
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE orders SELECT * FROM stage_orders;

--supplier
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE supplier SELECT * FROM stage_supplier;

--customer
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE customer SELECT * FROM stage_customer;

--nation
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE nation SELECT * FROM stage_nation;

--region
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
SET mapred.output.compress=true;
SET mapred.min.split.size=1024000000;
SET mapred.max.split.size=1024000000;
SET hive.merge.mapredfiles=true;
SET hive.merge.smallfiles.avgsize=1024000000;
SET hive.merge.size.per.task=1024000000;
SET mapred.min.split.size.per.node=1024000000;
SET mapred.min.split.size.per.rack=1024000000;

INSERT OVERWRITE TABLE region SELECT * FROM stage_region;
```

# C. TPC-H Queries

## Pricing Summary Report Query (Q1)

This query reports the amount of business that was billed, shipped, and returned.
The pricing summary report query provides a summary pricing report for all lineitems shipped as of a given date.

### SQL Statement

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
 sum(l_extendedprice) as sum_base_price,
 sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
 sum(l_extendedprice* (1-l_discount) *(1+l_tax)) as sum_charge,
 avg(l_quantity) as avg_qty,
 avg(l_extendedprice) as avg_price,
 avg(l_discount) as avg_disc,
 count(*) as count_order
from lineitem
where l_shipdate <= date( '1998-09-02')
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

### Hive Statement

```
INSERT OVERWRITE TABLE q1_pricing_summary_report
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY), SUM(L_EXTENDEDPRICE),
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)),
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)),
 AVG(L_QUANTITY), AVG(L_EXTENDEDPRICE), AVG(L_DISCOUNT), COUNT(1)
FROM lineitem
WHERE L_SHIPDATE<='1998-09-02'
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG, L_LINESTATUS;
```

### BigSQL Statement

```
select l_returnflag , l_linestatus , sum ( l_quantity ) as sum_qty ,
 sum ( l_extendedprice ) as sum_base_price ,
 sum ( l_extendedprice *(1 - l_discount ) ) as sum_disc_price ,
 sum ( l_extendedprice * (1 - l_discount ) *(1+ l_tax ) ) as sum_charge ,
 avg ( l_quantity ) as avg_qty ,
 avg ( l_extendedprice ) as avg_price ,
 avg ( l_discount ) as avg_disc ,
 count (*) as count_order
from lineitem
```

```
where l_shipdate <= '1998-09-02'
group by l_returnflag , l_linestatus
order by l_returnflag , l_linestatus;
```

## Minimum Cost Supplier Query (Q2)

This query finds which supplier should be selected to place an order for a given part in a given region.
The minimum cost supplier query finds, in a given region, for each part of a certain type and size, the supplier who can supply it at minimum cost. If several suppliers in that region offer the desired part type and size at the same (minimum) cost, the query lists the parts from suppliers with the 100 highest account balances.

### SQL Statement

```
select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from part, supplier, partsupp, nation, region
where p_partkey = ps_partkey
 and s_suppkey = ps_suppkey
 and p_size = 15
 and p_type like '%BRASS'
 and s_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and r_name = 'EUROPE'
 and ps_supplycost = (
  select min(ps_supplycost)
  from partsupp, supplier, nation, region
  where p_partkey = ps_partkey
   and s_suppkey = ps_suppkey
   and s_nationkey = n_nationkey
   and n_regionkey = r_regionkey
   and r_name = 'EUROPE' )
order by s_acctbal desc, n_name, s_name, p_partkey
fetch first 100 rows only;
```

### Hive Statement

```
insert overwrite table q2_minimum_cost_supplier_tmp1
select s.s_acctbal, s.s_name, n.n_name, p.p_partkey, ps.ps_supplycost,
 p.p_mfgr, s.s_address, s.s_phone, s.s_comment
from nation n join region r
 on n.n_regionkey = r.r_regionkey and r.r_name = 'EUROPE'
 join supplier s
 on s.s_nationkey = n.n_nationkey
 join partsupp ps
 on s.s_suppkey = ps.ps_suppkey
 join part p
 on p.p_partkey = ps.ps_partkey
  and p.p_size = 15 and p.p_type like '%BRASS' ;

insert overwrite table q2_minimum_cost_supplier_tmp2
select p_partkey, min(ps_supplycost)
from q2_minimum_cost_supplier_tmp1
group by p_partkey;
```

```
insert overwrite table q2_minimum_cost_supplier
select t1.s_acctbal, t1.s_name, t1.n_name, t1.p_partkey,
 t1.p_mfgr, t1.s_address, t1.s_phone, t1.s_comment
from q2_minimum_cost_supplier_tmp1 t1
 join q2_minimum_cost_supplier_tmp2 t2
  on t1.p_partkey = t2.p_partkey and t1.ps_supplycost = t2.ps_min_supplycost
order by s_acctbal desc, n_name, s_name, p_partkey
limit 100;
```

### BigSQL Statement

```
select s_acctbal , s_name , n_name , p_partkey , p_mfgr , s_address , s_phone ,
    s_comment
from part , supplier , partsupp , nation , region
where p_partkey = ps_partkey
 and s_suppkey = ps_suppkey
 and p_size = 15
 and p_type like '%␣BRASS'
 and s_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and r_name = 'EUROPE'
 and ps_supplycost = (
  select min ( ps_supplycost )
  from partsupp , supplier , nation , region
  where p_partkey = ps_partkey
   and s_suppkey = ps_suppkey
   and s_nationkey = n_nationkey
   and n_regionkey = r_regionkey
   and r_name = 'EUROPE' )
order by s_acctbal desc , n_name , s_name , p_partkey
fetch first 100 rows only ;
```

## Shipping Priority Query (Q3)

This query retrieves the 10 unshipped orders with the highest value.
The shipping priority query retrieves the shipping priority and potential revenue of the orders having the largest revenue among those that had not been shipped as of a given date. Only the 10 orders with the largest revenue are listed

### SQL Statement

```
select l_orderkey,  sum(l_extendedprice*(1-l_discount)) as revenue,
 o_orderdate, o_shippriority
from customer, orders, lineitem
where c_mktsegment = 'BUILDING'
 and c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and o_orderdate < date( '1995-03-15')
 and l_shipdate > date( '1995-03-15')
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate
fetch first 10 rows only;
```

### Hive Statement

```
Insert overwrite table q3_shipping_priority
select l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue,
 o_orderdate, o_shippriority
from customer c
 join orders o
  on c.c_mktsegment = 'BUILDING' and c.c_custkey = o.o_custkey
  join lineitem l
   on l.l_orderkey = o.o_orderkey
where o_orderdate < '1995-03-15'
 and l_shipdate > '1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate
limit 10;
```

### BigSQL Statement

```
select l_orderkey , sum ( l_extendedprice *(1 - l_discount ) ) as revenue ,
 o_orderdate , o_shippriority
from customer , orders , lineitem
where c_mktsegment = 'BUILDING'
 and c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and o_orderdate < '1995-03-15'
 and l_shipdate >  '1995-03-15'
group by l_orderkey , o_orderdate , o_shippriority
order by revenue desc , o_orderdate
fetch first 10 rows only ;
```

## Order Priority Checking Query (Q4)

This query determines how well the order priority system is working and gives an assessment of customer satisfaction.
The order priority checking query counts the number of orders ordered in a given quarter of a given year in which at least one lineitem was received by the customer later than its committed date. The query lists the count of such orders for each order priority.

### SQL Statement

```
select o_orderpriority, count(*) as order_count
from orders
where o_orderdate >= date('1993-07-01')
 and o_orderdate < date('1993-10-01')
 and exists (
  select *
  from lineitem
  where l_orderkey = o_orderkey
   and l_commitdate < l_receiptdate )
group by o_orderpriority
order by o_orderpriority;
```

### Hive Statement

```
INSERT OVERWRITE TABLE q4_order_priority_tmp
select DISTINCT l_orderkey
from lineitem
where l_commitdate < l_receiptdate;

INSERT OVERWRITE TABLE q4_order_priority
select o_orderpriority, count(1) as order_count
from orders o
 join q4_order_priority_tmp t
  on o.o_orderkey = t.o_orderkey
  and o.o_orderdate >= '1993-07-01' and o.o_orderdate < '1993-10-01'
group by o_orderpriority
order by o_orderpriority;
```

### BigSQL Statement

```
select o_orderpriority , count (*) as order_count
from orders
where o_orderdate >= '1993-07-01'
 and o_orderdate < '1993-10-01')
 and exists (
  select *
  from lineitem
  where l_orderkey = o_orderkey
   and l_commitdate < l_receiptdate )
group by o_orderpriority
order by o_orderpriority ;
```

## Local Supplier Volume Query (Q5)

This query lists the revenue volume done through local suppliers.
The local supplier volume query lists for each nation in a region the revenue volume that resulted from lineitem transactions in which the customer ordering parts and the supplier filling them were both within that nation. The query is run in order to determine whether to institute local distribution centers in a given region. The query considers only parts ordered in a given year.

### SQL Statement

```
select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from customer, orders, lineitem,
 supplier, nation, region
where c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and l_suppkey = s_suppkey
 and c_nationkey = s_nationkey
 and s_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and r_name = 'ASIA'
 and o_orderdate >= date('1994-01-01')
 and o_orderdate < date('1995-01-01')
group by n_name
order by revenue desc;
```

### Hive Statement

```
insert overwrite table q5_local_supplier_volume
select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from customer c join
 ( select n_name, l_extendedprice, l_discount, s_nationkey, o_custkey from orders o
     join
   ( select n_name, l_extendedprice, l_discount, l_orderkey, s_nationkey from
      lineitem l join
     ( select n_name, s_suppkey, s_nationkey from supplier s join
       ( select n_name, n_nationkey
          from nation n
           join region r
            on n.n_regionkey = r.r_regionkey and r.r_name = 'ASIA'
        ) n1 on s.s_nationkey = n1.n_nationkey
      ) s1 on l.l_suppkey = s1.s_suppkey
    ) l1 on l1.l_orderkey = o.o_orderkey and o.o_orderdate >= '1994-01-01'
    and o.o_orderdate < '1995-01-01'
 ) o1
 on c.c_nationkey = o1.s_nationkey and c.c_custkey = o1.o_custkey
group by n_name
order by revenue desc;
```

### BigSQL Statement

```
select n_name , sum ( l_extendedprice * (1 - l_discount ) ) as revenue
from customer , orders , lineitem ,
 supplier , nation , region
where c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and l_suppkey = s_suppkey
 and c_nationkey = s_nationkey
 and s_nationkey = n_nationkey
 and n_regionkey = r_regionkey
 and r_name = 'ASIA'
 and o_orderdate >= '1994-01-01'
 and o_orderdate < '1995-01-01'
group by n_name
order by revenue desc;
```

## Forecasting Revenue Change Query (Q6)

This query quantifies the amount of revenue increase that would have resulted from eliminating certain company-wide discounts in a given percentage range in a given year. Asking this type of "what if" query can be used to look for ways to increase revenues.
The forecasting revenue change query considers all the lineitems shipped in a given year with discounts between 0.05 and 0.07. The query lists the amount by which the total revenue would have increased if these discounts had been eliminated for lineitems with l_quantity less than quantity. Note that the potential revenue increase is equal to the sum of [l_extendedprice * l_discount] for all lineitems with discounts and quantities in the qualifying range.

### SQL Statement

```
select sum(l_extendedprice*l_discount) as revenue
from lineitem
where l_shipdate >= date('1994-01-01')
```

```
 and l_shipdate < date('1995-01-01')
 and l_discount between 0.05 and 0.07
 and l_quantity < 24;
```

### Hive Statement

```
insert overwrite table q6_forecast_revenue_change
select sum(l_extendedprice*l_discount) as revenue
from lineitem
where l_shipdate >= '1994-01-01'
 and l_shipdate < '1995-01-01'
 and l_discount >= 0.05 and l_discount <= 0.07
 and l_quantity < 24;
```

### BigSQL Statement

```
select sum(l_extendedprice*l_discount) as revenue
from lineitem
where l_shipdate >= '1994-01-01'
 and l_shipdate < '1995-01-01'
 and l_discount between 0.05 and 0.07
 and l_quantity < 24;
```

## Volume Shipping Query (Q7)

This query determines the value of goods shipped between certain nations to help in the re-negotiation of shipping contracts.
The volume shipping query finds, for two given nations, the gross discounted revenues derived from lineitems in which parts were shipped from a supplier in either nation to a customer in the other nation during 1995 and 1996. The query lists the supplier nation, the customer nation, the year, and the revenue from shipments that took place in that year.

### SQL Statement

```
select supp_nation, cust_nation, l_year, sum(volume) as revenue
from ( select
  n1.n_name as supp_nation,
  n2.n_name as cust_nation,
  extract(year from l_shipdate) as l_year, l_extendedprice * (1 - l_discount) as
      volume
  from supplier, lineitem, orders,
   customer, nation n1, nation n2
  where s_suppkey = l_suppkey
   and o_orderkey = l_orderkey
   and c_custkey = o_custkey
   and s_nationkey = n1.n_nationkey
   and c_nationkey = n2.n_nationkey
   and (
         (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
      or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE') )
   and l_shipdate between date('1995-01-01') and date('1996-12-31')
     ) as shipping
group by supp_nation, cust_nation, l_year
order by supp_nation, cust_nation, l_year;
```

## Hive Statement

```
insert overwrite table q7_volume_shipping_tmp
select *
from (
 select n1.n_name as supp_nation, n2.n_name as cust_nation,
  n1.n_nationkey as s_nationkey, n2.n_nationkey as c_nationkey
  from nation n1 join nation n2
   on n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY'
 UNION ALL
 select n1.n_name as supp_nation, n2.n_name as cust_nation,
  n1.n_nationkey as s_nationkey, n2.n_nationkey as c_nationkey
  from nation n1 join nation n2
   on n2.n_name = 'FRANCE' and n1.n_name = 'GERMANY'
) a;

insert overwrite table q7_volume_shipping
select supp_nation, cust_nation, l_year, sum(volume) as revenue
from (
 select supp_nation, cust_nation, year(l_shipdate) as l_year,
      l_extendedprice * (1 - l_discount) as volume
   from q7_volume_shipping_tmp t join
    ( select l_shipdate, l_extendedprice, l_discount, c_nationkey, s_nationkey
       from supplier s join
        ( select l_shipdate, l_extendedprice, l_discount, l_suppkey, c_nationkey
          from customer c join
           ( select l_shipdate, l_extendedprice, l_discount, l_suppkey, o_custkey
             from orders o join lineitem l
              on o.o_orderkey = l.l_orderkey
               and l.l_shipdate >= '1995-01-01'
               and l.l_shipdate <= '1996-12-31'
           ) l1 on c.c_custkey = l1.o_custkey
        ) l2 on s.s_suppkey = l2.l_suppkey
    ) l3 on l3.c_nationkey = t.c_nationkey and l3.s_nationkey = t.s_nationkey
) shipping
group by supp_nation, cust_nation, l_year
order by supp_nation, cust_nation, l_year;
```

## BigSQL Statement

```
select supp_nation, cust_nation, l_year, sum(volume) as revenue
from ( select
  n1.n_name as supp_nation,
  n2.n_name as cust_nation,
  substr(l_shipdate,1,4) as l_year,
  l_extendedprice * (1 - l_discount) as volume
  from supplier, lineitem, orders,
   customer, nation n1, nation n2
  where s_suppkey = l_suppkey
   and o_orderkey = l_orderkey
   and c_custkey = o_custkey
   and s_nationkey = n1.n_nationkey
   and c_nationkey = n2.n_nationkey
   and (
         (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
      or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE') )
   and l_shipdate between '1995-01-01' and '1996-12-31' ) as shipping
group by supp_nation, cust_nation, l_year
order by supp_nation, cust_nation, l_year;
```

## National Market Share Query (Q8)

This query determines how the market share of a given nation within a given region has changed over two years for a given part type.
The market share for a given nation within a given region is defined as the fraction of the revenue, the sum of [l_extendedprice * (1-l_discount)], from the products of a specified type in that region that was supplied by suppliers from the given nation. The query determines this for the years 1995 and 1996 presented in this order.

### SQL Statement

```
select o_year, sum(case
     when nation = 'BRAZIL'
         then volume
         else 0 end) / sum(volume) as mkt_share
from ( select year(o_orderdate) as o_year,
        l_extendedprice * (1-l_discount) as volume,
        n2.n_name as nation
       from part, supplier, lineitem, orders,
        customer, nation n1, nation n2, region
       where p_partkey = l_partkey
        and s_suppkey = l_suppkey
        and l_orderkey = o_orderkey
        and o_custkey = c_custkey
        and c_nationkey = n1.n_nationkey
        and n1.n_regionkey = r_regionkey
        and r_name = 'AMERICA'
        and s_nationkey = n2.n_nationkey
        and o_orderdate between date('1995-01-01') and date('1996-12-31')
        and p_type = 'ECONOMY␣ANODIZED␣STEEL' ) as all_nations
group by o_year
order by o_year;
```

### Hive Statement

```
insert overwrite table q8_national_market_share
select o_year, sum(case
   when nation = 'BRAZIL'
      then volume
      else 0.0 end) / sum(volume) as mkt_share
from ( select year(o_orderdate) as o_year,
        l_extendedprice * (1-l_discount) as volume,
        n2.n_name as nation
       from nation n2 join
       ( select o_orderdate, l_discount, l_extendedprice, s_nationkey
         from supplier s join
         ( select o_orderdate, l_discount, l_extendedprice, l_suppkey
           from part p join
           ( select o_orderdate, l_partkey, l_discount, l_extendedprice, l_suppkey
             from lineitem l join
             ( select o_orderdate, o_orderkey
               from orders o join
               ( select c.c_custkey
                 from customer c join
                 ( select n1.n_nationkey
                   from nation n1 join region r
                    on n1.n_regionkey = r.r_regionkey and r.r_name = 'AMERICA'
                  ) n11 on c.c_nationkey = n11.n_nationkey
                ) c1 on c1.c_custkey = o.o_custkey
              ) o1 on l.l_orderkey = o1.o_orderkey
                and o1.o_orderdate >= '1995-01-01'
                and o1.o_orderdate < '1996-12-31'
            ) l1 on p.p_partkey = l1.l_partkey
              and p.p_type = 'ECONOMY␣ANODIZED␣STEEL'
         ) p1 on s.s_suppkey = p1.l_suppkey
       ) s1 on s1.s_nationkey = n2.n_nationkey
) all_nation
group by o_year
order by o_year;
```

## BigSQL Statement

```
select o_year, sum(case
    when nation = 'BRAZIL'
    then volume
    else 0 end) / sum(volume) as mkt_share
from ( select substr(o_orderdate,1,4) as o_year,
        l_extendedprice * (1-l_discount) as volume,
        n2.n_name as nation
       from part, supplier, lineitem, orders,
        customer, nation n1, nation n2, region
       where p_partkey = l_partkey
        and s_suppkey = l_suppkey
        and l_orderkey = o_orderkey
        and o_custkey = c_custkey
        and c_nationkey = n1.n_nationkey
        and n1.n_regionkey = r_regionkey
        and r_name = 'AMERICA'
        and s_nationkey = n2.n_nationkey
        and o_orderdate between '1995-01-01' and '1996-12-31'
        and p_type = 'ECONOMY␣ANODIZED␣STEEL' ) as all_nations
group by o_year
order by o_year;
```

## Product Type Profit Measure Query (Q9)

This query determines how much profit is made on a given line of parts, broken out by supplier nation and year.
The product type profit measure query finds, for each nation and each year, the profit for all parts ordered in that year that contain a specified substring in their names and that were filled by a supplier in that nation.

## SQL Statement

```
select nation, o_year, sum(amount) as sum_profit
from ( select n_name as nation,
        year(o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
       from part, supplier, lineitem,
        partsupp , orders , nation
       where s_suppkey = l_suppkey
        and ps_suppkey = l_suppkey
        and ps_partkey = l_partkey
        and p_partkey = l_partkey
        and o_orderkey = l_orderkey
        and s_nationkey = n_nationkey
        and p_name like '%green%' ) as profit
group by nation , o_year
order by nation , o_year desc;
```

## Hive Statement

```
insert overwrite table q9_product_type_profit
select nation , o_year , sum(amount) as sum_profit
from (
 select n_name as nation , year(o_orderdate) as o_year ,
  l_extendedprice * (1 - l_discount) -  ps_supplycost * l_quantity as amount
 from orders o join
 ( select l_extendedprice , l_discount , l_quantity , l_orderkey , n_name ,
     ps_supplycost
   from part p join
   ( select l_extendedprice , l_discount , l_quantity , l_partkey , l_orderkey ,
      n_name , ps_supplycost
     from partsupp ps join
     ( select l_suppkey , l_extendedprice , l_discount , l_quantity , l_partkey ,
        l_orderkey , n_name
       from ( select s_suppkey , n_name
              from nation n join supplier s on n.n_nationkey = s.s_nationkey
            ) s1 join lineitem l on s1.s_suppkey = l.l_suppkey
      ) l1 on ps.ps_suppkey = l1.l_suppkey and ps.ps_partkey = l1.l_partkey
    ) l2 on p.p_name like '%green%'
        and p.p_partkey = l2.l_partkey
  ) l3 on o.o_orderkey = l3.l_orderkey
)profit
group by nation , o_year
order by nation , o_year desc;
```

## BigSQL Statement

```
select nation , o_year , sum(amount) as sum_profit
from ( select n_name as nation ,
        substr(o_orderdate ,1 ,4) as o_year ,
        l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
       from part , supplier , lineitem ,
        partsupp , orders , nation
       where s_suppkey = l_suppkey
        and ps_suppkey = l_suppkey
        and ps_partkey = l_partkey
        and p_partkey = l_partkey
        and o_orderkey = l_orderkey
        and s_nationkey = n_nationkey
        and p_name like '%green%' ) as profit
group by nation , o_year
order by nation , o_year desc;
```

## Returned Item Reporting Query (Q10)

The query identifies customers who might be having problems with the parts that are shipped to them.
The returned item reporting query finds the top 20 customers, in terms of their effect on lost revenue for a given quarter, who have returned parts. The query considers only parts that were ordered in the specified quarter. The query lists the customer's name, address, nation, phone number, account balance, comment information and revenue lost. Revenue lost is defined as sum(l_extendedprice*(1-l_discount)) for all qualifying lineitems.

### SQL Statement

```
select c_custkey, c_name,  sum(l_extendedprice * (1 - l_discount)) as revenue,
 c_acctbal, n_name, c_address, c_phone, c_comment
from customer, orders,
 lineitem, nation
where c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and o_orderdate >= date('1993-10-01')
 and o_orderdate < date('1994-01-01')
 and l_returnflag = 'R'
 and c_nationkey = n_nationkey
group by c_custkey, c_name, c_acctbal,
 c_phone, n_name, c_address, c_comment
order by revenue desc
fetch first 20 rows only;
```

### Hive Statement

```
insert overwrite table q10_returned_item
select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as revenue,
 c_acctbal, n_name, c_address, c_phone, c_comment
from customer c join orders o
 on c.c_custkey = o.o_custkey
 and o.o_orderdate >= '1993-10-01' and o.o_orderdate < '1994-01-01'
 join nation n
  on c.c_nationkey = n.n_nationkey
  join lineitem l
  on l.l_orderkey = o.o_orderkey and l.l_returnflag = 'R'
group by c_custkey, c_name, c_acctbal, c_phone, n_name, c_address, c_comment
order by revenue desc
limit 20;
```

### BigSQL Statement

```
select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as revenue,
 c_acctbal, n_name, c_address, c_phone, c_comment
from customer, orders,
 lineitem, nation
where c_custkey = o_custkey
 and l_orderkey = o_orderkey
 and o_orderdate >= '1993-10-01'
 and o_orderdate < '1994-01-01'
 and l_returnflag = 'R'
 and c_nationkey = n_nationkey
group by c_custkey, c_name, c_acctbal,
 c_phone, n_name, c_address, c_comment
order by revenue desc
fetch first 20 rows only;
```

## Important Stock Identification Query (Q11)

This query finds the most important subset of suppliers' stock in a given nation.
The important stock identification query finds, from scanning the available stock of suppliers in a given nation, all the parts that represent a significant percentage of the total value of all available parts.

### SQL Statement

```
select ps_partkey, sum(ps_supplycost * ps_availqty) as value
from partsupp, supplier, nation
where ps_suppkey = s_suppkey
 and s_nationkey = n_nationkey
 and n_name = 'GERMANY'
group by ps_partkey having
 sum(ps_supplycost * ps_availqty) > (
  select sum(ps_supplycost * ps_availqty) *  0.0000001
  from partsupp, supplier, nation
  where ps_suppkey = s_suppkey
   and s_nationkey = n_nationkey
   and n_name = 'GERMANY' )
order by value desc;
```

### Hive Statement

```
insert overwrite table q11_part_tmp
select ps_partkey, sum(ps_supplycost * ps_availqty) as part_value
from nation n join supplier s
  on s.s_nationkey = n.n_nationkey
  and n.n_name = 'GERMANY'
  join partsupp ps
  on ps.ps_suppkey = s.s_suppkey
group by ps_partkey;

insert overwrite table q11_sum_tmp
select sum(part_value) as total_value
from q11_part_tmp;

insert overwrite table q11_important_stock
select ps_partkey, part_value as value
from (
 select ps_partkey, part_value, total_value
 from q11_part_tmp join q11_sum_tmp
) a
where part_value > total_value * 0.0000001
order by value desc;
```

### BigSQL Statement

```
select ps_partkey, sum(ps_supplycost * ps_availqty) as val
from partsupp, supplier, nation
where ps_suppkey = s_suppkey
 and s_nationkey = n_nationkey
 and n_name = 'GERMANY'
group by ps_partkey having
 sum(ps_supplycost * ps_availqty) > (
  select  sum(ps_supplycost * ps_availqty) *   0.0000001
  from partsupp, supplier, nation
  where ps_suppkey = s_suppkey
   and s_nationkey = n_nationkey
   and n_name = 'GERMANY' )
order by val desc;
```

## Shipping Modes and Order Priority Query (Q12)

This query determines whether selecting less expensive modes of shipping is negatively affecting the critical-priority orders by causing more parts to be received by customers after the committed date.

The shipping modes and order priority query counts, by ship mode, for lineitems actually received by customers in a given year, the number of lineitems belonging to orders for which the l_receiptdate exceeds the l_commitdate for two different specified ship modes. Only lineitems that were actually shipped before the l_commitdate are considered. The late lineitems are partitioned into two groups, those with priority URGENT or HIGH, and those with a priority other than URGENT or HIGH.

### SQL Statement

```
select l_shipmode,
 sum(case
   when o_orderpriority ='1-URGENT'
     or o_orderpriority ='2-HIGH'
    then 1
    else 0
   end) as high_line_count,
 sum(case
   when o_orderpriority <> '1-URGENT'
    and o_orderpriority <> '2-HIGH'
    then 1
    else 0
   end) as low_line_count
from orders, lineitem
where o_orderkey = l_orderkey
 and l_shipmode in ('MAIL', 'SHIP')
 and l_commitdate < l_receiptdate
 and l_shipdate < l_commitdate
 and l_receiptdate >= date('1994-01-01')
 and l_receiptdate < date ('1995-01-01')
group by l_shipmode
order by l_shipmode;
```

### Hive Statement

```
insert overwrite table q12_shipping
select l_shipmode,
  sum(case
    when o_orderpriority ='1-URGENT'
      or o_orderpriority ='2-HIGH'
    then 1
    else 0 end ) as high_line_count,
  sum(case
    when o_orderpriority <> '1-URGENT'
     and o_orderpriority <> '2-HIGH'
    then 1
    else 0 end ) as low_line_count
from
 orders o join lineitem l
 on o.o_orderkey = l.l_orderkey
 and l.l_commitdate < l.l_receiptdate
 and l.l_shipdate < l.l_commitdate
 and l.l_receiptdate >= '1994-01-01'
 and l.l_receiptdate < '1995-01-01'
where l.l_shipmode = 'MAIL' or l.l_shipmode = 'SHIP'
group by l_shipmode
order by l_shipmode;
```

### BigSQL Statement

```
select l_shipmode,
 sum(case
   when o_orderpriority ='1-URGENT'
     or o_orderpriority ='2-HIGH'
    then 1
    else 0 end) as high_line_count,
 sum(case
   when o_orderpriority <> '1-URGENT'
    and o_orderpriority <> '2-HIGH'
    then 1
    else 0 end) as low_line_count
from orders, lineitem
where o_orderkey = l_orderkey
 and l_shipmode in ('MAIL', 'SHIP')
 and l_commitdate < l_receiptdate
 and l_shipdate < l_commitdate
 and l_receiptdate >= '1994-01-01'
 and l_receiptdate < '1995-01-01'
group by l_shipmode
order by l_shipmode;
```

## Customer Distribution Query (Q13)

This query seeks relationships between customers and the size of their orders.
This query determines the distribution of customers by the number of orders they have made, including customers who have no record of orders, past or present. It counts and reports how many customers have no orders, how many have 1, 2, 3, etc. A check is made to ensure that the orders counted do not fall into one of several special categories of orders. Special categories are identified in the order comment column by looking for a particular pattern.

### SQL Statement

```
select c_count, count(*) as custdist
from (
 select c_custkey, count(o_orderkey)
 from customer left outer join orders on
  customer.c_custkey = orders.o_custkey
  and o_comment not like '%special%requests%'
 group by c_custkey ) as c_orders (c_custkey, c_count)
group by c_count
order by custdist desc, c_count desc
```

### Hive Statement

```
insert overwrite table q13_customer_distribution
select c_count, count(1) as custdist
from (
 select c_custkey, count(o_orderkey) as c_count
 from customer c left outer join orders o
  on c.c_custkey = o.o_custkey
  and not o.o_comment like '%special%requests%'
  group by c_custkey
) c_orders
group by c_count
order by custdist desc, c_count desc;
```

### BigSQL Statement

```
select c_count, count(*) as custdist
from (
 select c_custkey, count(o_orderkey)
 from customer left outer join orders on
  c_custkey = o_custkey
  and o_comment not like '%special%requests%'
  group by c_custkey ) as c_orders (c_custkey, c_count)
group by c_count
order by custdist desc, c_count desc;
```

## Promotion Effect Query (Q14)

This query monitors the market response to a promotion such as TV advertisements or a special campaign.
The promotion effect query determines what percentage of the revenue in a given year and month was derived from promotional parts. The query considers only parts actually shipped in that month and gives the percentage.

### SQL Statement

```
select 100.00 *
  sum(case
        when p_type like 'PROMO%'
        then l_extendedprice*(1-l_discount)
        else 0
      end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from lineitem, part
```

```
where l_partkey = p_partkey
and l_shipdate >= date('1995-09-01')
and l_shipdate < date ('1995-10-01');
```

### Hive Statement

```
insert overwrite table q14_promotion_effect
select 100.00 * sum(case
               when p_type like 'PROMO%'
               then l_extendedprice*(1-l_discount)
               else 0.0 end ) / sum(l_extendedprice * (1 - l_discount)) as
                   promo_revenue
from part p join lineitem l
 on l.l_partkey = p.p_partkey
 and l.l_shipdate >= '1995-09-01' and l.l_shipdate < '1995-10-01';
```

### BigSQL Statement

```
select 100.00 *
  sum(case
   when p_type like 'PROMO%'
   then l_extendedprice*(1-l_discount)
   else 0 end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from lineitem, part
where l_partkey = p_partkey
and l_shipdate >= '1995-09-01'
and l_shipdate < '1995-10-01';
```

## Top Supplier Query (Q15)

This query determines the top supplier so it can be rewarded, given more business, or identified for special recognition.
The top supplier query finds the supplier who contributed the most to the overall revenue for parts shipped during a given quarter of a given year. In case of a tie, the query lists all suppliers whose contribution was equal to the maximum, presented in supplier number order.

### SQL Statement

```
create view revenue as
select l_suppkey as supplier_no, sum(l_extendedprice * (1 - l_discount)) as
    total_revenue
from lineitem
where l_shipdate >= date ('1996-01-01')
 and l_shipdate < date ('1996-04-01')
group by l_suppkey;

select s_suppkey, s_name, s_address, s_phone, total_revenue
from supplier, revenue
where s_suppkey = supplier_no
 and total_revenue = (
   select max(total_revenue)
   from revenue )
order by s_suppkey;
drop view revenue;
```

### Hive Statement

```
insert overwrite table revenue
select l_suppkey as supplier_no, sum(l_extendedprice * (1 - l_discount)) as
    total_revenue
from lineitem
where l_shipdate >= '1996-01-01' and l_shipdate < '1996-04-01'
group by l_suppkey;

insert overwrite table max_revenue
select max(total_revenue)
from revenue;

insert overwrite table q15_top_supplier
select s_suppkey, s_name, s_address, s_phone, total_revenue
from supplier s join revenue r
  on s.s_suppkey = r.supplier_no
  join max_revenue m
  on r.total_revenue = m.max_revenue
order by s_suppkey;
```

### BigSQL Statement

```
create view revenue as
select l_suppkey as supplier_no, sum(l_extendedprice * (1 - l_discount)) as
    total_revenue
from lineitem
where l_shipdate >= date '1996-01-01'
 and l_shipdate < date '1996-04-01'
group by l_suppkey;

select s_suppkey, s_name, s_address, s_phone, total_revenue
from supplier, revenue
where s_suppkey = supplier_no
 and total_revenue = (
  select max(total_revenue)
  from revenue )
order by s_suppkey;
drop view revenue;
```

## Parts/Supplier Relationship Query (Q16)

This query finds out how many suppliers can supply parts with given attributes. It might be used, for example, to determine whether there is a sufficient number of suppliers for heavily ordered parts.

The Parts/Supplier relationship query counts the number of suppliers who can supply parts that satisfy a particular customer's requirements. The customer is interested in parts of eight different sizes as long as they are not of a given type, not of a given brand, and not from a supplier who has had complaints registered at the Better Business Bureau.

### SQL Statement

```
select p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
from partsupp, part
```

```
where p_partkey = ps_partkey
 and p_brand <> 'Brand#45'
 and p_type not like 'MEDIUM␣POLISHED%'
 and p_size in (49, 14, 23,45,19,3,36,9)
 and ps_suppkey not in (
  select s_suppkey
  from supplier
  where s_comment like '%Customer%Complaints%' )
group by p_brand,p_type,p_size
order by supplier_cnt desc,p_brand,p_type,p_size;
```

## Hive Statement

```
insert overwrite table supplier_tmp
select s_suppkey
from supplier
where not s_comment like '%Customer%Complaints%';

insert overwrite table q16_tmp
select p_brand, p_type, p_size, ps_suppkey
from partsupp ps join part p
  on p.p_partkey = ps.ps_partkey
  and p.p_brand <> 'Brand#45'
  and not p.p_type like 'MEDIUM␣POLISHED%'
  join supplier_tmp s
  on ps.ps_suppkey = s.s_suppkey;

insert overwrite table q16_parts_supplier_relationship
select p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
from (
 select *
 from q16_tmp
 where p_size = 49 or p_size = 14 or p_size = 23 or
         p_size = 45 or p_size = 19 or p_size = 3 or
         p_size = 36 or p_size = 9
) q16_all
group by p_brand, p_type, p_size
order by supplier_cnt desc, p_brand, p_type, p_size;
```

## BigSQL Statement

```
select p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
from partsupp, part
where p_partkey = ps_partkey
 and p_brand <> 'Brand#45'
 and p_type not like 'MEDIUM␣POLISHED%'
 and p_size in (49, 14, 23,45,19,3,36,9)
 and ps_suppkey not in (
  select s_suppkey
  from supplier
  where s_comment like '%Customer%Complaints%' )
group by p_brand,p_type,p_size
order by supplier_cnt desc,p_brand,p_type,p_size;
```

## Small-Quantity-Order Revenue Query (Q17)

This query determines how much average yearly revenue would be lost if orders were no longer filled for small quantities of certain parts. This may reduce overhead expenses by concentrating sales on larger shipments.
The small-quantity-order revenue query considers parts of a given brand and with a given container type and determines the average lineitem quantity of such parts ordered for all orders (past and pending) in the 7-year database. What would be the average yearly gross (undiscounted) loss in revenue if orders for these parts with a quantity of less than 20% of this average were no longer taken?

### SQL Statement

```
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where p_partkey = l_partkey
 and p_brand = 'Brand#23'
 and p_container = 'MED␣BOX'
 and l_quantity < (
  select 0.2 * avg(l_quantity)
  from lineitem
  where l_partkey = p_partkey );
```

### Hive Statement

```
insert overwrite table lineitem_tmp
select l_partkey as t_partkey, 0.2 * avg(l_quantity) as t_avg_quantity
from lineitem
group by l_partkey;

insert overwrite table q17_small_quantity_order_revenue
select sum(l_extendedprice) / 7.0 as avg_yearly
from (
 select l_quantity, l_extendedprice, t_avg_quantity
 from lineitem_tmp t join
 ( select l_quantity, l_partkey, l_extendedprice
   from part p join lineitem l
   on p.p_partkey = l.l_partkey
    and p.p_brand = 'Brand#23'
    and p.p_container = 'MED␣BOX'
 ) l1 on l1.l_partkey = t.t_partkey
) a
where l_quantity < t_avg_quantity;
```

### BigSQL Statement

```
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where p_partkey = l_partkey
 and p_brand = 'Brand#23'
 and p_container = 'MED␣BOX'
 and l_quantity < (
  select 0.2 * avg(l_quantity)
  from lineitem
  where l_partkey = p_partkey );
```

## Large Volume Customer Query (Q18)

The query ranks customers based on their having placed a large quantity order. Large quantity orders are defined as those orders whose total quantity is above a certain level.
The large volume customer query finds a list of the top 100 customers who have ever placed large quantity orders.

### SQL Statement

```
select c_name,c _custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
from customer, orders, lineitem
where o_orderkey in (
  select l_orderkey
  from lineitem group by l_orderkey having sum(l_quantity) > 300 )
 and c_custkey = o_custkey
 and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by o_totalprice desc, o_orderdate
fetch first 100 rows only
```

### Hive Statement

```
insert overwrite table q18_tmp
select l_orderkey, sum(l_quantity) as t_sum_quantity
from lineitem
group by l_orderkey;

insert overwrite table q18_large_volume_customer
select c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice,sum(l_quantity)
from customer c join orders o
 on c.c_custkey = o.o_custkey join q18_tmp t
 on o.o_orderkey = t.l_orderkey
  and t.t_sum_quantity > 300 join lineitem l
 on o.o_orderkey = l.l_orderkey
group by c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice
order by o_totalprice desc,o_orderdate
limit 100;
```

### BigSQL Statement

```
select c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice,sum(l_quantity)
from customer, orders, lineitem
where o_orderkey in (
  select l_orderkey
  from lineitem group by l_orderkey having sum(l_quantity) > 300 )
 and c_custkey = o_custkey
 and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by o_totalprice desc, o_orderdate
fetch first 100 rows only;
```

## Discounted Revenue Query (Q19)

The discounted revenue query reports the gross discounted revenue attributed to the sale of selected parts handled in a particular manner. This query is an example of code such as might be produced programmatic by a data mining tool.
Q19 finds the gross discounted revenue for all orders for three different types of parts that were shipped by air or delivered in person . Parts are selected based on the combination of specific brands, a list of containers, and a range of sizes.

### SQL Statement

```
select sum(l_extendedprice * (1 - l_discount) ) as revenue
from lineitem, part
where ( p_partkey = l_partkey
  and p_brand = 'Brand#12'
  and p_container in ( 'SM␣CASE', 'SM␣BOX', 'SM␣PACK', 'SM␣PKG')
  and l_quantity >= 1 and l_quantity <= 11
  and p_size between 1 and 5
  and l_shipmode in ('AIR', 'AIR␣REG')
  and l_shipinstruct = 'DELIVER␣IN␣PERSON' )
 or ( p_partkey = l_partkey
  and p_brand = 'Brand#23'
  and p_container in ('MED␣BAG', 'MED␣BOX', 'MED␣PKG', 'MED␣PACK')
  and l_quantity >= 10 and l_quantity <= 20
  and p_size between 1 and 10
  and l_shipmode in ('AIR', 'AIR␣REG')
  and l_shipinstruct = 'DELIVER␣IN␣PERSON' )
 or ( p_partkey = l_partkey
  and p_brand = 'Brand#34'
  and p_container in ( 'LG␣CASE', 'LG␣BOX', 'LG␣PACK', 'LG␣PKG')
  and l_quantity >= 20 and l_quantity <= 30
  and p_size between 1 and 15
  and l_shipmode in ('AIR', 'AIR␣REG')
  and l_shipinstruct = 'DELIVER␣IN␣PERSON')
```

### Hive Statement

```
insert overwrite table q19_discounted_revenue
select sum(l_extendedprice * (1 - l_discount) ) as revenue
from lineitem l join part p
  on p.p_partkey = l.l_partkey
where (
 p_brand = 'Brand#12'
 and p_container REGEXP 'SM␣CASE||SM␣BOX||SM␣PACK||SM␣PKG'
 and l_quantity >= 1 and l_quantity <= 11
 and p_size >= 1 and p_size <= 5
 and l_shipmode REGEXP 'AIR||AIR␣REG'
 and l_shipinstruct = 'DELIVER␣IN␣PERSON' )
or (
 p_brand = 'Brand#23'
 and p_container REGEXP 'MED␣BAG||MED␣BOX||MED␣PKG||MED␣PACK'
 and l_quantity >= 10 and l_quantity <= 20
 and p_size >= 1 and p_size <= 10
 and l_shipmode REGEXP 'AIR||AIR␣REG'
 and l_shipinstruct = 'DELIVER␣IN␣PERSON' )
```

```
or (
 p_brand = 'Brand#34'
 and p_container REGEXP 'LG␣CASE||LG␣BOX||LG␣PACK||LG␣PKG'
 and l_quantity >= 20 and l_quantity <= 30
 and p_size >= 1 and p_size <= 15
 and l_shipmode REGEXP 'AIR||AIR␣REG'
 and l_shipinstruct = 'DELIVER␣IN␣PERSON' )
```

### BigSQL Statement

```
select sum(l_extendedprice * (1 - l_discount) ) as revenue
from lineitem, part
where ( p_partkey = l_partkey
 and p_brand = 'Brand#12'
 and p_container in ( 'SM␣CASE', 'SM␣BOX', 'SM␣PACK', 'SM␣PKG')
 and l_quantity >= 1 and l_quantity <= 11
 and p_size between 1 and 5
 and l_shipmode in ('AIR', 'AIR␣REG')
 and l_shipinstruct = 'DELIVER␣IN␣PERSON' )
or ( p_partkey = l_partkey
 and p_brand = 'Brand#23'
 and p_container in ('MED␣BAG', 'MED␣BOX', 'MED␣PKG', 'MED␣PACK')
 and l_quantity >= 10 and l_quantity <= 20
 and p_size between 1 and 10
 and l_shipmode in ('AIR', 'AIR␣REG')
 and l_shipinstruct = 'DELIVER␣IN␣PERSON' )
or ( p_partkey = l_partkey
 and p_brand = 'Brand#34'
 and p_container in ( 'LG␣CASE', 'LG␣BOX', 'LG␣PACK', 'LG␣PKG')
 and l_quantity >= 20 and l_quantity <= 30
 and p_size between 1 and 15
 and l_shipmode in ('AIR', 'AIR␣REG')
 and l_shipinstruct = 'DELIVER␣IN␣PERSON');
```

## Potential Part Promotion Query (Q20)

The potential part promotion query identifies suppliers who have an excess of a given part available; an excess is defined to be more than 50% of the parts like the given part that the supplier shipped in a given year for a given nation. Only parts whose names share a certain naming convention are considered.

### SQL Statement

```
select s_name, s_address
from supplier, nation
where s_suppkey in (
  select ps_suppkey
  from partsupp
  where ps_partkey in (
    select p_partkey
    from part
    where p_name like 'forest%' )
   and ps_availqty > (
    select 0.5 * sum(l_quantity)
    from lineitem
    where l_partkey = ps_partkey
     and l_suppkey = ps_suppkey
     and l_shipdate >= date ('1994-01-01')
     and l_shipdate < date('1995-01-01') ) )
 and s_nationkey = n_nationkey
 and n_name = 'CANADA'
order by s_name
```

## Hive Statement

```
insert overwrite table q20_tmp1
select distinct p_partkey
from part
where p_name like 'forest%';

insert overwrite table q20_tmp2
select l_partkey, l_suppkey, 0.5 * sum(l_quantity)
from lineitem
where l_shipdate >= '1994-01-01'
  and l_shipdate < '1995-01-01'
group by l_partkey, l_suppkey;

insert overwrite table q20_tmp3
select ps_suppkey, ps_availqty, sum_quantity
from partsupp ps join q20_tmp1 t1
  on ps.ps_partkey = t1.p_partkey
  join q20_tmp2 t2
  on ps.ps_partkey = t2.l_partkey and ps.ps_suppkey = t2.l_suppkey;

insert overwrite table q20_tmp4
select ps_suppkey
from  q20_tmp3
where ps_availqty > sum_quantity
group by ps_suppkey;

insert overwrite table q20_potential_part_promotion
select s_name, s_address
from supplier s join nation n
  on s.s_nationkey = n.n_nationkey
    and n.n_name = 'CANADA'
  join q20_tmp4 t4
  on s.s_suppkey = t4.ps_suppkey
order by s_name;
```

## BigSQL Statement

```
select s_name,s_address
from supplier, nation
where s_suppkey in (
 select ps_suppkey
 from partsupp
 where ps_partkey in (
  select p_partkey
  from part
  where p_name like 'forest%' )
   and ps_availqty > (
    select 0.5 * sum(l_quantity)
    from lineitem
    where l_partkey = ps_partkey
     and l_suppkey = ps_suppkey
     and l_shipdate >= '1994-01-01'
     and l_shipdate < '1995-01-01') ) )
 and s_nationkey = n_nationkey
 and n_name = 'CANADA'
order by s_name;
```

## Suppliers Who Kept Orders Waiting Query (Q21)

This query identifies certain suppliers who were not able to ship required parts in a timely manner.

The suppliers who kept orders waiting query identifies suppliers, for a given nation, whose product was part of a multisupplier order (with current status of 'F') where they were the only supplier who failed to meet the committed delivery date.

### SQL Statement

```
select s_name, count(*) as numwait
from supplier, lineitem l1, orders, nation
where s_suppkey = l1.l_suppkey
 and o_orderkey = l1.l_orderkey
 and o_orderstatus = 'F'
 and l1.l_receiptdate > l1.l_commitdate
 and exists (
    select *
    from lineitem l2
    where l2.l_orderkey = l1.l_orderkey
     and l2.l_suppkey <> l1.l_suppkey )
 and not exists (
    select *
    from lineitem l3
    where l3.l_orderkey = l1.l_orderkey
     and l3.l_suppkey <> l1.l_suppkey
     and l3.l_receiptdate > l3.l_commitdate )
 and s_nationkey = n_nationkey
 and n_name = 'SAUDI␣ARABIA'
group by s_name
order by numwait desc, s_name
fetch first 100 rows only
```

### Hive Statement

```
insert overwrite table q21_tmp1
select l_orderkey, count(distinct l_suppkey), max(l_suppkey) as max_suppkey
from lineitem
group by l_orderkey;

insert overwrite table q21_tmp2
select l_orderkey, count(distinct l_suppkey), max(l_suppkey) as max_suppkey
from lineitem
where l_receiptdate > l_commitdate
group by l_orderkey;
```

```
insert overwrite table q21_suppliers_who_kept_orders_waiting
select s_name, count(1) as numwait
from (
 select s_name
 from
  ( select s_name, t2.l_orderkey, l_suppkey, count_suppkey, max_suppkey
    from q21_tmp2 t2 right outer join
    ( select s_name, l_orderkey, l_suppkey
      from
      ( select s_name, t1.l_orderkey, l_suppkey, count_suppkey, max_suppkey
        from q21_tmp1 t1 join
        ( select s_name, l_orderkey, l_suppkey
          from orders o join
          ( select s_name, l_orderkey, l_suppkey
            from nation n join supplier s
            on s.s_nationkey = n.n_nationkey
             and n.n_name = 'SAUDI␣ARABIA'
            join lineitem l
            on s.s_suppkey = l.l_suppkey
            where l.l_receiptdate > l.l_commitdate )
          l1 on o.o_orderkey = l1.l_orderkey and o.o_orderstatus = 'F'
        ) l2 on l2.l_orderkey = t1.l_orderkey
      ) a
      where (  count_suppkey > 1)
         or ( (count_suppkey = 1) and ( l_suppkey <> max_suppkey ) )
    ) l3 on l3.l_orderkey = t2.l_orderkey
  ) b
  where (  count_suppkey is null)
     or ( (count_suppkey = 1) and (l_suppkey = max_suppkey ) )
)c
group by s_name
order by numwait desc, s_name
limit 100;
```

## BigSQL Statement

```
select s_name, count(*) as numwait
from supplier, lineitem l1, orders, nation
where s_suppkey = l1.l_suppkey
 and o_orderkey = l1.l_orderkey
 and o_orderstatus = 'F'
 and l1.l_receiptdate > l1.l_commitdate
 and exists (
   select *
   from lineitem l2
   where l2.l_orderkey = l1.l_orderkey
    and l2.l_suppkey <> l1.l_suppkey )
 and not exists (
    select *
    from lineitem l3
    where l3.l_orderkey = l1.l_orderkey
     and l3.l_suppkey <> l1.l_suppkey
     and l3.l_receiptdate > l3.l_commitdate )
 and s_nationkey = n_nationkey
 and n_name = 'SAUDI␣ARABIA'
group by s_name
order by numwait desc, s_name
fetch first 100 rows only;
```

## Global Sales Opportunity Query (Q22)

The global sales opportunity query identifies geographies where there are customers who may be likely to make a purchase.
This query counts how many customers within a specific range of country codes have not placed orders for 7 years but who have a greater than average "positive" account balance. It also reflects the magnitude of that balance. Country code is defined as the first two characters of c_phone.

### SQL Statement

```
select cntrycode, count(*) as numcust, sum(c_acctbal) as totacctbal
from (
    select substr(c_phone,1,2) as cntrycode, c_acctbal
    from customer
    where substr(c_phone,1,2) in ('13','31','12','29','30','18','17')
     and c_acctbal > (
       select avg(c_acctbal)
       from customer
       where c_acctbal > 0.00
        and substr(c_phone,1,2) in ('13','31','12','29','30','18','17') )
     and not exists (
      select *
      from orders
      where o_custkey = c_custkey ) ) as custsale
group by cntrycode
order by cntrycode
```

### Hive Statement

```
insert overwrite table q22_customer_tmp
select c_acctbal, c_custkey, substr(c_phone, 1, 2) as cntrycode
from customer
where
  substr(c_phone, 1, 2) = '13' or
  substr(c_phone, 1, 2) = '31' or
  substr(c_phone, 1, 2) = '23' or
  substr(c_phone, 1, 2) = '29' or
  substr(c_phone, 1, 2) = '30' or
  substr(c_phone, 1, 2) = '18' or
  substr(c_phone, 1, 2) = '17';

insert overwrite table q22_customer_tmp1
select avg(c_acctbal)
from q22_customer_tmp
where c_acctbal > 0.00;

insert overwrite table q22_orders_tmp
select o_custkey
from orders
group by o_custkey;

insert overwrite table q22_global_sales_opportunity
select cntrycode, count(1) as numcust, sum(c_acctbal) as totacctbal
from (
 select cntrycode, c_acctbal, avg_acctbal
 from q22_customer_tmp1 ct1 join
  ( select cntrycode , c_acctbal
    from q22_orders_tmp ot
    right outer join q22_customer_tmp ct
    on ct.c_custkey = ot.o_custkey
    where o_custkey is null
  ) ct2
) a
where c_acctbal > avg_acctbal
group by cntrycode
order by cntrycode;
```

## BigSQL Statement

```
select cntrycode , count(*) as numcust , sum(c_acctbal) as totacctbal
from (
    select substr(c_phone ,1 ,2) as cntrycode , c_acctbal
    from customer
    where substr(c_phone ,1 ,2) in ('13','31','12','29','30','18','17')
     and c_acctbal > (
      select avg(c_acctbal)
      from customer
      where c_acctbal > 0.00
       and substr(c_phone ,1 ,2) in ('13','31','12','29','30','18','17') )
     and not exists (
      select *
      from orders
      where o_custkey = c_custkey ) ) as custsale
group by cntrycode
order by cntrycode;
```

# D. Project Documentation

## D.1. Project Management

### D.1.1. Milestones

At the start of this thesis we defined the following time schedule for this project:

| | | |
|---|---|---|
| w01 | 16.09. – 22.09. | research |
| w02 | 23.09. – 29.09. | use case preparation |
| w03 | 30.09. – 06.10. | cluster setup |
| w04 | 07.10. – 13.10. | testing Hive and IBM BigSQL |
| w05 | 14.10. – 20.10. | setting up IBM DB2 DPF |
| w06 | 21.10. – 27.10. | IBM DB2 performance tests |
| w07 | 28.10. – 03.11. | IBM DB2 performance tests |
| w08 | 04.11. – 10.11. | Hive performance tests |
| w09 | 11.11. – 17.11. | Hive performance tests |
| w10 | 18.11. – 24.11. | IBM BigSQL performance tests |
| w11 | 25.11. – 01.12. | IBM BigSQL performance tests |
| w12 | 02.12. – 08.12. | analyzing and interpreting measurement results |
| w13 | 08.12. – 15.12. | reserve |
| w14 | 16.12. – 22.12 | reserve |

Table D.1.: project planning

We had a close schedule during our bachelor thesis which last only 14 weeks. The problem of scheduling tasks and measurements was that we had only one physical cluster that we could work on, but there were three different products that we had to test and configure to a production level for the experiments within 14 weeks.

### D.1.2. Project Monitoring

For this bachelor thesis we earn 12 ETCS points, each credit stands for 30 hours of work, which result a total effort of 360 hours per student. We planed to work 2,5 days per week at the school for our thesis. At the end of this work we both have invested a small amount more time as demanded:

| | |
|---|---|
| Christof Büchi | 396h |
| Susanne Mathys | 384h |

Table D.2.: amount of work

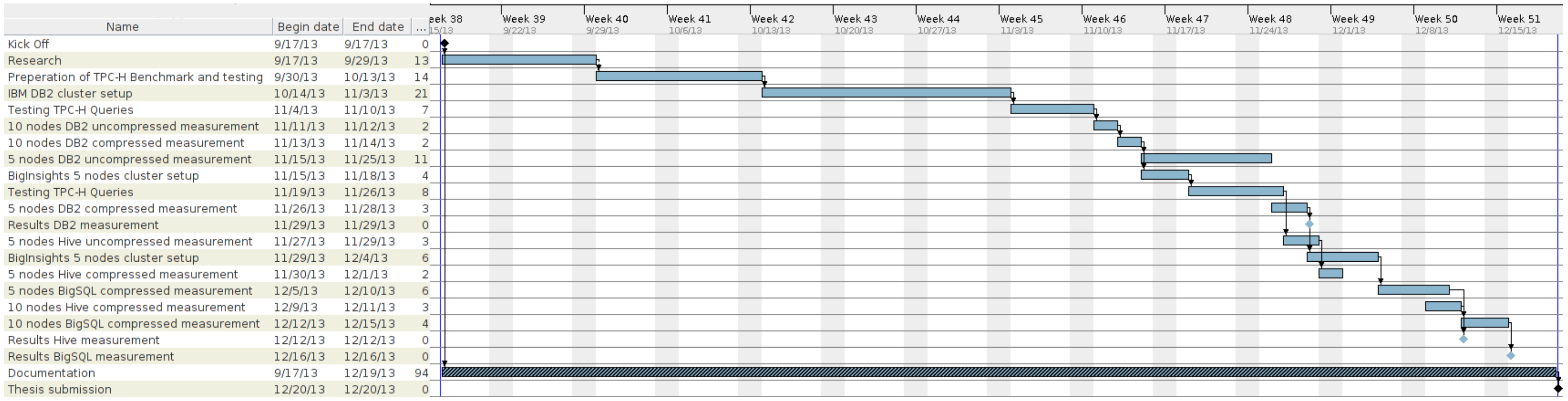
### D.1.3. Course of the Project

We started with the IBM DB2 cluster, as this was the more unknown system for us. At the beginning of our work we were aware, that we had to focus on IBM DB2 as long as the measurements on the cluster lasts. As we both were unexperienced in setting up a IBM DB2 DPF cluster, we made a few mistakes. First we selected a scale-factor of 3000 for the TPC-H benchmark files. The data that was stored on the cluster was simply to large for our disks. As we had to reorganize the DB2 tables, we run out of storage-space because normally the reorg-process is not working in-place. So we choose to decrement the scale-factor to 1000.
We used a lot of time for enabling compression and reorg the database. We spent also a lot of time for setting indexes on the tables, although we could take advantage of the IBM DB2 Advis tool.
We started IBM DB2 performance measurement at 11. november and all 5 nodes were used until 15. november
Then we decided to work parallel on two clusters with 5 nodes. We then start the preparation of Hive and run the DB2 measurements on the same time.

Hive was a new technology for us and we had to invest a lot of time for tuning Hive for our queries as mentioned in Chapter Experiments. With a few executions for testing purpose on the Hadoop cluster, we realized that compression is a key factor for the query execution time. Testing different compression algorithms, configure and install them on our cluster was a time consuming task. Finally we built the internal Hive tables with RCFile and Gzip compression. Hive measurements took place from 27. november until 12. november.

At the time the IBM DB2 measurements were finished we did some investigation about Presto. During our thesis we wanted to react to the actual announcement of Presto, as we were free to lead our study in different directions. Unfortunately with our version of IBM BigInsights it was not possible to install the Presto components on our cluster.

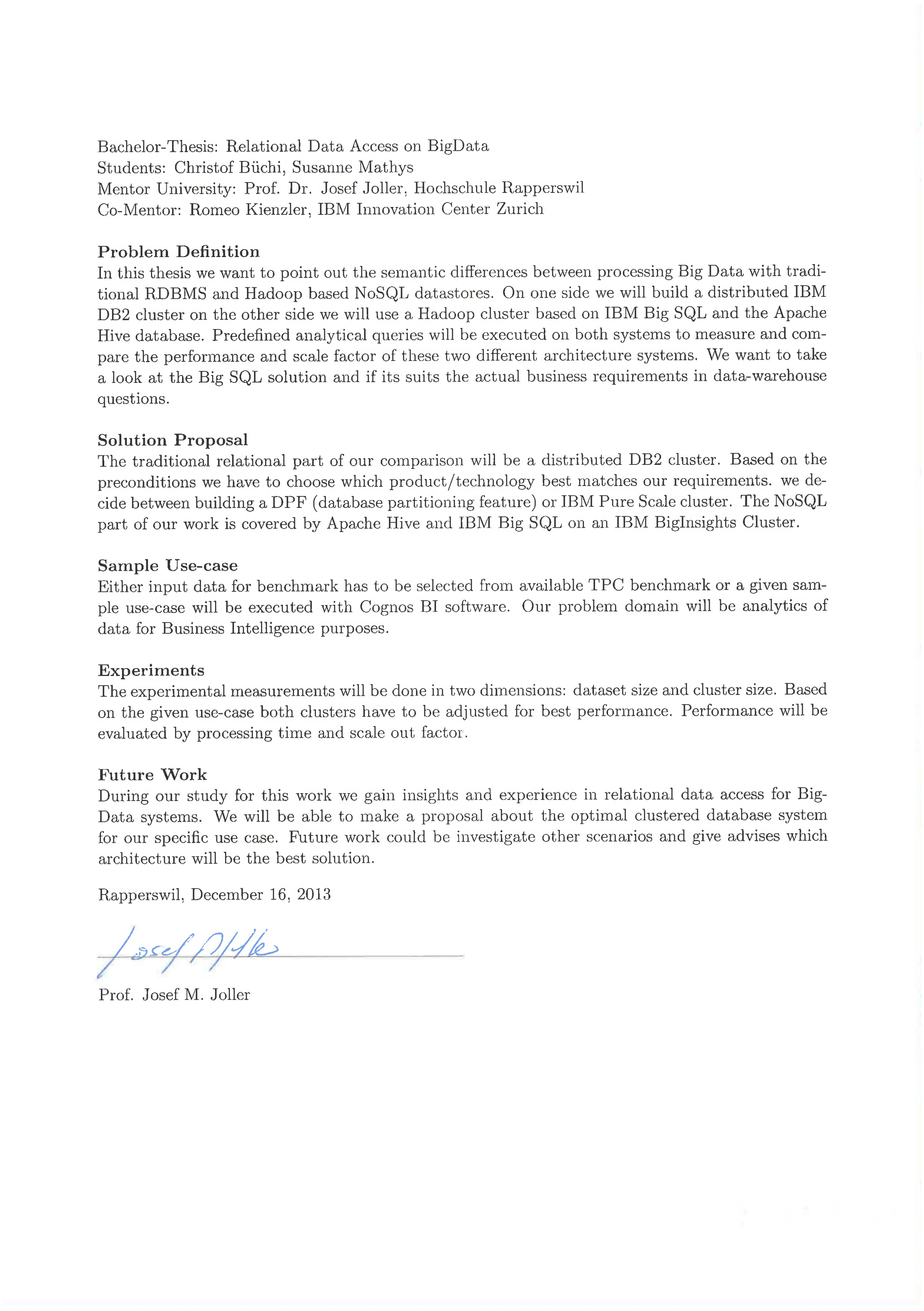
We also analyzed the tuning possibilities of IBM BigSQL in parallel to the 5 nodes Hive executions. At this time it was a benefit for us, that with IBM BigSQL we could use the database and its setting from Hive and query statements in SQL were present from the DB2 executions. Last task was to run the 10 node experiments and prepare the results for the documentation.



| Name | Begin date | End date | ... |
|---|---|---|---|
| Kick Off | 9/17/13 | 9/17/13 | 0 |
| Research | 9/17/13 | 9/29/13 | 13 |
| Preperation of TPC-H Benchmark and testing | 9/30/13 | 10/13/13 | 14 |
| IBM DB2 cluster setup | 10/14/13 | 11/3/13 | 21 |
| Testing TPC-H Queries | 11/4/13 | 11/10/13 | 7 |
| 10 nodes DB2 uncompressed measurement | 11/11/13 | 11/12/13 | 2 |
| 10 nodes DB2 compressed measurement | 11/13/13 | 11/14/13 | 2 |
| 5 nodes DB2 uncompressed measurement | 11/15/13 | 11/25/13 | 11 |
| BigInsights 5 nodes cluster setup | 11/15/13 | 11/18/13 | 4 |
| Testing TPC-H Queries | 11/19/13 | 11/26/13 | 8 |
| 5 nodes DB2 compressed measurement | 11/26/13 | 11/28/13 | 3 |
| Results DB2 measurement | 11/29/13 | 11/29/13 | 0 |
| 5 nodes Hive uncompressed measurement | 11/27/13 | 11/29/13 | 3 |
| BigInsights 5 nodes cluster setup | 11/29/13 | 12/4/13 | 6 |
| 5 nodes Hive compressed measurement | 11/30/13 | 12/1/13 | 2 |
| 5 nodes BigSQL compressed measurement | 12/5/13 | 12/10/13 | 6 |
| 10 nodes Hive compressed measurement | 12/9/13 | 12/11/13 | 3 |
| 10 nodes BigSQL compressed measurement | 12/12/13 | 12/15/13 | 4 |
| Results Hive measurement | 12/12/13 | 12/12/13 | 0 |
| Results BigSQL measurement | 12/16/13 | 12/16/13 | 0 |
| Documentation | 9/17/13 | 12/19/13 | 94 |
| Thesis submission | 12/20/13 | 12/20/13 | 0 |

Figure D.1.: Project Gant Chart

## D.2. Scope of Work



Bachelor-Thesis: Relational Data Access on BigData
Students: Christof Büchi, Susanne Mathys
Mentor University: Prof. Dr. Josef Joller, Hochschule Rapperswil
Co-Mentor: Romeo Kienzler, IBM Innovation Center Zurich

**Problem Definition**
In this thesis we want to point out the semantic differences between processing Big Data with traditional RDBMS and Hadoop based NoSQL datastores. On one side we will build a distributed IBM DB2 cluster on the other side we will use a Hadoop cluster based on IBM Big SQL and the Apache Hive database. Predefined analytical queries will be executed on both systems to measure and compare the performance and scale factor of these two different architecture systems. We want to take a look at the Big SQL solution and if its suits the actual business requirements in data-warehouse questions.

**Solution Proposal**
The traditional relational part of our comparison will be a distributed DB2 cluster. Based on the preconditions we have to choose which product/technology best matches our requirements. we decide between building a DPF (database partitioning feature) or IBM Pure Scale cluster. The NoSQL part of our work is covered by Apache Hive and IBM Big SQL on an IBM BigInsights Cluster.

**Sample Use-case**
Either input data for benchmark has to be selected from available TPC benchmark or a given sample use-case will be executed with Cognos BI software. Our problem domain will be analytics of data for Business Intelligence purposes.

**Experiments**
The experimental measurements will be done in two dimensions: dataset size and cluster size. Based on the given use-case both clusters have to be adjusted for best performance. Performance will be evaluated by processing time and scale out factor.

**Future Work**
During our study for this work we gain insights and experience in relational data access for Big-Data systems. We will be able to make a proposal about the optimal clustered database system for our specific use case. Future work could be investigate other scenarios and give advises which architecture will be the best solution.

Rapperswil, December 16, 2013

[illegible]

Prof. Josef M. Joller

# E. Personal Reports

## Christof Büchi

From the first time, I heard about this topic i was really fascinated and motivated. We had the possibility to write about a very up-to-date topic. Big Data is nowadays more and more an important subject in information science. The topic was very less framed, which means we had a lot of flexibility to create an intersting thesis, but on the other hand we had a lot of factors to involve. For me, it was very important to create a business related use case because big data is mostly used for big business decisions. The TPC-H benchmark is focused on this, and has a real world data structure. As an additional benefit, it includes some generated but realistic data - with possibility to scale up.
First, we chose this scale-factor too big, which was costs us too much time. As second, we gave us the requirement to build a production-ready IBM DB2 cluster. Both of us had less experience in distributed database cluster. We already know the Hadoop cluster and its difficulties, but IBM DB2 DPF was completely new for both of us.
While Susanne has a some knowledge in database systems, I was mostly focusing on the cluster and its setup. With that in mind, we spread the work into multiple work units. During the work, we also had a lot of new things to learn and to read from other papers. We invested lot of time together to discuss those new aspects and new possibilities to get the best results. I am remembering in discussions for RCFile and other file structures inside the hive-tables, which was really interesting. It was a big benefit to discuss such questions, because I could learn many things from those discussions, and I am sure I could also integrate my thoughts and ideas. I am very thankful to Susanne for all those discussions, which maybe costs a few nerves.
As already written, we had a open scope for this thesis, which I am also very thankful to our adviser, Prof. Josef M. Dr. Joller and also to our external co-examiner Romeo Kienzler. That enabled us to focus on the essential parts of this work.
In the beginning of the work we were really optimistic to get scaling up characteristics of the used systems. During the work we had to reduce our aim, which was very pity, but necessary to finish it in 14 weeks. It was a very short time with much intent, which results also in a few hours in the evening in the serverroom while resetting the cluster and preparation of the new environment. This thesis also helps me at organizing my time in such open scoped projects.

## Susanne Mathys

Thanks to Prof. Dr. Joller which gave us a free hand, we could focus on actual subjects and also include or excludes topics during our study. The support of Romeo Kienzler was also a benefit for this work. Especially at the beginning he gave us lots of ideas and hints for our measurements. Later he was valuable because we did proofreading of our texts.
Teamwork with Christof was pleasing and we could distribute lot of the work in different tasks and

assign them to the one of us which has more knowledge in that filed. On the other side we often worked together to discuss and solve the problems which approved during the study.

During this work I did not only improve my knowledge and experience with the Hadoop platform as well as raised up my DB2 skills. Before I had only worked with DB2 as a software engineer and did not care about administrating a DB2 database system. It was interesting to analyze the workload of the TPC-H queries and suggesting which queries will be long or short running on our clusters. Further interpreting the results of measurements and find out why a certain query will be performed faster on the Big Data platform was also a compelling task.

The fact that the DB2 cluster with 10 nodes worked straight forward for this workload was impressive to me. It is remarkable that a system that was developed decades ago still can be optimized and adjusted to fit the requirements of today for processing Big Data.
The Hadoop ecosystem provides several possibilities for business intelligence approach over Big Data. In this work we saw that in some circumstances processing an analytical query on a Big Data platform could be faster and maybe taking this route could bring future advantages.
At the moment the market of Big Data analytics is fast changing and growing and often new version of improved products are released. This results in improvement but also in a need of a good overview on the market and one has to be stay informed about the changes which are going through the Big Data topic, which can be really challenging.

# List of Figures

# List of Tables

# Bibliography

[1] SAP AG. SAP HANA. http://www.saphana.com/. [Online; accessed 5-Dec-2013].

[2] Cloudera. Cloudera Impala. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html. [Online; accessed 8-Dec-2013].

[3] IBM Cooperation. http://www-01.ibm.com/software/data/infosphere/biginsights/features.html. [Online; accessed 8-December-2013].

[4] IBM Cooperation. Biginsights JAQL. http://publib.boulder.ibm.com/infocenter/bigins/v1r1/index.jsp/. [Online; accessed 8-Dec-2013].

[5] IBM Cooperation. Ibm db2 10.5 information center for linux, unix, and windows - mdc. http://pic.dhe.ibm.com/infocenter/db2luw/v10r5/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.partition.doc%2Fdoc%2Fc0007201.html. [Online; accessed 5-December-2013].

[6] IBM Cooperation. Ibm db2 purescale. http://www-01.ibm.com/software/data/db2/linux-unix-windows/purescale/. [Online; accessed 5-December-2013].

[7] IBM Cooperation. Ibm gpfs. http://www-03.ibm.com/systems/software/gpfs/. [Online; accessed 5-December-2013].

[8] IBM Cooperation. Ibm nosql-support. http://www-01.ibm.com/software/data/db2/linux-unix-windows/nosql-support.html. [Online; accessed 5-December-2013].

[9] Transaction Processing Performance Council. Tpc benchmarks. http://www.tpc.org/. [Online; accessed 1-December-2013].

[10] Transaction Processing Performance Council. Tpc-h - ten most recently published results. http://www.tpc.org/tpch/results/tpch_last_ten_results.asp. [Online; accessed 1-December-2013].

[11] Transaction Processing Performance Council. Tpc-h benchmark documentation. http://www.tpc.org/tpch/spec/tpch2.16.0.pdf. [Online; accessed 1-December-2013].

[12] Transaction Processing Performance Council. Tpc-h benchmark specification. http://www.tpc.org/tpch/spec/tpch_2_16_0.zip. [Online; accessed 1-December-2013].

[13] A. Crume, J. Buck, C. Maltzahn, and S. Brandt. Compressing intermediate keys between

mappers and reducers in scihadoop. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 7–12, 2012.

[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. 51(1):107–113, 2008.

[15] The Apache Software Foundation. Apache Flume. http://flume.apache.org/. [Online; accessed 8-Dec-2013].

[16] The Apache Software Foundation. Apache HBase. http://hbase.apache.org/. [Online; accessed 8-Dec-2013].

[17] The Apache Software Foundation. Apache HCatalog. http://hive.apache.org/hcatalog/. [Online; accessed 8-Dec-2013].

[18] The Apache Software Foundation. Apache Hive. http://hive.apache.org/. [Online; accessed 8-Dec-2013].

[19] The Apache Software Foundation. Apache Mahout. http://mahout.apache.org/. [Online; accessed 8-Dec-2013].

[20] The Apache Software Foundation. Apache Oozie. http://oozie.apache.org/. [Online; accessed 8-Dec-2013].

[21] The Apache Software Foundation. Apache Pig. http://pig.apache.org/. [Online; accessed 8-Dec-2013].

[22] The Apache Software Foundation. Apache Sqoop. http://sqoop.apache.org/. [Online; accessed 8-Dec-2013].

[23] The Apache Software Foundation. Apache Zookeeper. http://zookeeper.apache.org/. [Online; accessed 8-Dec-2013].

[24] The Apache Software Foundation. Hive TPC-H Benchmark. https://issues.apache.org/jira/browse/HIVE-600. [Online; accessed 5-Dec-2013].

[25] The Apache Software Foundation. HiveQL. https://cwiki.apache.org/confluence/display/Hive/LanguageManual. [Online; accessed 5-Dec-2013].

[26] Google. A hadoop library of snappy compression. https://code.google.com/p/hadoop-snappy/. [Online; accessed 5-December-2013].

[27] Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1199–1208, Washington, DC, USA, 2011. IEEE Computer Society.

[28] http://www.capgemini.com/. The Principles of the Business Data Lake. http://www.capgemini.com/resources/the-principles-of-the-business-data-lake. [Online; accessed 18-Dec-2013].

[29] The McKinsey Global Institute. Competing through data. http://www.mckinsey.com/insights/marketing_sales/competing_through_data_three_experts_offer_their_game_plans. [Online; accessed 5-December-2013].

[30] Xuhui Liu, Jizhong Han, Yunqin Zhong, Chengde Han, and Xubin He. Implementing webgis on hadoop: A case study of improving small file i/o performance on hdfs. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1–8, 2009.

[31] Microsoft. Sql server partitioned tables. http://technet.microsoft.com/en-us/library/ms190787.aspx. [Online; accessed 5-December-2013].

[32] MySQL.com. Mysql cluster cge. http://oracle.com/rac. [Online; accessed 5-December-2013].

[33] Raghunath Nambiar, Meikel Poess, Andrew Masland, H Reza Taheri, Matthew Emmerton, Forrest Carman, and Michael Majdalany. Tpc benchmark roadmap 2012. In *Selected Topics in Performance Evaluation and Benchmarking*, pages 1–20. Springer, 2013.

[34] Oracle. Oracle real application clusters. http://oracle.com/rac. [Online; accessed 5-December-2013].

[35] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[36] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, August 2009.

[37] Twitter. Scalding. https://twitter.com/scalding/. [Online; accessed 8-Dec-2013].

[38] Tom White. Hadoop–the definite guide, 2009.