Comparative Analysis of RDBMS and NoSQL Databases

Jam Jahanzeb Khan Behan
Free University of Brussels
Bruxelles, Belgium
Email: jbehan@ulb.ac.be

Ali Inam
Institute of Business Administration
Karachi, Pakistan
Email: ali.inam03@gmail.com

Meesum Ali
Institute of Business Administration
Karachi, Pakistan
Email: meesumdex@gmail.com

Muhammad Talha Khan
Institute of Business Administration
Karachi, Pakistan
Email: talhakhan298@gmail.com

***Abstract*—Big Data has been the subject of increased research since data has been termed the new oil for the 21st century. Recently, smart grids have been used by energy providers to store the massive amount of data that is generated at regular time intervals. K-Electric is one such company in Pakistan that provides the residents of Karachi City with electrical energy. The company stores their data in a Not only Structured Query Language (NoSQL) database, since the smart grid data has a high volume, accelerated velocity, and tremendous variety. Hence, we feel that we can provide an important comparison of NoSQL tools using this data. NoSQL tools have been actively used for storage purposes in the industry. Companies like eBay, GitHub, and Amazon have been using these tools for storage and analytical purposes alike. In this paper, we compare and analyze four different technologies: MySQL, MongoDB, MonetDB, and InfluxDB using the data generated by the smart grids of K-Electric.**

Keywords–NoSQL; Big Data; RDBMS; Performance Comparison; Smart Grid.

I. INTRODUCTION

A smart grid is an electrical grid that provides a variety of operations and energy measures. These measures can include smart meters, smart appliances, renewable energy resources, and energy-efficient resources. The most important aspects of the smart grid are electronic power conditioning, control of the production, and distribution of electricity.

The Big Data phenomenon is defined using 3 Vs, where we have too much data (volume) that is being collected at an extremely high rate (velocity) and contains mostly unstructured data (variety) [1] [2]. Traditionally, data has been managed and stored in Relational Database Management Systems (RDBMSs) with the focus to optimize the storage space. However, querying is a time consuming task in these traditional RDBMS technologies. In RDBMS, the data is distributed in different tables, and then these tables are virtually joined for performing advanced querying, hence the slow response time. However, with the sudden explosion of data, due to the Web and data accessibility, the old technologies could not handle the increasing demand for data storage and querying. Unfortunately, since this amount of data is not manageable by traditional RDBMS technologies, we witnessed the rise of NoSQL databases. These new technologies have been used to analyze Big Data to reveal new insights and optimize the decision making strategy for executives. As of present, there are more than 225 NoSQL databases [3]–[7].

A database system that is distributed does not require a fixed table schema. The schema is mostly built at runtime based on the query. As there is no schema, (i) the join operations are usually avoided, (ii) the technology can be scaled horizontally, (iii) the system does not expose a Structured Query Language (SQL) interface and (iv) the tool can be open source [8]. However, even though the NoSQL databases are the by-products of the Web 2.0 era, these tools were solely used when the Web service providers had a large number of users. These providers discovered that the RDBMS can be used either when the database is small but requires frequent read & write transactions, or when the database is large but requires batch transactions while rarely needing write transactions. They concluded that RDBMS cannot be used for large databases with heavy read & write workloads [5].

In this paper, we aim to use the data that is stored in the RDBMS and see how well it can be analyzed using a NoSQL system. The data is collected from K-Electric, a vertically integrated investor-owned utility company managing the generation, transmission, and distribution of energy to consumers. The purpose of this paper is to analyze the performance of K-Electric's relational data in a non-relational environment.

The rest of the paper is structured as follows: Section II highlights some of the related work done on comparing RDBMS technologies with NoSQL technologies. In Section III, we provide a detailed account of the technologies we have selected for our experiments. Section IV briefly outlines the structure of the data obtained from K-Electric. Section V explains the technical setup for the technologies, the experiments that we have performed, and the results of these experiments. Finally, we give our concluding remarks in Section VI.

II. RELATED WORK

In this section, we highlight other works that have aimed at comparing NoSQL databases.

Hadjigeorgiou et al. [9] have compared the performance of MongoDB and MySQL when they are scaled and sharded. The metrics they have used are (i) total queries per second and (ii) total queries per second per thread. The authors tested the systems on a dataset related to the music industry. Firstly, they make different schemas for the RDBMS and for the NoSQL systems. Metrics are recorded for three experiments that are done using (i) a single node, (ii) multiple nodes, and (iii) sharding. The authors conclude that the most important factor was the query type used since MongoDB was able to handle more complex queries faster, due mainly to its simpler schema while having to duplicate the data. MongoDB also performed better during insertions. They also state that MySQL

 ISBN: 978-1-61208-790-0

performs better when deleting data since it performs better in simple search queries. This might be the case because deletion requires finding the record to be deleted first, which is easier in MySQL since there is only one instance. Finally, the authors highlight that both databases have had a linear trend in the benchmarks.

Ansari et al. [10] have selected Hbase, MongoDB, Cassandra, and Elasticsearch NoSQL technologies and compared them using data from smart grids. The smart grid meter data that they were using was structural column-based data. For experimentation purposes, they used the default configuration of the respective NoSQL technology. They compared the databases on effectiveness (using the WRITE and READ parameters) and scalability (by measuring the execution performance of the full mechanism). The results showed that Cassandra had the smallest average latency in both read and write processes. This is possible because Cassandra is one of the best column-based databases and the data to be evaluated was column-based data.

Venkatraman et al. [11] discussed the four main data models of non-relational databases and compared them to SQL databases. They first presented the context of Big Data analytics and NoSQL databases and then compared them based on high availability, partition tolerance, high scalability, consistency, auto-sharding, write frequently & read less (priority is given to write operations as compared to read operations), fault tolerance (no single point of failure), multiversion concurrency control (MVCC), and, finally, concurrency control (locks). The authors performed benchmark tests, however, they did not provide the results. The authors only discussed and explained the results. They state that Couchbase processes more operations per second with lower average latency in reading and writing data than both MongoDB and Cassandra. Also, Cassandra is faster in writing than MongoDB, however, both have almost equal reading speed. The authors conclude that the flexible data modeling of NoSQL is well suited to support dynamic scalability and improved performance for Big Data analytics.

Santos et al. [12] have used Geographic Information Systems (GIS) data to compare PostGIS (a spatial database extender for PostgreSQL object-relational database), MongoDB, and Neo4j with Neo4j-Spatial. For comparison purposes, the authors have performed different types of operations (read, write, etc.), where each operation contains a group of queries. Even though all groups include 20 parameterized queries, the parameter values vary within predefined ranges for each group. The data comparison metrics used are (i) Nearby Points of Interest Radius and K-Nearest Neighbors (KNNs), (ii) Urban Routing, (iii) Map View, and (iv) Position Tracking. In the conclusion, the authors have highlighted that, since the spatial attributes are much more complex to handle as compared to strings, numbers, and other relational data types, evaluating and benchmarking spatial DBMS performances is not as simple as doing so in RDBMS. The authors also state that there was a need for data heterogeneity within the same RDBMS, as each type of query runs faster in a different data structure.

III. Selected Technologies

In this paper, we have selected three NoSQL technologies to compare against MySQL [13], the RDBMS technology in place at K-Electric. We have selected MongoDB [14], MonetDB [15], and InfluxDB [16] as the NoSQL datastores.

A. MongoDB

In MongoDB, the data is stored in flexible, JavaScript Object Notation (JSON) like documents, where fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in the application code, making data easy to work with. Ad hoc queries, indexing, and real-time aggregation provide powerful ways to access and analyze the data. It is a distributed database, so high availability, horizontal scaling, and geographic distribution are built-in and easy to use while providing querying and indexing functionalities. Furthermore, MongoDB is an open-source project, hence, aiding in its popularity of use. We have selected MongoDB because it contains the best mixture obtained from RDBMS and NoSQL technologies, which in turn enables users to build new applications. It provides the data model flexibility, elastic scalability, and high performance of NoSQL databases, hence aiding in a continuous enhancement of applications, while scaling on commodity hardware [17].

B. InfluxDB

We have selected InfluxDB because it is an open-source time-series database that is optimized for fast, high-availability storage, and retrieval of time series data. It has no external dependencies and provides an SQL-like language with built-in time-centric functions for querying. Each point consists of several key-value pairs called the fieldset and a timestamp. A series is defined when a set of key-value pairs are grouped together. Finally, series are grouped together by a string identifier to form a measurement. Points are indexed by their time and tagset. Retention policies are defined on measurement and control of how data is downsampled and deleted. Continuous queries run periodically, storing results in a target measurement.

C. MonetDB

MonetDB is an open-source column-oriented database management system designed to provide high performance on complex queries against large databases, such as combining tables with hundreds of columns and millions of rows. Its architecture is represented in three layers, each with its own set of optimizers. The front-end provides a query interface for SQL, where queries are parsed into domain-specific representations, like relational algebra for SQL, and optimized. The generated logical execution plans are then translated into instructions, which are passed to the next layer. The middle or back-end layer provides a number of cost-based optimizers. The bottom layer is the database kernel, which provides access to the data stored in Binary Association Tables, where each table consists of an Object-identifier and value columns, representing a single column in the database. Internal data representation also relies on the memory addressing ranges of contemporary CPUs using demand paging of memory-mapped files and, thus, departing from traditional DBMS designs involving complex management of large data stores in limited memory. We have selected MonetDB because it has been designed to provide high performance on complex queries against large databases and also because it has been applied in high-performance applications for better analytics.

 ISBN: 978-1-61208-790-0

IV. DATA

The data comprises of meter readings from over 9000 smart meters spread throughout Karachi, Pakistan. Based on the type of the meter installed, the data is generated at different intervals. These smart meters are installed at consumer sites, on Pole Mounted Transformers (PMTs), and on distribution feeders. The data is initially stored in an internal buffer, of each respective meter. The device then communicates with the server, based on configurable intervals (using the push/pull protocol). In case of any communication lapse, the infrastructure is designed to record the lost data over a period of seven days. The data is recorded in Head End, the objective of which is to acquire meter data and monitor device parameters automatically, thus avoiding any human intervention.

The data being utilized for this project is collected over a period of three months, with an uncompressed size of approximately 45 GB. The devices installed at the consumers end generate data over a 30 minutes interval, while the devices placed on distribution assets generate data after every 15 minutes. As a first step, we aimed to understand the data on hand by (i) manually looking at a smaller chunk of manageable data and by (ii) asking the domain experts. We also had regular meetings with the employees of K-Electric, who provided an extensive explanation regarding the data: what each field corresponded to, how a particular field is important for further processing, the type of values that each field contains, and which fields were of high importance.

Once the data was analyzed, we gained the understanding of the fields provided in the data. The details of the fields are stated in Table I.

TABLE I. DATA FIELD DESCRIPTIONS

Field name	Field definition
DeviceID	The unique meter identification ID
Time	The time at which the reading was recorded at
Date	The date on which the reading was recorded at in MM/DD/YYYY format
Value	The profile value we obtain against the corresponding Result-TypeID
MeasuredUnit	The unit of measurement we obtain after multiplying Value with the number (10Ŝcaler)
Scaler	Represents the number (10Ŝcaler) to be multiplied with the Value to get the Value measured according to the units in MeasuredUnit
ResultTypeID	The profile for which the value has been generated.
Status	This is a 32 bit number to represent the status of the meter itself
Description	The description of the DeviceID. Not properly maintained

We imported the original data from the databases to use the data for querying purposes and then evaluate the query execution times. Some fields have been highlighted as an essential part of the analysis. However, we omitted the fields: **Description**, **Status**, **MeasuredUnit**, and **Scalar** since these columns did not provide any information relevant to our analysis. Moreover, a new field by the name of **Timestamp** was created by concatenation of the Time and Date fields.

V. EXPERIMENTATION

In this section, we provide the technical details for each of the selected technology and how they were set up. Also, in this section, we provide the queries that have been devised for comparison purposes, the benchmark we obtained while using MySQL (since it is the main technology at K-Electric), and how the other tools performed as compared to the results of MySQL.

A. Technical Details

Since we wanted to work independently, that is without the restriction of having to carry the data, or the setup, we decided to use Amazon Web Services (AWS). To setup the environment, we created instances (not a VM environment) of each of the four technologies. We also had to make customized adjustments to some of the databases instances, and the details are as follows:

MySQL: As stated in the previous section, the database deployed at K-Electric is MySQL and, for our purpose, we created an AWS instance for MySQL and accessed that instance by means of MySQL Workbench on our personal computers. To enable working on the data in RDBMS, we intially required a schema of the data and store data in form of tables. Fortunately, K-Electric stored the data into one huge table–and we, therefore, kept our own schema in accordance to that. For our instance, we stored the data in a table named "dataset".

MongoDB: In MongoDB every dataset is a collection, and we can query each collection using their keys. For the experimentation, we created a collection called "SM_RECS". Furthermore, we created custom indexes on two attributes: DateTime and ID. We would like to mention, due to its nature of complexity, we decided to opt out of the UNION queries in MongoDB.

MonetDB: We followed the same procedure as that of MySQL and created a single table called "dataset".

InfluxDB: For InfluxDB, we stored our data in a table named "dataset" and, after much searching, we found out that InfluxDB does not, in fact, have native support for UNION. Hence, we were unable to perform the UNION queries [18].

B. Queries

We have written queries of different categories for each database (see Table III) and ran them on the AWS instances. The queries belong to one of the following categories:

1) Simple Query
2) Range Query
3) Aggregated Query
4) Nested Query
5) UNION Query

As stated previously, we were unable to perform UNION queries for MongoDB and InfluxDB.

C. Results

To provide an unbiased experimental runtime, the repeated the experiments 10 times. The average time required for experimentation to complete and the results are outlined in Table II. As it can be seen from the results, all the technologies were able to obtain the same results (in terms of the number of records) for identical queries. Hence, we compare the results based on the Runtime(s) columns in Table II that correspond to the total time taken to obtain the results while computing on the given technology instance. As stated previously, the results obtained from MySQL serve as a baseline for the NoSQL technologies, and it is safe to say that all the NoSQL technologies were able to obtain better results than the baseline.

 ISBN: 978-1-61208-790-0

It can be observed that MonetDB was able to outperform MySQL, and was still able to provide results for all categories of queries stated in Section V-B. It is also worth mentioning that InfluxDB outperformed all the systems in terms of computational time. However, since it does not provide UNION query facilities, we cannot rely on this system for being a replacement of the traditional RDBMS.

TABLE II. QUERY RUNTIME AND RESULTS

Query Number	Result	Runtime (s)
MongoDB 1	84965 row(s) returned	0.65
MongoDB 2	781 row(s) returned	57.26
MongoDB 3	1 row(s) returned	0.04
MongoDB 4	697409 row(s) returned	0.88
MongoDB 5	33856 row(s) returned	854.75
MongoDB 6	33856 row(s) returned	874.18
MongoDB 7	0 row(s) returned	1054.73
InfluxDB 1	84965 row(s) returned	1.38
InfluxDB 2	781 row(s) returned	13.55
InfluxDB 3	1 row(s) returned	10.85
InfluxDB 4	697409 row(s) returned	0.08
InfluxDB 5	33856 row(s) returned	15.37
InfluxDB 6	33856 row(s) returned	10.37
InfluxDB 7	0 row(s) returned	58.88
MonetDB 1	84965 row(s) returned	8.54
MonetDB 2	781 row(s) returned	60.46
MonetDB 3	1 row(s) returned	57.19
MonetDB 4	697409 row(s) returned	1.88
MonetDB 5	33856 row(s) returned	78.64
MonetDB 6	33856 row(s) returned	76.63
MonetDB 7	0 row(s) returned	256.15
MonetDB 8	101444 row(s) returned	265.65
MonetDB 9	3 row(s) returned	9054.64
MySQL 1	84965 row(s) returned	134.70
MySQL 2	781 row(s) returned	121.69
MySQL 3	1 row(s) returned	117.76
MySQL 4	697409 row(s) returned	121.81
MySQL 5	33856 row(s) returned	148.98
MySQL 6	33856 row(s) returned	158.33
MySQL 7	0 row(s) returned	451.45
MySQL 8	101444 row(s) returned	473.36
MySQL 9	3 row(s) returned	18184.06

VI. CONCLUSION

In this paper, we have analyzed the data stored in an RDBMS, using NoSQL technologies. However, due to their respective limitations, we were unable to use InfluxDB and MongoDB to their full potential. We have provided a baseline for analyzing smart grid data on NoSQL technologies. In the future, we aim to perform eperimentations on NewSQL [19] technologies to compare the results of RDBMS, NoSQL, and NewSQL on smart grid data.

REFERENCES

[1] Gartner, "Gartner Glossary," https://www.gartner.com/en/information-technology/glossary/big-data, [Online; accessed April 23rd, 2020].

[2] P. Zikopoulos and C. Eaton, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. McGraw-Hill Osborne Media, 2011.

[3] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The End of an Architectural Era: It's Time for a Complete Rewrite," in Proceedings of the 33rd International Conference on Very Large Data Bases. VLDB Endowment, 2007, pp. 1150–1160.

[4] A. Reeve, "Big Data and NoSQL: The Problem with Relational Databases," http://infocus. emc. com/april_reeve/big-data-and-nosql-the-problem-with-relationaldatabases/, [Online; accessed April 23rd, 2020].

[5] S. Edlich, "Your Ultimate Guide to the Non-Relational Universe!" http://nosql-database.org/, [Online; accessed April 23rd, 2020].

[6] S. Sagiroglu and D. Sinanc, "Big data: A review," in 2013 International Conference on Collaboration Technologies and Systems (CTS). IEEE, 2013, pp. 42–47.

[7] G. Stevens, "List of Nosql Database Management Systems," http://nosql-database.org/, [Online; accessed April 23rd, 2020].

[8] R. Agrawal et al., "The Claremont Report on Database Research," ACM Sigmod Record, vol. 37, no. 3, 2008, pp. 9–19.

[9] C. Hadjigeorgiou et al., "RDBMS vs NoSQL: Performance and Scaling Comparison," MSc in High, 2013.

[10] M. H. Ansari, V. T. Vakili, and B. Bahrak, "Evaluation of big data frameworks for analysis of smart grids," J. Big Data, vol. 6, 2019, p. 109.

[11] S. Venkatraman, K. Fahd, S. Kaspi, and R. Venkatraman, "SQL versus NoSQL movement with Big Data Analytics," International Journal of Information Technology and Computer Science, vol. 8, no. 12, 2016, pp. 59–66.

[12] P. O. Santos, M. M. Moro, and C. A. D. Jr., "Comparative Performance Evaluation of Relational and NoSQL Databases for Spatial and Mobile Applications," in Database and Expert Systems Applications - 26th International Conference, DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part I, ser. Lecture Notes in Computer Science, Q. Chen, A. Hameurlain, F. Toumani, R. R. Wagner, and H. Decker, Eds., vol. 9261. Springer, 2015, pp. 186–200.

[13] Oracle Corporation, "MySQL," https://www.mysql.com/, [Online; accessed April 23rd, 2020].

[14] MongoDB, Inc., "MongoDB," https://www.mongodb.com/, [Online; accessed April 23rd, 2020].

[15] MonetDB B.V., "MonetDB," https://www.monetdb.org/Home, [Online; accessed April 23rd, 2020].

[16] InfluxData Inc, "InfluxDB," https://www.influxdata.com/, [Online; accessed April 23rd, 2020].

[17] C. Kristina and D. Michael, "MongoDB: The Definitive Guide," 2010.

[18] Joson Morn, "Issues with InfluxDB," https://groups.google.com/forum/msg/influxdb/jGVE3uDStNg/9KYxjY46AQAJ, [Online; accessed 23-April-2020].

[19] A. Pavlo and M. Aslett, "What's Really New with NewSQL?" ACM Sigmod Record, vol. 45, no. 2, 2016, pp. 45–55.

 ISBN: 978-1-61208-790-0

TABLE III. QUERIES WRITTEN FOR EACH DATABASE

Name	Query
MySQL 1	SELECT * FROM dataset WHERE ID = '644'
MySQL 2	SELECT DISTINCT(ID) FROM dataset
MySQL 3	SELECT COUNT(*) FROM dataset
MySQL 4	SELECT * FROM dataset WHERE Date >= '09/01/2016' AND Date < '09/02/2016'
MySQL 5	SELECT ID,Date,SUM(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L1_015min' GROUP BY ID,Date
MySQL 6	SELECT ID,Date,avg(Value) AS average FROM dataset WHERE ResultTypeID = 'Voltage_L1_015min' GROUP BY ID,Date
MySQL 7	SELECT ID,ResultTypeID,avg(Value) AS average FROM dataset WHERE ResultTypeID = 'Current_L1_015min' or ResultTypeID = 'Current_L2_015min' or ResultTypeID = 'Current_L3_015min' GROUP BY ID,ResultTypeID Having avg(Value) > 0 AND avg(Value) < 5
MySQL 8	SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L1_015min' GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L2_015min' GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L3_015min' GROUP BY ID,Date
MySQL 9	SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = 'Current_L1_015min') UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = 'Current_L2_015min') UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = 'Current_L3_015min')
InfluxDB 1	SELECT * FROM dataset WHERE ID = '644'
InfluxDB 2	SELECT DISTINCT(ID) FROM dataset
InfluxDB 3	SELECT COUNT(Value) FROM dataset
InfluxDB 4	SELECT * FROM dataset WHERE TimeStamp >= '2016-09-01T00:00:00Z' AND TimeStamp <= '2016-09-02T23:59:59Z'
InfluxDB 5	SELECT SUM(Value) FROM dataset WHERE ResultTypeId = 'Voltage_L1_015min' GROUP BY time(1d)
InfluxDB 6	SELECT avg(Value) FROM dataset WHERE ResultTypeId = 'Voltage_L1_015min' GROUP BY time(1d)
InfluxDB 7	CREATE CONTINUOUS QUERY "meter_cq" ON "KE_SM_1" BEGIN SELECT avg(Value) AS "mean_meters" INTO "aggregate_meter" FROM dataset WHERE ResultTypeID = 'Current_L1_015min' AND ResultTypeID = 'Current_L2_015min' AND ResultTypeID = 'Current_L1_015min' GROUP BY time(1d); SELECT "mean_meters" FROM "aggregate_meter" WHERE "mean_meter" < 5
MongoDB 1	db.SM_RECS.find('ID':644)
MongoDB 2	db.SM_RECS.DISTINCT('ID')
MongoDB 3	db.SM_RECS.find().COUNT()
MongoDB 4	db.SM_RECS.find('Date' :$gt:'09/01/2016', 'Date':$lt:'09/02/2016')
MongoDB 5	db.SM_RECS.aggregate([$match: "ResultTypeID": " Voltage_L1_015min" , $group: _id: "ID", date:"Date",SUM: $SUM: "Value"])
MongoDB 6	db.SM_RECS.aggregate([$match: "ResultTypeID": " Voltage_L1_015min" , $group: _id: "ID", date:"Date",average: $avg: "Value"])
MongoDB 7	db.SM_RECS.aggregate([$or: [$match: "ResultTypeID": "Current_L1_015min","ResultTypeID": "Current_L2_015min","ResultTypeID": "Current_L2_015min"], $AND: [$avg:"Value" > 0,$avg:"Value" < 5], $group: _id: "$ID", typeID:"ResultTypeID",average: $avg: "$Value"], allowDiskUse: true)
MonetDB 1	SELECT * FROM dataset WHERE ID = '644'
MonetDB 2	SELECT DISTINCT(ID) FROM dataset
MonetDB 3	SELECT COUNT(*) FROM dataset
MonetDB 4	SELECT * FROM dataset WHERE Date >= '09/01/2016' AND Date < '09/02/2016'
MonetDB 5	SELECT ID, Date,SUM(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L1_015min' GROUP BY ID,Date
MonetDB 6	SELECT ID,Date,avg(Value) AS average FROM dataset WHERE ResultTypeID = 'Voltage_L1_015min' GROUP BY ID,Date
MonetDB 7	SELECT ID,ResultTypeID,avg(Value) AS average FROM dataset WHERE ResultTypeID = 'Current_L1_015min' or ResultTypeID = 'Current_L2_015min' or ResultTypeID = 'Current_L3_015min' GROUP BY ID,ResultTypeID Having avg(Value) > 0 AND avg(Value) < 5
MonetDB 8	SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L1_015min' GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L2_015min' GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = 'Voltage_L3_015min' GROUP BY ID,Date
MonetDB 9	SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = 'Current_L1_015min') UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = 'Current_L2_015min') UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = 'Current_L3_015min')

 ISBN: 978-1-61208-790-0