# Introduction of Time series database and exploring kdb+

Course: INFO-H-415 Advanced Databases

Project Report

Eugen Patrascu
Kunal Arora

Université Libre de Bruxelles

Université Libre de Bruxelles
http://www.ulb.ac.be

**Title:**
Introduction of Time series database and exploring kdb+

**Course:**
Advanced Database

**Project Period:**
Fall Semester 2018

**Project group:**
Kunal Arora
Eugen Patrascu

**Supervisor(s):**
Esteban Zimányi

**Page Numbers:** 30

**Date of Completion:**
December 17, 2018

**Abstract:**

Time series data is becoming more and more popular as it offers valuable insights in many industries. However, due to its unique characteristics, time series database technologies are required to efficiently store and analyse it. This report describes time series data, time series databases with kdb+ and details a financial application implemented using kdb+. It shows that usability and scalability of TSDBs are their main important features. Then it presents kdb+ as the second most popular time series database, widely used in the industry, especially in finance. Its 64-bit, columnar, in-memory architecture ensures its very high performance for both real time and historical data, way better than many other databases. Finally, various queries are run on stock trading data using kdb+'s language, q, which offers insights into the real world usage, as well as a comparison with general sql.

# Contents

# Introduction

Nowadays the volume and type of data are changing constantly and rapidly. We are dealing with larger amounts of data than ever before, often in the order of terabytes, and this data can under very different formats, structured or unstructured: tabular data, text, audio files, images etc. Another type of data which becomes more and more popular in many industries is time series data. If used properly, it can offer valuable information to many organisations. Time series data consists of values of certain parameters measures over a certain period of time, such as the evolution of stock prices, or the levels of CO2 in our homes as measured by a smart device. As this data has unique characteristics, it needs database systems optimised to deal with this type of information. Kdb+ is one of these databases, with high performance, increasing popularity, and which is used a lot in the financial services industry.

In this report, we will focus on time series data and one of the databases used to store, manage and analyse it, kdb+. It will start by defining time series data in more detail, offering more examples of where this kind of data is used currently, and will also describe the benefits of storing and analysing it. The report will next focus on time series databases, detailing the benefits, requirements and the recent trends of these systems. It will then describe the kdb+ database, its features, architecture and performance, and it will explain the characteristics that its own programming language, q, offers. The final chapter contains an application scenario of how kdb+ and q could be used in real life in the financial services industry.

# Chapter 1

# Time-series Database Fundamentals

This chapter will introduce and describe the concepts of time series data and time series databases. It starts by explaining what time series data is, where it is produced and how it can be effectively used. The second part of the chapter will focus on time series databases, presenting their properties, benefits and giving an overview of the change in popularity of time series databases over the years.

## 1.1 Time series data

### 1.1.1 Definition

Time series data "consists of repeated measurements of parameters over time together with the times at which the measurements were made." [1] The nature of these measurements, or data points, can vary greatly depending on the use case, but they are usually represented as numbers.

### 1.1.2 Examples of time series data

There are many industries and scenarios where time series data is used. A popular application is in finance, where the evolution of stock and commodities prices over time can be represented as time series. Sensor measurements are time series data as well: GPS data from our mobile phones, temperature and humidity levels from smart home devices, or environmental measurements data such as levels of CO2 or wind direction. Similarly, monitoring data can be produced, for example the CPU load, level of utilised disk storage and usage of memory in the case of data centres, or health rate and blood pressure variations in the case of healthcare data. There are many other examples of time series data, such as taxicab ride data, fuel consumption, or web event streams, therefore any data that collectively models how a system or process varies over time represents time series data[1].

[1] https://blog.timescale.com/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-database-dcf3b1b18563

### 1.1.3 Use of time series data

There are various use cases in which time series data can be helpful. The timeseries data by itself does not provide much information unless it is analysed: observing a measurement value at a certain point in time is less interesting than observing its variation over time. For example, in finance it would be important to observe how specific stock prices change depending on seasonality to see if they go through up and downs at regular times each year. Another use of the data could be in time series forecasting, where we use historical values of the parameters and their correspondent trends to try and predict future behaviour (e.g. such as how the price of a specific stock would behave in the next quarter)[2].

Based on the previous examples we can observe certain aspects which apply to any type of time series data: new data that arrives generally represents new data points (rather than an update of old measurements), there is a time order between the values, and the time is the primary axis of the dataset. Additionally, time series applications make use of very large amounts of data, as the speed of inserting new data points may be very high. These constrains are important in deciding how we store, manage and analyse time series data and thus they must be considered in designing suitable technologies.

## 1.2 Time series databases

### 1.2.1 Definition

A time series database (TSDB) is a software system optimised for handling time series data, considering the above-mentioned challenges: constantly increasing large volumes of data and a specific format. TSDBs allow the basic operations of creation, read, update and deletion of time series, as well as other operations. Some examples of those include the addition and multiplication of series, or other types of combinations of multiple time series into one, selections of time ranges, filters of high and low values, and possibly other statistical functions that can be applied to time series data[1].

Considering the requirements regarding the storage and retrieval of time series, in a survey from 2017 regarding the type of databases used for storing time series data 58% of the respondents replied that they used a database engine purposely-built for time series data, while 42% used other types of database. [3].

### 1.2.2 Benefits of time series databases

The reasons the majority of the people use TSDB for time series data is because of the following benefits of TSDB:

[2] https://www.investopedia.com/terms/t/timeseries.asp

[3] Percona, February 2017. https://www.percona.com/blog/2017/02/10/percona-blog-poll-database-engine-using-store-time-series-data/

1. **Scalability**

   Time series data scales up very quickly, and an effective TSDB is able to store and analyse continuously growing large data sets of millions of data points. Relational databases do not perform well on very large datasets, and while NoSQL databases are designed to perform at scale, their performance can be improved substantially when the focus is on the time dimension. This is what is achieved by using time series databases, which are optimised for dealing with time series data. Improvements are noticeable in the areas of data compression, ingest rates and speed of queries [4]

2. **Usability**

   Time series databases contain numerous useful commands and functions that are typically applied to time series data: aggregations of time series, range queries, value filters, continuous queries. The queries are constructed to provide a good user experience, so they provide value even when the data scale is not an issue. For these reasons TSDBs are used in a large number of systems and applications: financial software for trading (e.g. shares, commodities), applications collecting web traffic data (e.g. web logs), software that monitors physical systems (e.g. in data centres to monitor severs, health systems monitoring the health conditions, IoT devices), business decision making systems etc.

3. **Reduced downtime**

   The architecture of time series databases ensure availability of data in cases of hardware break-down or network partitions, in the cases where downtime is not acceptable.

4. **Lower expenses**

   As explained above, the resiliency of the TSDB means that there will be lower costs in case of outages. Similarly, scaling is also affordable and efficient by employing commodity hardware.

5. **Better business decisions**

   By using a time series database, a business is able to analyse data in real time, allowing them to make informed and timely decisions regarding resources consumption, systems maintenance or other business-relevant issues.

### 1.2.3 Properties of time series databases

A performant time series database would have the following properties [5]:

[4] https://blog.timescale.com/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-database-dcf3b1b18563

[5] http://basho.com/resources/time-series-databases/

1. **Data location**

   When the time series data is located in multiple machines, a TSDB should be able to co-locate data in related time ranges to the same physical part of the database so that the query execution will be faster. Otherwise, the data access will be slower and make the operations inefficient.

2. **Fast, easy range queries**

   Analysing range queries is an important operation for time series data and in the case of very large data sets many databases would perform very slowly. A time series database needs to read and write efficiently using the data location property mentioned above, and the queries should be easy to write and understand for the users.

3. **Data granularity**

   A TSDB should be able to allow the user to modify the levels of granularity of the data depending on the business needs, because as time series data gets old different granularity levels might be needed.

4. **High write performance**

   When dealing with time series data we are considering very large data sets that pile up fast, therefore the TSDB should have high availability and high performance for both read and write operations even during peak periods.

## 1.3 Popularity and trends in time series databases

Considering the benefits of using TSDB in the current context it is not surprising to see that time series databases have risen greatly in popularity in the last 24 months, as in figure 1.1. They are well above the popularity of other types of databases, with graph databases and key-value stores completing top 3. The popularity has been almost constantly rising in the past 2 years and the trend seems to continue like this. Another interesting thing to observe is that the popularity of relational databases is on a decreasing trend.

In table 1.2 we can observe the popularity of various time series databases. Influx DB dominates the list with a constant increase in popularity, while kdb+ is second in the list, with a recent strong growth in usage. While there is indeed a substantial difference between the popularity of Influx DB and kdb+, the usage scores confirm the stability of kdb+ as the second most used time series database.

Chart 1.2 shows the evolution of trends of time series databases over the past 5 years. As we can observe, kdb+ has been around for longer than the majority of the other time series databases, and its popularity has been constantly rising.

The rest of the report will focus on kdb+ as a time series database, detailing features, benefits, performance and demonstrating its usage in a financial application context.

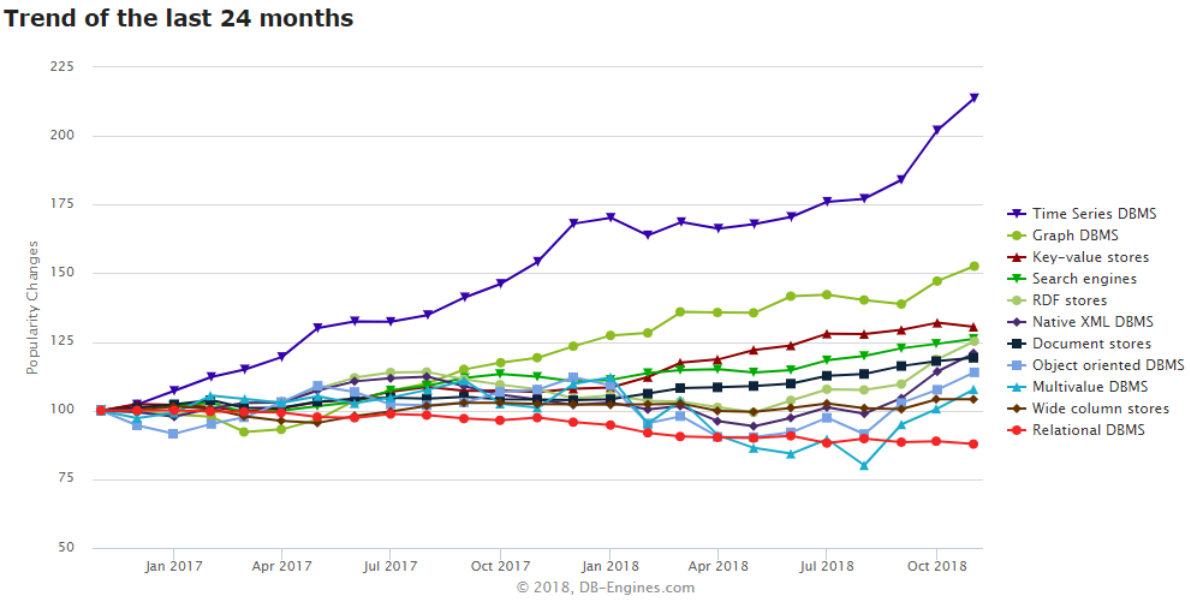


**Figure 1.1:** Line chart displaying the popularity changes in databases trends over the past 24 months (Source: https://db-engines.com/)

| Rank | | | DBMS | Database Model | Score | | |
|---|---|---|---|---|---|---|---|
| Nov 2018 | Oct 2018 | Nov 2017 | | | Nov 2018 | Oct 2018 | Nov 2017 |
| 1. | 1. | 1. | InfluxDB | Time Series DBMS | 13.64 | +0.66 | +4.30 |
| 2. | 2. | ↑ 4. | Kdb+ | Multi-model | 4.84 | +0.47 | +2.99 |
| 3. | 3. | 3. | Graphite | Time Series DBMS | 2.85 | +0.04 | -0.01 |
| 4. | 4. | ↓ 2. | RRDtool | Time Series DBMS | 2.73 | +0.05 | -0.47 |
| 5. | 5. | 5. | OpenTSDB | Time Series DBMS | 2.02 | +0.07 | +0.32 |
| 6. | 6. | ↑ 7. | Prometheus | Time Series DBMS | 1.95 | +0.24 | +1.14 |
| 7. | 7. | ↓ 6. | Druid | Time Series DBMS | 1.36 | +0.04 | +0.38 |
| 8. | 8. | | TimescaleDB | Time Series DBMS | 0.54 | -0.03 | |
| 9. | 9. | ↓ 8. | KairosDB | Time Series DBMS | 0.49 | -0.05 | +0.02 |
| 10. | 10. | ↓ 9. | eXtremeDB | Multi-model | 0.30 | +0.00 | +0.00 |
| 11. | 11. | ↓ 10. | Riak TS | Time Series DBMS | 0.27 | -0.02 | +0.06 |
| 12. | ↑ 13. | ↑ 14. | Axibase | Time Series DBMS | 0.21 | +0.04 | +0.12 |
| 13. | ↓ 12. | ↓ 11. | FaunaDB | Multi-model | 0.21 | +0.02 | +0.04 |
| 14. | ↑ 16. | ↑ 15. | GridDB | Multi-model | 0.17 | +0.06 | +0.08 |
| 15. | ↓ 14. | ↑ 18. | Warp 10 | Time Series DBMS | 0.14 | +0.02 | +0.09 |

**Figure 1.2:** Table displaying the current popularity scores and ranks of time series databases, as well as their scores and ranks in the previous month and previous year (Source: https://db-engines.com/)

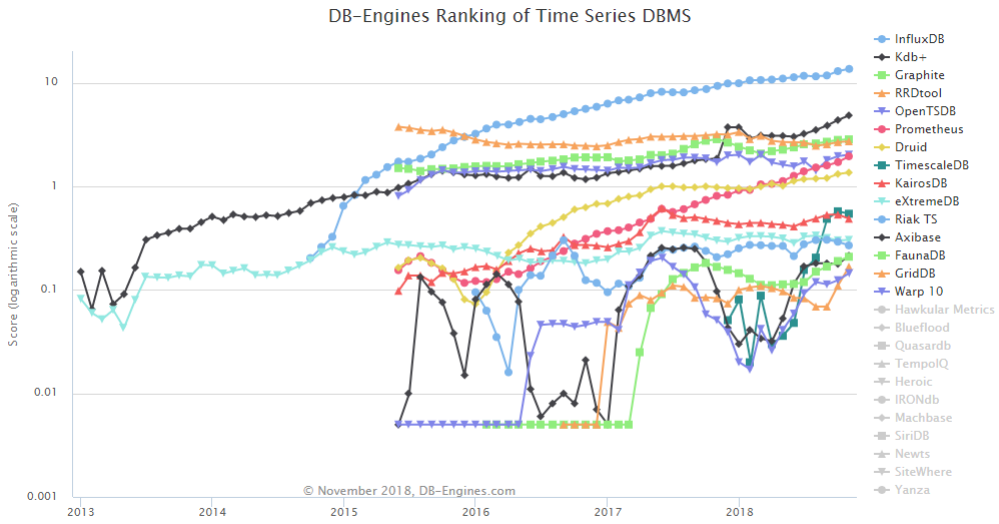


**Figure 1.3:** Chart displaying the change in popularity of time series databases over the past 5 years (Source: https://db-engines.com/)

# Chapter 2

# kdb+ database

Last chapter briefly introduced the time-series database technology which is needed to follow this report. This chapter is going to explain about one of the very popular time-series database called, kdb+ along with it's powerful and expressive query language called, Q. Due to its high demand in the market it has been a topic of interest and leading time-series database technology in almost all major financial institution.

## 2.1 General features

Through this section, we are going to explore some basic features of kdb+ database and q query language. kdb+ is a commercial column-based relational time-series database with in-memory abilities. The column-based database architecture is primarily useful when performing aggregate queries like, sum, count, avg etc. Backing up this database, is it's Q query language. According to it's developers [4], the primary design objectives of Q are efficiency, speed and expressiveness.

Few key features of kdb+ are as follows:

- **Column-oriented :** The tables in kdb+ are stored as columns and applies an operation to an entire column vector. This is specially useful when you have billions of records and want to aggregate based on columns which is a general case in financial data analysis.
- **In Memory database :** One of the important features that give speed to kdb+ is it's in memory storage and manipulation of the data. Since the tables are stored in memory and data manipulation is performed in memory with Q, there is no need for stored procedure language. In memory storage makes it really fast but it requires a lot of RAM which is no longer a major problem as servers with high RAM are now inexpensive.
- **Ordered lists :** In kdb+ database, all the rows are stored in a specific order as ordered lists are the foundation of all data structures. This makes processing the large amount of data very fast and efficient.

## 2.2 Architecture

Kdb+ is a high-performance, high-volume database designed to handle vast millions and billions of records of data. Kdb+ architecture allows capturing and processing of real-time, statistical and historical data. It runs on Windows, MacOS, Linux, and Solaris 32 64-bit server platforms and has wide range of possibility of storage architecture consisting of, local disks, In memory RAM, SANs and NAS [4].

Example of a typical kdb+ architecture in a financial trading application:

- The data from companies like, Bloomberg etc. or directly from Exchange is fed to the Ticker plant.
- Before Ticker plant start it execution, the data from Exchange is parsed by Feed Handler in order to get the relevant data for Ticker plant to process.
- In order to provide recovery management support, the ticker plant first stores the data in Logfiles and then updates its own tables.
- After the data has been stored in logfiles and updated in the ticker plant tables, it is sent/published to the real-time database and sent to subscribers who asked for it.
- At the end of certain time period typically, a business day all the data from the real-time database is transferred to historical database. After the data is saved to historical database, all the tables are deleted from the real-time database.
- Simple API interfaces in various languages (C/C++, Java, Python, R etc.) make for easy connectivity to external reporting, graphing and visualization of data for analysis purposes.
- The architecture also supports large-scale distributed architectures such as cloud.
- The kdb+ supports scalability, security, high-availability fail-over, transaction logging and accessibility.

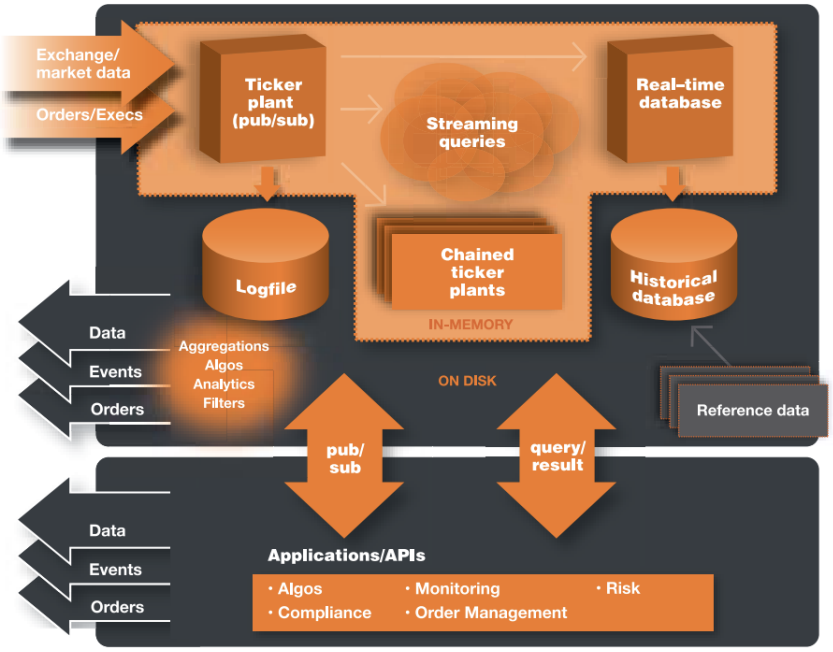


**Figure 2.1:** Typical kdb+ architecture in financial trading application.

## 2.3 Overview of Q language

Q is the programming language to work with kdb+ database. It is because of this interpreted language support that the retrieval and manipulation of data is significantly faster. Q expressions can be entered and executed in the q console, or loaded into the q script, which is a text file with extension *.q*
It is a vector-based processing language and due to this feature, it is well suited to perform complex calculations on large volumes of data with less overhead. It's syntax allow the select expressions and also, contain a rich super-set of built-in functions for easy processing.

### 2.3.1 Key features

- **Interpreted :** Q is interpreted rather than compiled. All the data functions and queries live on the main memory.
- **Evaluation order :** Expressions are evaluated from right-to-left meaning that, functions and operator to the left executes on what is to the right of it. This means that parenthesis are rarely needed to resolve operation order as there is no operator precedence.
- **Types :** It is a strongly typed language. There is no explicit declaration of variables rather, the type of variable name reflects the value assigned to it.
- **Null Values :** In Q, different types have different Null values. Arithmetic operations can involve infinite and null values in order to produce result.

- **Built-in Time data types :** As Q have built-in time data types support, it is highly optimized for time-series analysis operations.
- **Support for SQL queries :** Q support a lot of database queries which are similar to SQL counterparts to provide the backward compatibility with SQL query paradigm.
- **Quick reflexivity :** It provides a quick immediate robust feedback for faster development and testing purposes.

### 2.3.2 Basic CRUD operations

As tables form the basis of kdb+ database and are considered the most important data structure to store and manipulate data. So, will apply CRUD (create, read, update and delete) operations on table data structure.

- **Create**
  Creating table is very straight forward process in kdb+ database. As kdb+ is a key-value store so, we specify the key for each column of table and their specific values are followed by the key.*The columns must be of the same length.*

```
person:([]firstname:`Micheal`Karl`Bobby`Woddy;lastname:`Jordan`Marx`
    Deol`Woodland; age:`23`45`12`34; salary:`2000`6000`4500`4000;
    sex:`F`M`F`M)
```

```
q)person:([]firstname:`Micheal`Karl`Bobby`Woody;lastname:`Jordan`Marx`Deol`Woodl
and;age: 23 45 12 34; salary:2000 6000 4500 4000; sex:`F`M`F`M)
q)person
firstname lastname age salary sex
---------------------------------
Micheal   Jordan   23  2000   F
Karl      Marx     45  6000   M
Bobby     Deol     12  4500   F
Woody     Woodland 34  4000   M
```

**Figure 2.2:** Create (operation) the person table

- **Read**
  Reading is performed simply by using select statement on the table.

```
select sex,firstname, lastname, salary from person where age<30
```

```
q)select sex,firstname, lastname, salary from person where age<30
sex firstname lastname salary
-----------------------------
F   Micheal   Jordan   3000
F   Bobby     Deol     4500
```

**Figure 2.3:** Read (operation) from person table

- **Update**
  Update operation will modify some rows or add additional column to the

table and will return the resulting modified table. The update query is very similar to the select query.

```
person:update salary:salary+1000 from person where firstname in `
    Micheal`Steve
```

```
q)person:update salary:salary+1000 from person where firstname in `Micheal`Steve
q)person
firstname lastname age salary sex
--------------------------------
Micheal   Jordan   23  3000   F
Karl      Marx     45  6000   M
Bobby     Deol     12  4500   F
Woody     Woodland 34  4000   M
```

**Figure 2.4:** Update (operation) on person table

- **Delete**
  Delete operation is also similar to one in sql. It deletes some rows or the whole column from the kdb+ database.

```
1 person:delete from person where firstname in `Bobby
2 person:delete age from person
```

```
q)person:delete from person where firstname in `Bobby
q)person
firstname lastname age salary sex
--------------------------------
Micheal   Jordan   23  3000   F
Karl      Marx     45  6000   M
Woody     Woodland 34  4000   M
```

**Figure 2.5:** Delete (operation) a particular row value from person table

```
q)person:delete age from person
q)person
firstname lastname salary sex
----------------------------
Micheal   Jordan   3000   F
Karl      Marx     6000   M
Woody     Woodland 4000   M
```

**Figure 2.6:** Delete (operation) a column from person table

## 2.4 Installation and Dev environment setup

The installation and development environment setup process of kdb+ is pretty straight forward. Although kdb+ is a commercial software, they still have 32-bit version available to download for trial purpose for almost all major OS's including, linux, Windows, MacOS etc.

- Download the 32-bit trial verison from Kx website.[1]
- After downloading, if necessary, unzip the archive. A new folder named q will appear in the Downloads folder.
- Once downloaded, open the terminal and write following commands to run kdb+.

```
[kunalarora@Kunals-MacBook-Pro ~ $ cd Downloads/
[kunalarora@Kunals-MacBook-Pro Downloads $ cd q
[kunalarora@Kunals-MacBook-Pro q $ ls
README.txt      q.k             s.k             trade.q
m32             q.q             sp.q
[kunalarora@Kunals-MacBook-Pro q $ m32/q
KDB+ 3.6 2018.10.23 Copyright (C) 1993-2018 Kx Systems
m32/ 4()core 8192MB kunalarora kunals-macbook-pro.local 192.168.0.100 NONEXPIRE


Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com
To exit, type \\
To remove this startup msg, edit q.q
q)
```

**Figure 2.7:** Launching kdb+ in terminal

- Try to play around with q commands to ensure that kdb+ is successfully installed and running. See the below screen for some examples.

```
q)til 6
0 1 2 3 4 5
q)1 + 2
3
q)c:7
q)c
7
q)\\
kunalarora@Kunals-MacBook-Pro q $
```

**Figure 2.8:** Confirm successful installation

- To exit from kdb+ session, type \\
- This confirms that 32-bit version is successfully installed and setup locally.
- To install and download the 64-bit kdb+ version, you need to obtain the licence key from Kx Systems website.[2]

[1] https://kx.com/download/
[2] https://code.kx.com/q/tutorials/licensing/

## 2.5 Performance characteristics

Kdb+ can be efficiently used for both real time and historical data, offering a much better performance than other competitive databases (as shown in next section). It deals with the whole process composed of capturing the data, storing it, analysing it, etc, thus getting rid of the latency of performing these steps separately [4].

Kx Systems documents the following performance features[4]:

- Efficient join and indexing: its columnar setup ensures much faster search
- Fast time series ordered queries: being built for time series data, time ordered analysis is dramatically fast
- Distributed queries: it offers support for the queries to be be efficiently parallellized across multiple cores/systems due to its 64-bit architecture
- No limit on the number of clients: it uses chained servers to distribute the processing from the main server via publish-subscribe mechanisms
- Efficient for real time data: due to dynamic indexes
- Efficient for historical data: can easily handle trillions of records in historical databases

## 2.6 Performance benchmark

Kparc.com [2] has realized a performance benchmark for kdb+ and comparison with various other databases.

**Experiment data**: The data on which the experiments were performed is NYSE data, whose original data sets consist of: 65 billion trades and 1.1 trillion quotes, that being 100TB of data over 5000 days.

**Databases considered**: There were various databases that were bench-marked in this experiment, together with q. The databases were either column-store: vertical, big3accel, hadoop/impala, greenplum or row-store: postgres, big3rdbms, mongodb, spark.

**Queries**: 4 classic queries were used for the comparison: queries q1, q2, and q3 containing aggregations on top 100 symbols S (which is 25% of data), and q4 being an asof-join (where price is less than the current bid).

The queries run were the following [3]. The q version:

```
S:100#first flip idesc select count i by sym from trade where date=d
Q1: select last bid  by sym        from quote where date=d,sym in S
Q2: select max price by sym,ex     from trade where date=d,sym in S
Q3: select avg size  by sym,time.hh from trade where date=d,sym in S
Q4: select time,price,bid from aj[`time;select time,price from trade
  where date=d,sym=`QQQ;select time,bid from quote where date=d,sym=`
  QQQ] where price<bid
```

[3] http://kparc.com/q4/q4.txt

And the correspondent sql version:

```
create table S(sym char(4));insert into S select sym from trade where
  date=d group by sym order by count(*) desc fetch first 100 rows only
Q1: select sym,last(bid) from quote natural join S where date=d group by
  sym;
Q2: select sym,ex,max(price) from trade natural join S where date=d
  group by sym,ex;
Q3: select sym,hour(time),avg(size) from trade natural join S where date
  =d group by sym,hour(time);
Q4: select * from (select time,price,sym from trade where date=d and sym
  =`QQQ) t
  left outer join (select time,bid,sym from quote where date=d and sym
  =`QQQ) q
  on q.time=(select max(time) from q where time<=t.time and sym=`QQQ)
  where price<bid;
```

**Experiment conditions**: The machine on which the queries were performed had 16 cores and 256 GB RAM. It is also important to mention that the data contained partition on date, index on symbol and all queries were cached in RAM (no disk access).

**Experiments**: There were 3 different experiments run, on datasets of various sizes: 40 million rows, 640 million rows and 10 billion rows as we can see in tables 2.9 and 2.10.

SMALL day (40Million rows) times in milliseconds.

| | Q1 | Q2 | Q3 | Q4 | RAM(GB) | ETL | DSK(GB) |
|---|---|---|---|---|---|---|---|
| q | 17 | 12 | 36 | 3 | .2 | 40 | 1.0 |
| COLSTORE | | | | | | | |
| vertical | 500 | 140 | 150 | 8900 | 2.1 | 52 | .5 |
| greenplum | 4600 | 180 | 200 | DNF | 5.5 | 73 | 4.6 |
| impala | 4800 | 1190 | 1000 | DNF | 4.0 | 22 | .3 |
| big3accel | 4200 | 1600 | 2300 | DNF | 3.4 | 20 | 1.0 |
| ROWSTORE | | | | | | | |
| postgres | 7100 | 1500 | 1900 | DNF | 1.5 | 200 | 4.0 |
| big3rdbms | 6400 | 2200 | 3100 | DNF | 5.0 | 60 | 2.0 |
| mongodb | 8900 | 1700 | 5800 | DNF | 9.0 | 922 | 10.0 |
| spark/shark | 34000 | 7400 | 8400 | DNF | 50.0 | 156 | 2.4 |

**Figure 2.9:** Experiment 1 and its results

**Observations**: for ETL there are seconds to load, and DNF represents 'Did Not Finish'. The overhead is milliseconds for fast queries.

**Conclusions**: For these tasks of time series data analytics, q has a much better performance than all the databases benchmarked. The results depends on the queries, but in the case of the smaller table of 40 mil rows, in general, q is 10+ times faster than the column store databases, and 100+ times faster than the row store databases. Moreover, q has a much lower overhead (10-1000 times faster). It also uses 10 times less RAM, and it has better locality as it uses less SSD and

```
SCALE UP  (640Million rows: 1 big day)

q              290      33     130      29
impala       45000    2250    1880     DNF
greenplum    56000     900    1000     DNF

SCALE OUT (10Billion rows: 16 big days)

q              820      70     180      70
impala    10000000   18000   27000     DNF
```

**Figure 2.10:** Experiments 2 and 3 and their results

disk space. In table 2.10 we can see that q also scales much better (than impala for example, whose performance decreases dramatically over much larger datasets). In general, most databases perform poorly on Q1 and cannot complete Q4.

## 2.7 Which industry is using kdb+ software the most?

Kdb+ software is highly used by companies all over the world but due to it's ability of storing and manipulating tick data efficiently, financial companies use this software to solve their complex statistical and financial problems and day-to-day analysis.
Example of companies which are leading the financial market that are highly dependent on using kdb+ as their enterprise wide tick storage system:

- JP Morgan Chase and Co.
- Bank of America
- Barclays Capital
- Morgan Stanley group incorporation
- Bank of Tokyo-Mitsubishi UFJ ltd.
- The Royal Bank of Scotland

### 2.7.1 Use-case scenario of kdb+ of 3 major companies

Below are the major projects with slight description of three different financial firms which are using kdb+ to enhance their work force and make better analysis.[5]

**Barclays Capital**

- Project name: BATS (Barclays Algorithmic Trading System)
- Description: Distributed multi-server kdb+tick like setup, with support for multiple asset classes, credit, FX, fixed income.

**Morgan Stanley**

- Project name: Horizon
- Description: The Horizon system provides a holistic time series infrastructure using a single database technology, KDB+/Q, with consistent tools for data acquisition/loading, data quality, and production support that covers a broad range of asset classes, data types, and frequencies that meet current and future trading, analytical and operational needs for all MS users.

**JPMorgan Chase and Co.**

- Project name: TicDB
- Description: TicDB captures and stores market data for equities/futures in all markets across the globe, efficient access is made available to clients globally and real time and historical analytics are provided.

## 2.8 Kdb+ weakness

As it's very clear by now that kdb+ database has high demand in the financial institutions and reliable as it is backed by Kx systems and out in the market for almost 20 years. The performance has been impeccable as compared to other Time-series databases for even very large data sets. Given that, every database/technology have their pros and cons depending on the kind of problem we are dealing with. This section will explore some of the weaknesses of kdb+ database.

- **Hard to read:** Through q language support it becomes really easy to write short script for complex problems which is good. But it makes the code look very cryptic, which are very difficult to maintain and understand.
- **Steep learning curve:** Although q language use same semantics of SQL but, it could be really hard to learn it in the beginning and write efficient code.
- **Costly:** As kdb+ is a commercial software, for small scale companies it could be costly to purchase when compared with, taking the full advantage of the software.

# Chapter 3

# Application using kdb+ and q

In this chapter we will present how the kdb+ database and its own language, q, can be used in real life scenarios when dealing with financial data. In this scenario we are focusing on the analysis of share trades of certain companies and the queries that could potentially be applied in the case of this analysis.

## 3.1 Data Generation

One important thing about q is that it makes it easy to generate datasets with certain characteristics, which can be useful in the cases when you want to run experiments, to benchmark etc. In our scenario we decided to generate data as it is relatively hard to find open data of the nature that we wanted: volumes of shares of trades for certain companies at the specific time when they happened. The majority of the datasets regarding share prices and trades have a much higher granularity (e.g. day level), present only the selling prices or are not for free.

The data we will generate will have the following format: will contain the date, the time in 24h format, the company ticker, the volume of shares traded and the price at which the shares were bought. Our generation of the data will work by generating each column separately and then combining them into a table. In q we are using the 'list' data type to generate data.

We start by generating one million of dates in November 2018:

```
dates:2018.11.01+1000000?30
```

Now let's generate 1 million time points in the 24h format:

```
times:1000000?24:00:00.000000000
```

We generate next 1 million company tickers/symbols for Microsoft (MSFT), Facebook (FB) and Apple (AAPL):

```
symbols:1000000?`msft`fb`aapl
```

We continue by generating the volumes of the transactions for the 1 million data points, which will be multiples of 10:

```
volumes:10*1+1000000?1000
```

Lastly, we generate the share prices at the transactions times. Considering the generated dates, the price of a MSFT share was around 100$. We are going to uniformly distribute the prices within 10% of 100, and we are going to update the prices for the other companies later :

```
prices:90.0+(1000000?2001)%100
```

The results in the console look like this:

```
q)dates:2018.11.01+1000000?30
q)times:1000000?24:00:00.000000000
q)symbols:1000000?`msft`fb`aapl
q)volumes:10*1+1000000?1000
q)prices:90.0+(1000000?2001)%100
q)dates
2018.11.03 2018.11.04 2018.11.30 2018.11.12 2018.11.09 2018.11.09 2018.11.07
2018.11.16 2018.11.20 2018.11.04 2018.11.04 2018.11.23 2018.11.19 2018.11.15
2018.11.06 2018.11.16 2018.11.25 2018.11.11..
q)times
0D21:44:08.513676971 0D00:48:25.967822670 0D19:00:35.506331920 0D23:58:05.102
916508 0D19:20:22.538215667 0D05:38:22.739987522 0D15:04:22.221752107 0D10:47
:42.584379315 0D09:27:23.958252668 0D12:00:..
q)symbols
`msft`msft`msft`aapl`msft`msft`msft`msft`msft`msft`aapl`msft`msft`aapl`fb`aap
l`msft`aapl`msft`aapl`aapl`fb`aapl`msft`fb`fb`fb`msft`fb`aapl`msft`aapl`fb`aa
pl`aapl`aapl`fb`aapl`aapl`aapl`aapl`aapl`ms..
q)prices
102.64 97.73 92.78 105.97 92.77 91.21 106.58 92.16 106.4 103.56 99.49 95.66 1
09.99 93.68 95.96 95.18 96.29 109.35 97.07 100.24 90.35 91.46 97.14 101.29 10
4.21 93.19 95.62 93.81 99.75 94.32 105.4 97..
q)volumes
3710 820 2370 4570 7710 4090 2860 5000 4020 1720 5290 9750 830 8720 7490 7170
 5320 5500 60 8040 6040 7660 9650 3940 10000 6550 4720 880 2970 6220 5020 611
0 7960 6990 7270 9360 9260 1260 8950 5030 1..
```

**Figure 3.1:** The generation of trades data in lists

After the data was generated, we now combine the lists into a table:

```
trades:([] date:dates; time:times; symbol:symbols; volume:volumes; price:
    prices)
```

```
q)trades:([] date:dates; time:times; symbol:symbols; volume:volumes; price:prices)
q)5#trades
date       time                 symbol volume price
-----------------------------------------------------
2018.11.03 0D21:44:08.513676971 msft   3710   102.64
2018.11.04 0D00:48:25.967822670 msft   820    97.73
2018.11.30 0D19:00:35.506331920 msft   2370   92.78
2018.11.12 0D23:58:05.102916508 aapl   4570   105.97
2018.11.09 0D19:20:22.538215667 msft   7710   92.77
```

**Figure 3.2:** Generation of trades table

As we can see, the trades are not ordered. That can be done very easily:

```
trades:`date`time xasc trades
```

Let's inspect the table again:

```
q)trades:`date`time xasc trades
q)5#trades
date       time                 symbol volume price
-----------------------------------------------------
2018.11.01 0D00:00:04.544352740 fb     3320   93.81
2018.11.01 0D00:00:06.795980036 aapl   9840   105.28
2018.11.01 0D00:00:07.384952902 fb     8730   108.14
2018.11.01 0D00:00:08.218482881 fb     9700   94.05
2018.11.01 0D00:00:08.474667370 fb     7380   90.03
```

**Figure 3.3:** Ordering of the trades by date and by time

A few more updates and the table will be ready and resemble more the real life data. We have to update the share prices of Facebook and Apple. At the beginning of November 2018, a Facebook share was traded for about 150$ and an Apple one for about 200$. We update the values in the table as follows:

```
trades:update price:1.5*price from trades where symbol=`fb
trades:update price:2*price from trades where symbol=`aapl
```

Finally, the generated data looks like this:

```
q)trades:update price:1.5*price from trades where symbol=`fb
q)trades:update price:2*price from trades where symbol=`aapl
q)8#trades
date       time                 symbol volume price
-----------------------------------------------------
2018.11.01 0D00:00:04.544352740 fb     3320   140.715
2018.11.01 0D00:00:06.795980036 aapl   9840   210.56
2018.11.01 0D00:00:07.384952902 fb     8730   162.21
2018.11.01 0D00:00:08.218482881 fb     9700   141.075
2018.11.01 0D00:00:08.474667370 fb     7380   135.045
2018.11.01 0D00:00:09.738128632 fb     7540   158.16
2018.11.01 0D00:00:11.107282340 fb     7500   157.365
2018.11.01 0D00:00:11.808726936 fb     2190   145.47
```

**Figure 3.4:** Final table after share price updates

In order to ensure that the data was generated correctly, we can do some sanity checks for the 'price' and 'volume' columns. If we take the averages, minimum and maximum values of these columns by symbols, they should correspond to the parameters we passed during the data generation step:

```
select avg price, avg volume by symbol from trades
select min price, max price by symbol from trades
```

Indeed, they do:

```
q)select avg price, avg volume by symbol from trades
symbol| price    volume
------| -----------------
aapl  | 199.9983 4999.706
fb    | 150.0098 5000.961
msft  | 100.0126 5002.446
q)select min price, max price by symbol from trades
symbol| price price1
------| ------------
aapl  | 180   220
fb    | 135   165
msft  | 90    110
```

**Figure 3.5:** The values correspond to our data generation and data update queries

## 3.2 Queries

Queries in q are, in general, similar in syntax with SQL, which makes them easy to learn. At the same time, they are simpler, shorter and more powerful, extending the capabilities of sql with features useful for time series data [3].

For example, a select q query would normally have the following syntax:

```
select [col1] [by col2] from tbl [where cond]
```

While the SQL equivalent would be something like:

```
select [col2] [col1] from tbl [where cond] [group by col2 order by col2]
```

### 3.2.1 Basic queries with constraints

1. Return all Facebook trades

   ```
   select from trades where symbol in `fb
   ```

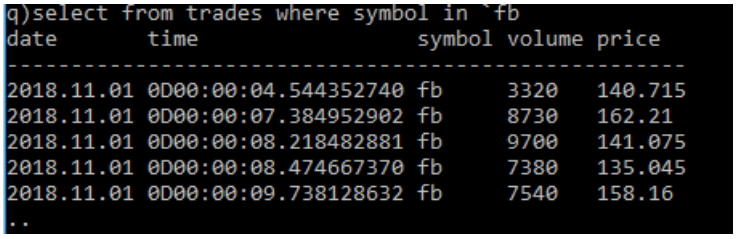


**Figure 3.6:** Query 1 and its result

2. Return all Facebook trades on a particular date

   Another interesting feature of q is the declaration of variables, which can be later used inside queries:

   ```
   dt: 2018.11.05
   select from trades where date=dt,symbol=`fb
   ```

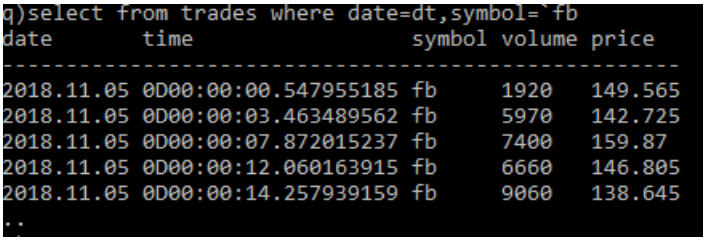


**Figure 3.7:** Query 2 and its result

3. Return all Facebook trades with a share price between 145 and 155

   ```
   select from trades where symbol=`fb, price > 145.0, price < 155.0
   ```

4. Return all Facebook trades between 19.00 and 19.10, in the evening, on a particular day

```
q)select from trades where symbol= fb, price > 145.0, price < 155.0
date       time                 symbol volume price
-----------------------------------------------------
2018.11.01 0D00:00:11.808726936 fb     2190   145.47
2018.11.01 0D00:00:16.746418923 fb     3850   146.82
2018.11.01 0D00:00:55.160050839 fb     4890   154.65
2018.11.01 0D00:01:05.160881727 fb     6580   154.335
2018.11.01 0D00:01:07.410356551 fb     2300   154.2
..
```

**Figure 3.8:** Query 3 and its result

```
select from trades where date = 2018.11.05, symbol = `fb, time >
    19:00:00.000,time < 19:10:00.000
```

```
q)select from trades where date = 2018.11.05, symbol = `fb, time > 19:00:00.000,time
 < 19:10:00.000
date       time                 symbol volume price
-----------------------------------------------------
2018.11.05 0D19:00:04.896218329 fb     8480   135.36
2018.11.05 0D19:00:05.195029824 fb     5880   160.485
2018.11.05 0D19:00:08.099721372 fb     1320   156.285
2018.11.05 0D19:00:08.756265789 fb     2880   142.005
2018.11.05 0D19:00:39.856781065 fb     8570   151.275
..
```

**Figure 3.9:** Query 4 and its result

5. Return all Facebook trades in ascending order of price within a certain time period

```
`price xasc select from trades where date within 2018.11.10
    2018.11.13, symbol =`fb
```

```
q)`price xasc select from trades where date within 2018.11.10 2018.11.13, symbol =`fb
date       time                 symbol volume price
-----------------------------------------------------
2018.11.10 0D08:34:13.201731294 fb     5350   135
2018.11.10 0D15:03:51.636220961 fb     8960   135
2018.11.10 0D17:50:37.841210514 fb     510    135
2018.11.11 0D12:30:09.735587239 fb     6260   135
2018.11.11 0D14:25:40.874335169 fb     130    135
..
```

**Figure 3.10:** Query 5 and its result

6. Return all Facebook or Apple trades

```
select from trades where symbol in `fb`aapl
```

7. Calculate count of all symbols within a certain time period and order descendingly

```
`numsymbol xdesc select numsymbol: count i by symbol from trades
    where date within 2018.11.10 2018.11.13
```

8. Find the maximum price of a Facebook share within a time period, and when this first happen

```
q)select from trades where symbol in `fb`aapl
date       time                 symbol volume price
-------------------------------------------------
2018.11.01 0D00:00:04.544352740 fb     3320   140.715
2018.11.01 0D00:00:06.795980036 aapl   9840   210.56
2018.11.01 0D00:00:07.384952902 fb     8730   162.21
2018.11.01 0D00:00:08.218482881 fb     9700   141.075
2018.11.01 0D00:00:08.474667370 fb     7380   135.045
..
```

**Figure 3.11:** Query 6 and its result

```
q)`numsymbol xdesc select numsymbol: count i by symbol from trades where date within 2
018.11.10 2018.11.13
symbol| numsymbol
------| ---------
aapl  | 44732
fb    | 44692
msft  | 44492
```

**Figure 3.12:** Query 7 and its result

```
select date, time,price from trades where date within 2018.11.10
    2018.11.13, symbol =`fb, price= exec first price from select max
     price from trades where symbol =`fb
```

```
q)select date, time,price from trades where date within 2018.11.10 2018.11.13, symbol
=`fb, price= exec first price from select max price from trades where symbol =`fb
date       time                 price
-----------------------------------
2018.11.10 0D00:09:12.112696319 165
2018.11.10 0D00:49:58.669512569 165
2018.11.10 0D01:04:42.442329078 165
2018.11.10 0D14:32:03.846387863 165
2018.11.10 0D21:16:23.349222689 165
..
```

**Figure 3.13:** Query 8 and its result

9. Select the last price for each symbol in hourly buckets

```
select last price by hour:time.hh, symbol from trades
```

```
q)select last price by hour:time.hh, symbol from trades
hour symbol| price
-----------| -------
0    aapl  | 214.58
0    fb    | 136.14
0    msft  | 97.17
1    aapl  | 192.6
1    fb    | 144.045
..
```

**Figure 3.14:** Query 9 and its result

### 3.2.2 Aggregate queries on trades table

1. Calculate daily high, low, open and close price for a particular symbol in a certain month.

```
select high:max price, low:min price, open:first price, close:
last price by date from trades where date.month=2018.11m,symbol
=`appl
```

```
q)select high:max price, low:min price, open:first price, close:last price by dat
e from trades where date.month=2018.11m,symbol=`appl
date      | high low    open   close
----------| ----------------------
2018.11.01| 220  180    198.7  188.82
2018.11.02| 220  180    218.94 189.72
2018.11.03| 220  180    218.4  187.64
2018.11.04| 220  180    197.98 184.12
2018.11.05| 220  180    192.06 204.08
2018.11.06| 220  180    204.96 208.66
2018.11.07| 220  180    207.24 203.8
2018.11.08| 220  180    181.68 217.54
2018.11.09| 220  180    215.08 192.64
2018.11.10| 220  180    199.96 217.86
2018.11.11| 220  180    187.5  190.52
2018.11.12| 220  180    212.6  185.86
2018.11.13| 220  180    201.96 216.48
2018.11.14| 220  180    195.8  201.54
2018.11.15| 220  180    191.78 209.68
2018.11.16| 220  180    192.34 190.7
2018.11.17| 220  180    206.36 187.9
2018.11.18| 220  180.02 188.22 207.96
2018.11.19| 220  180    190.34 214.1
2018.11.20| 220  180    198.76 193.08
..
```

**Figure 3.15:** Query 10 and its result

2. Calculate the Volume Weighted Average price of all the symbols.

```
select vmap:volume wavg price by symbol from trades
```

```
q)select vmap:volume wavg price by symbol from trades
symbol| vmap
------| --------
appl  | 199.9619
fb    | 150.0143
msft  | 100.0069
```

**Figure 3.16:** Query 11 and its result

3. Daily traded values for all symbols in a certain month

```
select dtv:sum volume by date,symbol from trades where date.
month=2018.11m
```

```
q)select dtv:sum volume by date,symbol from trades where date.month=2018.11m
date       symbol| dtv
-----------------| --------
2018.11.01 appl  | 55332880
2018.11.01 fb    | 55799970
2018.11.01 msft  | 55415280
2018.11.02 appl  | 56539250
2018.11.02 fb    | 54751850
2018.11.02 msft  | 56217470
2018.11.03 appl  | 55634320
2018.11.03 fb    | 56096450
2018.11.03 msft  | 56219130
2018.11.04 appl  | 56086670
2018.11.04 fb    | 54937500
2018.11.04 msft  | 55974920
2018.11.05 appl  | 55213990
2018.11.05 fb    | 55134480
2018.11.05 msft  | 56939390
2018.11.06 appl  | 54781110
2018.11.06 fb    | 55116470
2018.11.06 msft  | 55054210
2018.11.07 appl  | 56112730
2018.11.07 fb    | 54919170
..
```

**Figure 3.17:** Query 12 and its result

4. Calculate the price range in hourly brackets

```
select range:max price - min price by date,symbol,hour:time.hh
from trades
```

```
q)select range:max price - min price by date,symbol,hour:time.hh from trades
date       symbol hour| range
----------------------| -----
2018.11.01 appl   0   | 39.9
2018.11.01 appl   1   | 39.9
2018.11.01 appl   2   | 39.72
2018.11.01 appl   3   | 39.92
2018.11.01 appl   4   | 39.88
2018.11.01 appl   5   | 40
2018.11.01 appl   6   | 39.8
2018.11.01 appl   7   | 39.96
2018.11.01 appl   8   | 39.76
2018.11.01 appl   9   | 39.9
2018.11.01 appl   10  | 39.8
2018.11.01 appl   11  | 39.94
2018.11.01 appl   12  | 39.98
2018.11.01 appl   13  | 39.58
2018.11.01 appl   14  | 39.8
2018.11.01 appl   15  | 39.8
2018.11.01 appl   16  | 39.38
2018.11.01 appl   17  | 39.96
2018.11.01 appl   18  | 39.82
2018.11.01 appl   19  | 39.86
..
```

**Figure 3.18:** Query 13 and its results

5. Calculate the hourly mean, variance and standard deviation of price for Apple

```
select mean:avg price, variance:var price, stdDev:dev price by
date, hour:time.hh from trades where symbol=`appl
```

```
q)select mean:avg price, variance:var price, stdDev:dev price by date, hour:time.
hh from trades where symbol=`appl
date       hour| mean     variance stdDev
---------------| -------------------------
2018.11.01 0   | 199.8171 142.0505 11.91849
2018.11.01 1   | 199.3792 135.5159 11.64113
2018.11.01 2   | 200.2049 130.6685 11.43103
2018.11.01 3   | 199.007  123.6663 11.12054
2018.11.01 4   | 200.1264 139.1972 11.79819
2018.11.01 5   | 200.1312 128.4472 11.33345
2018.11.01 6   | 200.2627 125.9159 11.22123
2018.11.01 7   | 199.8556 127.3738 11.286
2018.11.01 8   | 200.125  130.8988 11.4411
2018.11.01 9   | 200.1829 127.9234 11.31032
2018.11.01 10  | 200.2933 125.7191 11.21245
2018.11.01 11  | 200.3053 138.6056 11.77309
2018.11.01 12  | 200.1599 139.4567 11.80918
2018.11.01 13  | 200.1664 129.6772 11.38759
2018.11.01 14  | 199.7584 129.9923 11.40141
2018.11.01 15  | 200.5904 131.8661 11.4833
2018.11.01 16  | 199.8588 134.6351 11.60324
2018.11.01 17  | 199.4381 128.9132 11.35399
2018.11.01 18  | 199.8673 129.3266 11.37218
2018.11.01 19  | 199.9407 141.5454 11.89728
..
```

**Figure 3.19:** Query 14 and its result

## 3.3 Comparative Queries (q-SQL vs SQL)

This section compares syntax difference between q-SQL and SQL. It also shows how efficient and expressive q-SQL script is to query simple logic of manipulating data. We will show this syntactical difference over three different queries.

**Queries dependent on order**

Consider the below table stored in mysql and kdb+ database. We need to find the price change between consecutive rows.

| time | price |
|---|---|
| 07:00 | 0.9 |
| 08:30 | 1.5 |
| 09:59 | 1.9 |
| 10:00 | 2 |
| 12:00 | 9 |

**Table 3.1:** Example table

- **q-SQL**

```
update change:price-prev price from a
```

- **SQL**

```
select a.time, b.price-(select a.price from tab a where a.id =
    b.id + 1)as diff from tab b
```

The resulting table is as shown below.

| time | price | change |
|---|---|---|
| 07:00 | 0.9 | |
| 08:30 | 1.5 | 0.6 |
| 09:59 | 1.9 | 0.4 |
| 10:00 | 2 | 0.1 |
| 12:00 | 9 | 7 |

**Table 3.2:** Resulting table

**Select top N by category**
Given a table of stock trade prices at various times today, find the top two trade prices for each ticker. Consider the table as shown below.

| time | sym | price |
|---|---|---|
| 09:00 | a | 80 |
| 09:03 | b | 10 |
| 09:05 | c | 30 |
| 09:10 | a | 85 |
| 09:20 | a | 75 |
| 09:30 | b | 13 |
| 09:40 | b | 14 |

**Table 3.3:** Stock trade table

- **q-SQL**

```
select 2 sublist desc price by sym from trade
```

- **SQL**

```
SELECT sym, price FROM (
          SELECT
              ROW_NUMBER() OVER ( PARTITION BY sym ORDER BY
    price DESC ) AS 'RowNumber',
              sym, price FROM trade
      ) dt WHERE RowNumber <= 2
```

The resulting table is as shown below.

| sym | price |
|---|---|
| a | 85 |
| a | 80 |
| b | 14 |
| b | 13 |
| c | 30 |

**Table 3.4:** Resulting table

# Conclusion

This report presented an introduction into time series data and time series databases, with a focus on kdb+. It started by discussing time series data, where it showed how widely used it was, from the evolution of commodities, to data centre measurements or web event streams. Its popularity was explained by its many benefits offered through analysis or forecasting. However, due to the characteristics of this type of data, a suitable database technology had to be implemented: time series databases. The report offered then an introduction into these systems, explaining their features, mainly scalability and usability. TSDBs were shown to have had the highest increase in popularity out of all types of databases, with InfluxDB and Kdb+ topping the trend chart.

Furthermore, the report described and analysed the kdb+ database. It showed the database's main features, such as being column oriented and in-memory. It also described its architecture, together with its own programming language, q, and q-sql like CRUD operations. An important section was focused on the performance and benchmark of kdb+ with other databases, the main reasons that explain its popularity, especially in the financial sector. It offers very high performance due to its architecture, being suitable for both real time and historical data.

The last chapter of the report described the implementation of an application in a financial context, related to stock trading. It therefore demonstrated the usability of kdb+ in this sector, through various queries that could be used in the real-life situation. They were also useful to observe the simplicity of q and q-sql over general sql, as well as its higher power and ease of use in certain contexts.

# Bibliography


[1] Ted Dunning and Ellen Friedman. "Time series databases". In: *New Ways to Store and Access data* (2015).

[2] *Performance benchmark for q*. `http://kparc.com/q4/readme.txt`. Accessed: 2018-12-11.

[3] *Q language queries*. `https://www.tutorialspoint.com/kdbplus/q_language_queries.htm`. Accessed: 2018-12-11.

[4] Kx Systems. *The 21st Century Time-series Database*. Tech. rep. Kx Systems, 2010.

[5] TimeStored.com. *Who use's kdb+? What is kdb+ used for?* `http://www.timestored.com/kdb-guides/who-uses-kdb`. Accessed: 2016.