



[KDB+]

High-Performance Database

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

Kdb+ is a high-performance column-oriented database from **Kx Systems Inc.** kdb+ is designed to capture, analyze, compare, and store data — all at high speeds and on high volumes of data.

The tutorial starts off with a basic introduction of Kdb+ followed by its architecture, installation, and a basic-to-advanced coverage of **q programming language**.

Audience

This reference has been prepared for beginners to help them understand the KDB+ database and write smart queries in **q** languages for KDB+.

Prerequisites

As we are going to start from scratch, you can set off without any preparation. However, it would definitely help if you have a working knowledge of any database or a programming language.

Copyright & Disclaimer

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute, or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness, or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
Table of Contents	ii
PART 1: KDB+ ARCHITECTURE	1
1. KDB+ – Overview	2
Background.....	2
Why and Where to Use KDB+.....	2
Getting Started	2
2. KDB+ – Architecture	5
Kdb+/ tick Architecture	5
Components of Kdb+ Tick Architecture.....	7
PART 2: Q PROGRAMMING LANGUAGE	9
3. Q Programming Language	10
Starting the “q” Environment.....	10
Data Types	10
Atom and List Formation	11
4. Type Casting.....	12
Casting Integers to Floats	12
Casting Strings to Symbols.....	12
Casting Strings to Non-Symbols.....	13
5. Temporal Data	14
Date	14
Times	15
Datetimes	15
6. Lists.....	17
Types of List.....	17
7. Indexing	19
Index Notation	19
Indexed Assignment	19
Joining Lists.....	20
Nesting.....	20
Elided Indices.....	21
8. Dictionaries.....	23
Lookup	24
Operations on Dictionaries.....	24
Column Dictionaries	25
Flipping a Dictionary.....	26

9. Table	27
Creating Tables	27
Getting Table Information	28
Primary Keys and Keyed Tables	28
Manipulating Tables	30
10. Verb & Adverbs	35
Each	35
Each-Left and Each-Right.....	35
11. Joins	38
Simple Join.....	38
Asof Join (aj)	38
Left Join (lj)	39
Union Join (uj).....	39
12. Functions	41
Types of Functions.....	41
Frequently Used Functions.....	41
System Commands	52
13. Built-in Functions	56
String Functions	56
Mathematical Functions.....	57
Aggregate Functions.....	58
Uniform Functions	59
Miscellaneous Functions	60
14. Queries	62
Basics Queries.....	62
Queries with Constraints	63
Queries with Aggregations	65
15. Inter-Process Communication	67
Initialize Server	67
Communication Handle	67
Synchronous and Asynchronous Messages.....	68
16. Message Handler (.z Library)	69
Predefined Message Handlers.....	69
PART 3: ADVANCED TOPICS	72
17. Attributes	73
Types of Attributes	73
18. Functional Queries	77
Functional select.....	77
Functional Exec.....	79
Functional Update	79
Functional delete.....	79
19. Table Arithmetic	81

20. Tables on Disk	84
Flat file	84
Splayed Tables	85
Partitioned Tables.....	85
21. Maintenance Functions.....	87
.Q.en	87
.Q.dpft	87
.Q.chk.....	88
22. KDB+ – FAQs	89
The Basics	89
Data Types and Structure	90
List	91
Indexing & Mixed type	92
Dictionaries.....	93
Functions	94
Table	95

PART 1: KDB+ ARCHITECTURE

1. KDB+ – OVERVIEW

This is a complete guide to **kdb+** from kx systems, aimed primarily at those learning independently. **kdb+**, introduced in 2003, is the new generation of the **kdb** database which is designed to capture, analyze, compare, and store data.

A **kdb+** system contains the following two components:

- **KDB+** – the database (k database plus)
- **Q** – the programming language for working with **kdb+**

Both **kdb+** and **q** are written in **k programming language** (same as **q** but less readable).

Background

Kdb+/q originated as an obscure academic language but over the years, it has gradually improved its user friendliness.

- **APL** (1964, A Programming Language)
- **A+** (1988, modified APL by Arthur Whitney)
- **K** (1993, crisp version of A+, developed by A. Whitney)
- **Kdb** (1998, in-memory column-based db)
- **Kdb+/q** (2003, q language – more readable version of k)

Why and Where to Use KDB+

Why? – If you need a single solution for real-time data with analytics, then you should consider **kdb+**. **Kdb+** stores database as ordinary native files, so it does not have any special needs regarding hardware and storage architecture. It is worth pointing out that the database is just a set of files, so your administrative work won't be difficult.

Where to use KDB+? – It's easy to count which investment banks are NOT using **kdb+** as most of them are using currently or planning to switch from conventional databases to **kdb+**. As the volume of data is increasing day by day, we need a system that can handle huge volumes of data. **KDB+** fulfills this requirement. **KDB+** not only stores an enormous amount of data but also analyzes it in real time.

Getting Started

With this much of background, let us now set forth and learn how to set up an environment for **KDB+**. We will start with how to download and install **KDB+**.

Downloading & Installing KDB+

You can get the free 32-bit version of KDB+, with all the functionality of the 64-bit version from

<http://kx.com/software-download.php>

Agree to the license agreement, select the operating system (available for all major operating system). For Windows operating system, the latest version is 3.2. Download the latest version. Once you unzip it, you will get the folder name "**windows**" and inside the windows folder, you will get another folder "**q**". Copy the entire **q** folder onto your c:/ drive.

Open the Run terminal, type the location where you store the **q** folder; it will be like "c:/q/w32/q.exe". Once you hit Enter, you will get a new console as follows:

```

C:\q\w32\q.exe
KDB+ 3.2 2015.03.05 Copyright (C) 1993-2015 Kx Systems
w32/ 2<core 1893MB myaccount-raj rajesh-pc 192.168.0.102 NONEPIRE

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/wiki/Tutorials
To exit, type \
To remove this startup msg, edit q.q
q>_

```

On the first line, you can see the version number which is 3.2 and the release date as 2015.03.05

Directory Layout

The trial/free version is generally installed in directories,

For linux/Mac:

~/q	/ main q directory (under the user's home)
~/q/l32	/ location of linux 32-bit executable
~/q/m32	/ Location of mac 32-bit executable

For Windows:








c:/q	/ Main q directory
------	--------------------

c:/q/w32/

/ Location of windows 32-bit executable

Example Files:

Once you download kdb+, the directory structure in the Windows platform would appear as follows:

 w32	File folder	
 q	q KDB source	19 KB
 q	Q File	1 KB
 README	Text Document	4 KB
 s	q KDB source	6 KB
 sp	Q File	1 KB
 trade	Q File	1 KB

In the above directory structure, **trade.q** and **sp.q** are the example files which we can use as a reference point.

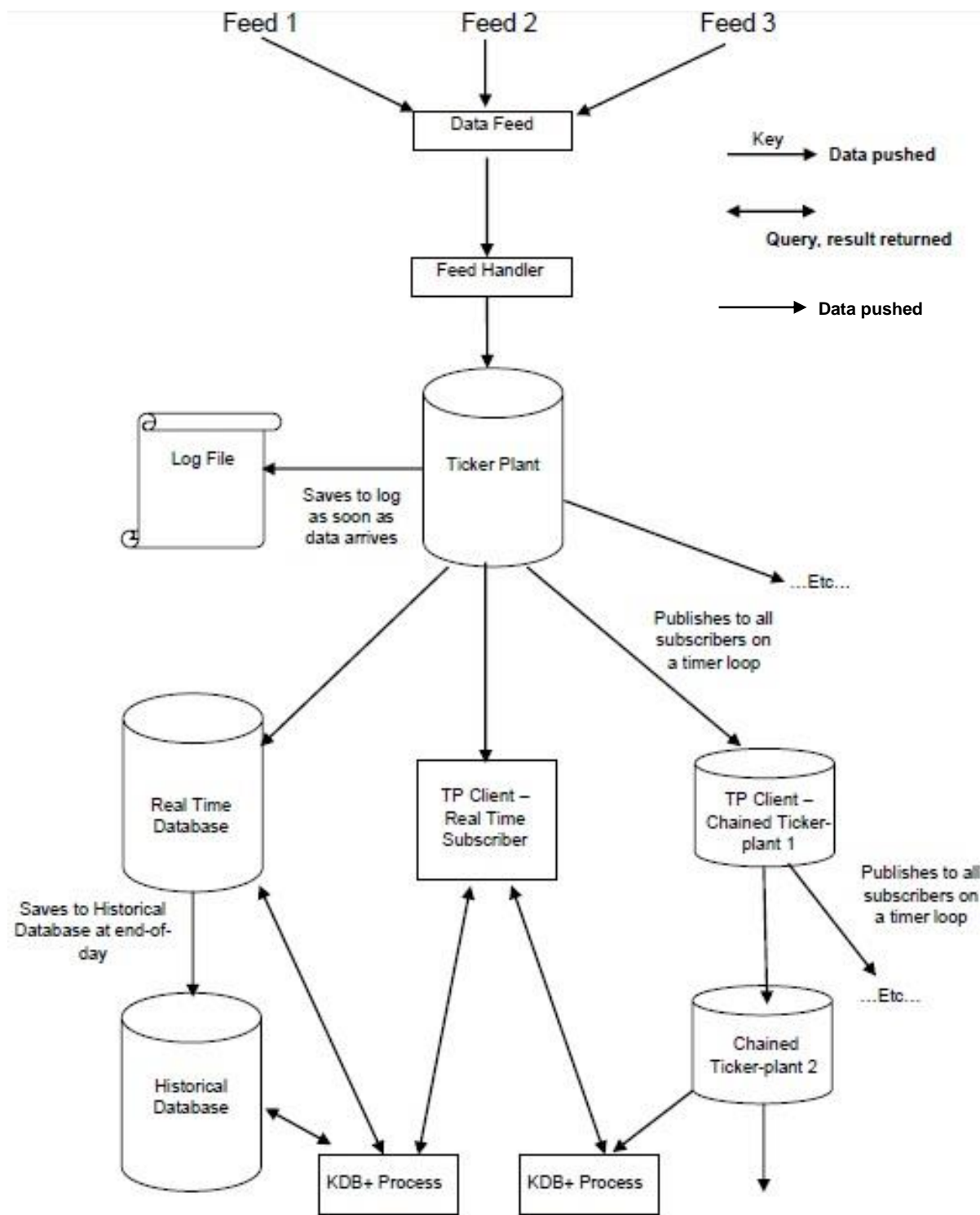
2. KDB+ – ARCHITECTURE

Kdb+ is a high-performance, high-volume database designed from the outset to handle tremendous volumes of data. It is fully 64-bit, and has built-in multi-core processing and multi-threading. The same architecture is used for real-time and historical data. The database incorporates its own powerful query language, **q**, so analytics can be run directly on the data.

kdb+tick is an architecture which allows the capture, processing, and querying of real-time and historical data.

Kdb+/ tick Architecture

The following illustration provides a generalized outline of a typical Kdb+/tick architecture, followed by a brief explanation of the various components and the through-flow of data.



- The **Data Feeds** are a time series data that are mostly provided by the data feed providers like Reuters, Bloomberg or directly from exchanges.

- To get the relevant data, the data from the data feed is parsed by the **feed handler**.
- Once the data is parsed by the feed handler, it goes to the **ticker-plant**.
- To recover data from any failure, the ticker-plant first updates/stores the new data to the log file and then updates its own tables.
- After updating the internal tables and the log files, the on-time loop data is continuously sent/published to the real-time database and all the chained subscribers who requested for data.
- At the end of a business day, the log file is deleted, a new one created and the real-time database is saved onto the historical database. Once all the data is saved onto the historical database, the real-time database purges its tables.

Components of Kdb+ Tick Architecture

Data Feeds

Data Feeds can be any market or other time series data. Consider data feeds as the raw input to the feed-handler. Feeds can be directly from the exchange (live-streaming data), from the news/data providers like Thomson-Reuters, Bloomberg, or any other external agencies.

Feed Handler

A feed handler converts the data stream into a format suitable for writing to kdb+. It is connected to the data feed and it retrieves and converts the data from the feed-specific format into a Kdb+ message which is published to the ticker-plant process. Generally a feed handler is used to perform the following operations:

- Capture data according to a set of rules.
- Translate (/enrich) that data from one format to another.
- Catch the most recent values.

Ticker Plant

Ticker Plant is the most important component of KDB+ architecture. It is the ticker plant with which the real-time database or directly subscribers (clients) are connected to access the financial data. It operates in **publish and subscribe** mechanism. Once you obtain a subscription (license), a tick (routinely) publication from the publisher (ticker plant) is defined. It performs the following operations:

- Receives the data from the feed handler.
- Immediately after the ticker plant receives the data, it stores a copy as a log file and updates it once the ticker plant gets any update so that in case of any failure, we should not have any data loss.
- The clients (real-time subscriber) can directly subscribe to the ticker-plant.

- At the end of each business day, i.e., once the real-time database receives the last message, it stores all of today's data onto the historical database and pushes the same to all the subscribers who have subscribed for today's data. Then it resets all its tables. The log file is also deleted once the data is stored in the historical database or other directly linked subscriber to real time database (rtdb).
- As a result, the ticker-plant, the real-time database, and the historical database are operational on a 24/7 basis.

Since the ticker-plant is a Kdb+ application, its tables can be queried using **q** like any other Kdb+ database. All ticker-plant clients should only have access to the database as subscribers.

Real-Time Database

A real-time database (rdb) stores today's data. It is directly connected to the ticker plant. Typically it would be stored in memory during market hours (a day) and written out to the historical database (hdb) at the end of day. As the data (rdb data) is stored in memory, processing is extremely fast.

As kdb+ recommends to have a RAM size that is four or more times the expected size of data per day, the query that runs on rdb is very fast and provides superior performance. Since a real-time database contains only today's data, the date column (parameter) is not required.

For example, we can have rdb queries like,

```
select from trade where sym = `ibm
OR
select from trade where sym = `ibm, price > 100
```

Historical Database

If we have to calculate the estimates of a company, we need to have its historical data available. A historical database (hdb) holds data of transactions done in the past. Each new day's record would be added to the hdb at the end of day. Large tables in the hdb are either stored splayed (each column is stored in its own file) or they are stored partitioned by temporal data. Also some very large databases may be further partitioned using **par.txt** (file).

These storage strategies (splayed, partitioned, etc.) are efficient while searching or accessing the data from a large table.

A historical database can also be used for internal and external reporting purposes, i.e., for analytics. For example, suppose we want to get the company trades of IBM for a particular day from the trade (or any) table name, we need to write a query as follows:

```
thisday: 2014.10.12

select from trade where date= thisday, sym=`ibm
```

Note – We will write all such queries once we get some overview of the **q** language.

PART 2: Q PROGRAMMING LANGUAGE

3. Q PROGRAMMING LANGUAGE

Kdb+ comes with its built-in programming language that is known as **q**. It incorporates a superset of standard SQL which is extended for time-series analysis and offers many advantages over the standard version. Anyone familiar with SQL can learn **q** in a matter of days and be able to quickly write her own ad-hoc queries.

Starting the “q” Environment

To start using kdb+, you need to start the **q** session. There are three ways to start a **q** session:

- Simply type “c:/q/w32/q.exe” on your run terminal.
- Start the MS-DOS command terminal and type **q**.
- Copy the **q.exe** file onto “C:\Windows\System32” and on the run terminal, just type “q”.

Here we are assuming that you are working on a Windows platform.

Data Types

The following table provides a list of supported data types:

Name	Example	Char	Type	Size
boolean	1b	b	1	1
byte	0xff	x	4	1
short	23h	h	5	2
int	23i	i	6	4
long	23j	j	7	8
real	2.3e	e	8	4
float	2.3f	f	9	8
char	“a”	c	10	1
varchar	`ab	s	11	*
month	2003.03m	m	13	4
date	2015.03.17T18:01:40.134	z	15	8
minute	08:31	u	17	4
second	08:31:53	v	18	4
time	18:03:18.521	t	19	4
enum	`u\$b, where u:`a`b	*	20	4

Atom and List Formation

Atoms are single entities, e.g., a single number, a character or a symbol. In the above table (of different data types), all supported data types are atoms. A list is a sequence of atoms or other types including lists.

Passing an atom of any type to the monadic (i.e. single argument function) type function will return a negative value, i.e., **-n**, whereas passing a simple list of those atoms to the type function will return a positive value **n**.

Example 1 – Atom and List Formation

```

/ Note that the comments begin with a slash “ / ” and cause the parser
/ to ignore everything up to the end of the line.

x: `mohan                / `mohan is a symbol, assigned to a variable x
type x                   / let's check the type of x
-11h                     / -ve sign, because it's single element.
y: (`abc;`bca;`cab)      / list of three symbols, y is the variable name.
type y
11h                      / +ve sign, as it contain list of atoms (symbol).
y1: (`abc`bca`cab)      / another way of writing y, please note NO semicolon
y2: (`$"symbols may have interior blanks") / string to symbol conversion
y[0]                     / return `abc
y 0                      / same as y[0], also returns `abc
y 0 2                    / returns `abc`cab, same as does y[0 2]
z: (`abc; 10 20 30; (`a`b); 9.9 8.8 7.7) / List of different types,
z 2 0                    / returns (`a`b; `abc),
z[2;0]                  / return `a. first element of z[2]
x: "Hello World!"       / list of character, a string
x 4 0                   / returns "oH" i.e. 4th and 0th(first) element

```


4. TYPE CASTING

It is often required to change the data type of some data from one type to another. The standard casting function is the "\$" **dyadic operator**.

Three approaches are used to cast from one type to another (except for string):

1. Specify desired data type by its symbol name
2. Specify desired data type by its character
3. Specify desired data type by its short value.

Casting Integers to Floats

In the following example of casting integers to floats, all the three different ways of casting are equivalent:

```
q)a:9 18 27
q)$[`float;a] / Specify desired data type by its symbol name, 1st way
9 18 27f
q)$["f";a] / Specify desired data type by its character, 2nd way
9 18 27f
q)$[9h;a] / Specify desired data type by its short value, 3rd way
9 18 27f
```

Check if all the three operations are equivalent,

```
q)($[`float;a]~$["f";a]) and ($[`float;a] ~ $[9h;a])
1b
```

Casting Strings to Symbols

Casting string to symbols and vice versa works a bit differently. Let's check it with an example:

```
q)b: ("Hello";"World";"HelloWorld") / define a list of strings
q)b
"Hello"
"World"
"HelloWorld"
```

```

q)c: ` $b          / this is how to cast strings to symbols
q)c                / Now c is a list of symbols
`Hello`World`HelloWorld

```

Attempting to cast strings to symbols using the keyed words `symbol or 11h will fail with the type error:

```

q)b
"Hello"
"World"
"HelloWorld"
q)`symbol$b
'type
q)11h$b
'type

```

Casting Strings to Non-Symbols

Casting strings to a data type other than symbol is accomplished as follows:

```

q)b:900           / b contain single atomic integer
q)c:string b     / convert this integer atom to string "900"
q)c
"900"
q)`int $ c       / converting string to integer will return the / ASCII
                  equivalent of the character "9", "0" and / "0" to produce
                  the list of integer 57, 48 and / 48.
57 48 48i
q)6h $ c         / Same as above
57 48 48i
q)"i" $ c        / Same a above
57 48 48i
q)"I" $ c
900i

```

So to cast an entire string (the list of characters) to a single atom of data type **x** requires us to specify the upper case letter representing data type **x** as the first argument to the **\$** operator. If you specify the data type of **x** in any other way, it result in the cast being applied to each character of the string.

5. TEMPORAL DATA

The **q** language has many different ways of representing and manipulating temporal data such as times and dates.

Date

A date in kdb+ is internally stored as the integer number of days since our reference date is 01Jan2000. A date after this date is internally stored as a positive number and a date before that is referenced as a negative number.

By default, a date is written in the format "YYYY.MM.DD"

```
q)x:2015.01.22 / This is how we write 22nd Jan 2015
q)`int$x      / Number of days since 2000.01.01
5500i
q)`year$x    / Extracting year from the date
2015i
q)x.year     / Another way of extracting year
2015i
q)`mm$x     / Extracting month from the date
1i
q)x.mm      / Another way of extracting month
1i
q)`dd$x    / Extracting day from the date
22i
q)x.dd     / Another way of extracting day
22i
```

Arithmetic and logical operations can be performed directly on dates.

```
q)x+1      / Add one day
2015.01.23
q)x-7     / Subtract 7 days
2015.01.15
```

The 1st of January 2000 fell on a Saturday. Therefore any Saturday throughout the history or in the future when divided by 7, would yield a remainder of 0, Sunday gives 1, Monday yield 2.

Day	mod 7
Saturday	0
Sunday	1
Monday	2
Tuesday	3
Wednesday	4
Thursday	5
Friday	6

Times

A time is internally stored as the integer number of milliseconds since the stroke of midnight. A time is written in the format HH:MM:SS.MSS

```

q)tt1: 03:30:00.000 / tt1 store the time 03:30 AM
q)tt1
03:30:00.000
q)`int$tt1 / Number of milliseconds in 3.5 hours
12600000i
q)`hh$tt1 / Extract the hour component from time
3i
q)tt1.hh
3i
q)`mm$tt1 / Extract the minute component from time
30i
q)tt1.mm
30i
q)`ss$tt1 / Extract the second component from time
0i
q)tt1.ss
0i

```

As in case of dates, arithmetic can be performed directly on times.

Datetimes

A datetime is the combination of a date and a time, separated by 'T' as in the ISO standard format. A datetime value stores the fractional day count from midnight Jan 1, 2000.

```
q)dt:2012.12.20T04:54:59:000          / 04:54.59 AM on 20thDec2012
q)type dt
-15h
q)dt
2012.12.20T04:54:59.000
q)`float$dt
4737.205
```

The underlying fractional day count can be obtained by casting to float.

End of ebook preview
If you liked what you saw...
Buy it from our store @ <https://store.tutorialspoint.com>