DATASTAX

Advanced Data Modeling

How to Create Efficient Data Models for Massively Scalable Applications

Online Hands-on Workshop

Presenters

Artem Chebotko, Ph.D.

Solutions Architect at DataStax

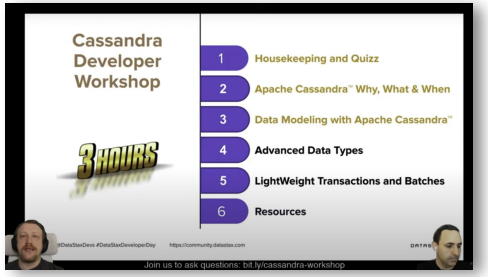
- Author of data modeling methodology for Apache Cassandra™
- 15+ years in database research and development
- 15+ years in teaching for academia and industry
- 50+ refereed publications in international venues

Aleks Volochnev

Developer Advocate at DataStax

- Apache Cassandra™ expert and workshop guru
- Experienced developer and educator
- Certified cloud architect

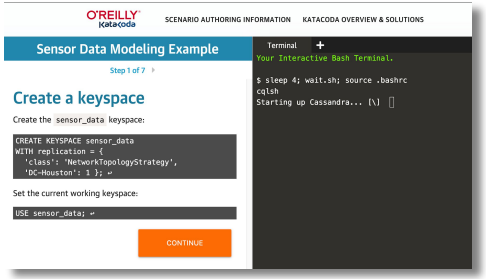
Course: youtube.com/DataStaxDevs



YouTube

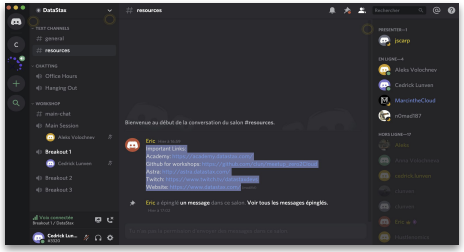
Twitch

Hands-On: katacoda.com/datastax



KataCoda

Questions: bit.ly/cassandra-workshop

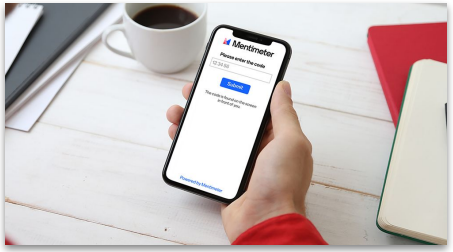


Discord

YouTube

Available on the iPhone App Store

Quizz: menti.com



Mentimeter

GET IT ON Google play

DATASTAX

Few Questions to know you better

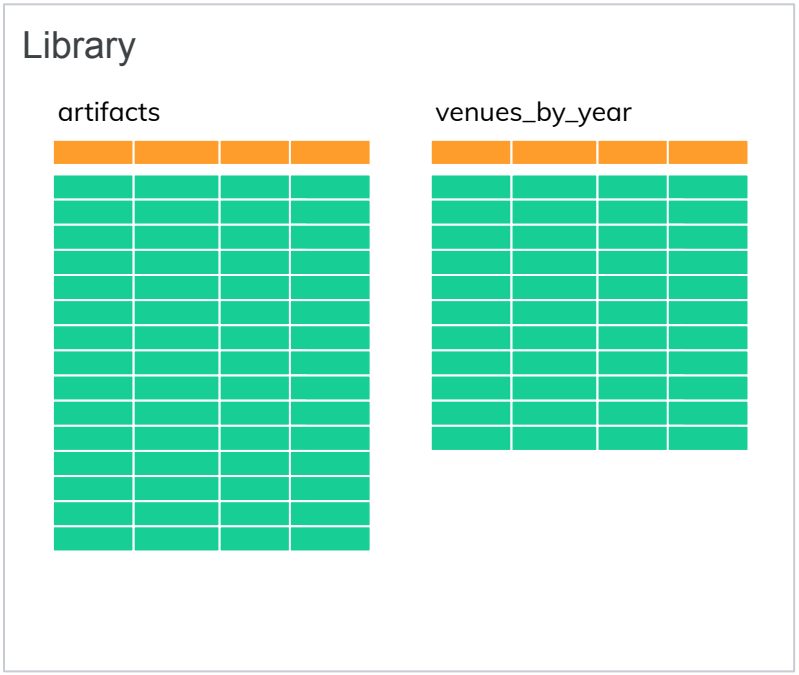
How Cassandra Organizes Data

How Cassandra Organizes Data

Library

```
CREATE KEYSPACE library
WITH replication = {
'Class': 'NetworkTopologyStrategy',
  'DC-West': '3',
  'DC-East': '5'
};
```

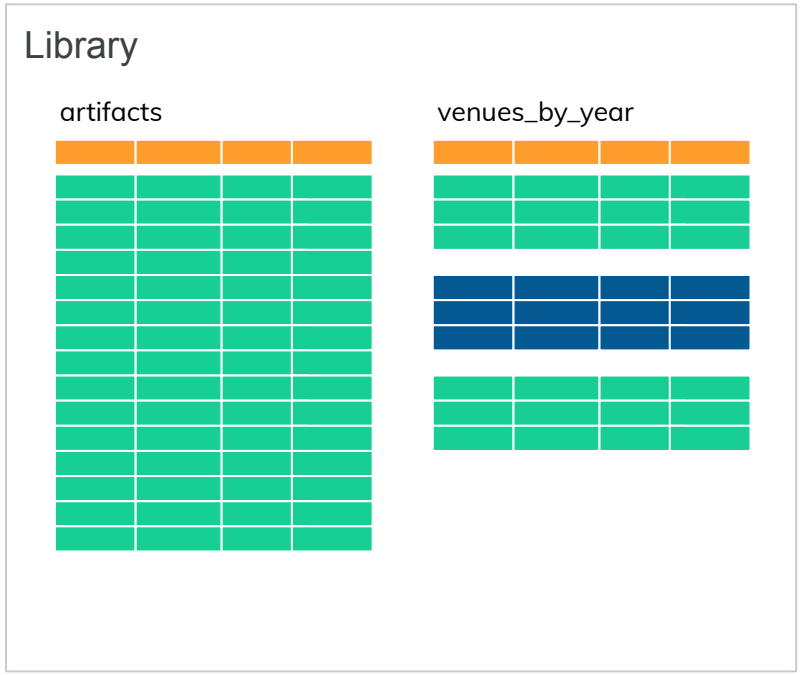
How Cassandra Organizes Data




```
CREATE KEYSPACE library
WITH replication = {
'Class': 'NetworkTopologyStrategy',
  'DC-West': '3',
  'DC-East': '5'
};
```

```
CREATE TABLE library.venues_by_year(
    year INT,
    name TEXT,
    country TEXT,
    homepage TEXT,
    PRIMARY KEY ((year),name)
);
```

How Cassandra Organizes Data

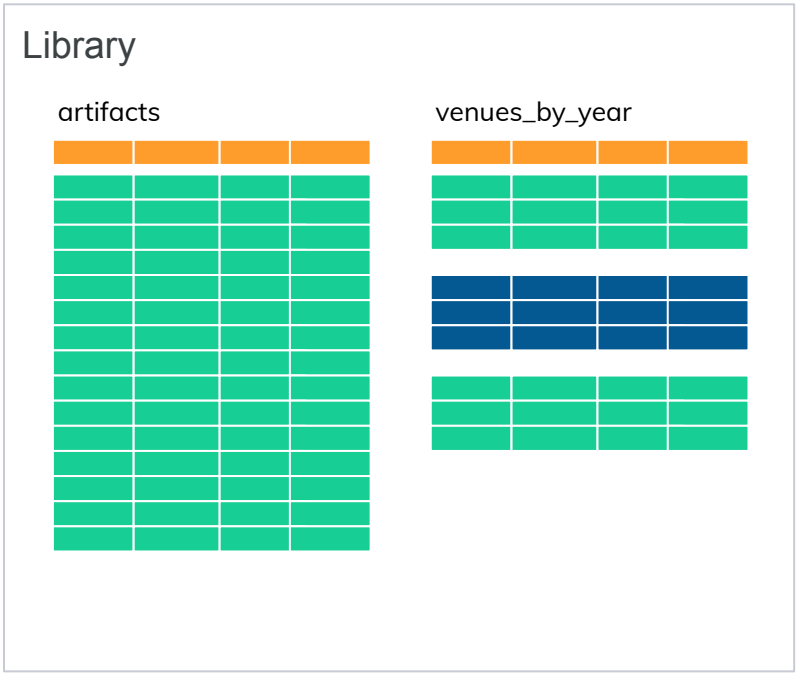


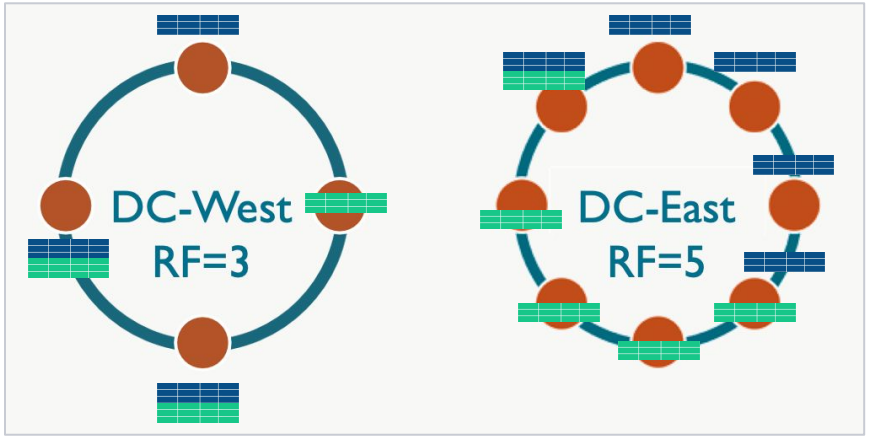
YEAR	NAME	COUNTRY	...
2019	Cassandra Summit	USA	...
2019	DataStax Accelerate	USA	...

YEAR	NAME	COUNTRY	...
2015	A	...	...
2015	B	...	...

```
CREATE TABLE library.venues_by_year(
    ...
    PRIMARY KEY ((year),name)
);
```

How Cassandra Organizes Data






```
CREATE TABLE library.venues_by_year(
    ...
    PRIMARY KEY ((year),name)
);
```

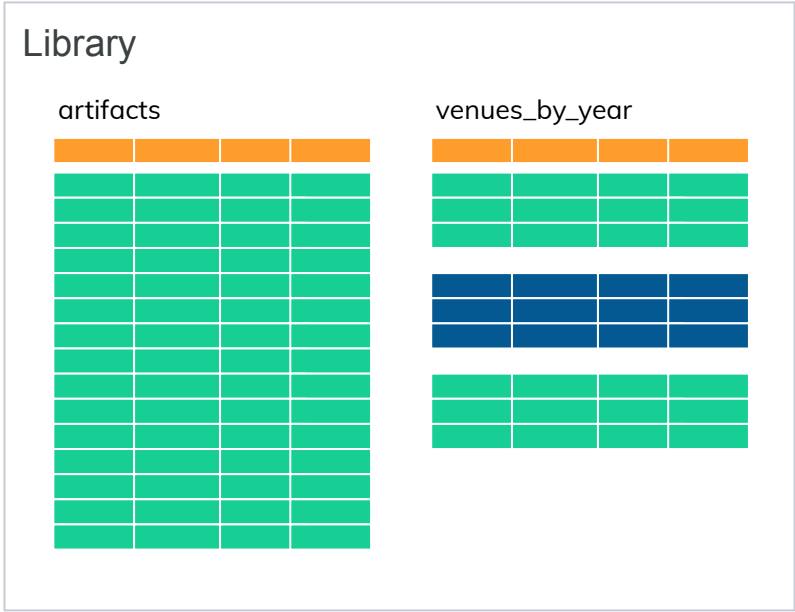
Cassandra Query Language

Data Definition

- CREATE KEYSPACE, CREATE TABLE
- CREATE INDEX, CREATE CUSTOM INDEX
- CREATE MATERIALIZED VIEW

Data Manipulation

- SELECT
- INSERT, UPDATE, DELETE

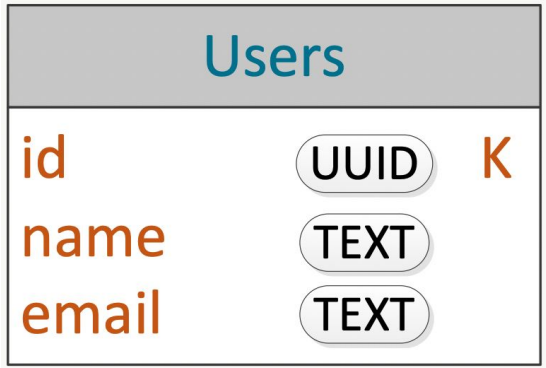


CQL CREATE TABLE

```
CREATE TABLE name
    Column type [STATIC],
    Column type [STATIC]

    ...

    PRIMARY KEY ((column, ...), column, ...)
) WITH CLUSTERING ORDER BY(
    Clustering_column [ASC|DESC], ...
);
```

Single-Row Partitions

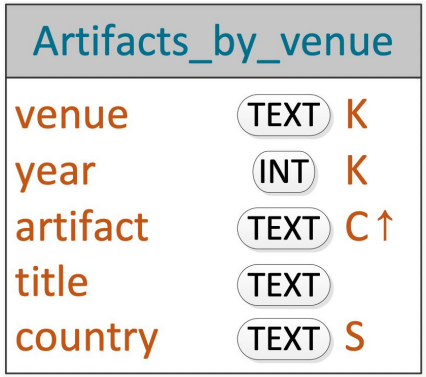
```
CREATE TABLE users (
    id UUID,
    name TEXT,
    email TEXT,
    PRIMARY KEY (id)
);
```



ID	EMAIL	NAME
67657da3-4443-46ab-b60a-510a658fc7bb	achebotko@datastax.com	Artem
98757da3-4443-46ab-b60a-510a658fc7aa	aleksandr.volochnev@datastax.com	Aleks

Multi-Row Partitions

```
CREATE TABLE artifacts_by_venue (
    venue TEXT, year INT,
    artifact TEXT,
    title TEXT,
    country TEXT STATIC,
    PRIMARY KEY ((venue, year), artifact)
);
```



VENUE	YEAR	ARTIFACT	TITLE	COUNTRY
DataStax Accelerate	2019	A...	Linear Scalability ...	USA
		...	...	
		Z...	Building Cloud	
Data Modeling Zone	2019	A...	New Approach to ...	USA
		...	...	
		Z...	...	

CQL SELECT

```
SELECT      selectors
FROM        table_name
WHERE       primary_key_conditions
            AND index_conditions
GROUP BY    primary_key_columns
ORDER BY    clustering_key_columns ( ASC | DESC )
LIMIT       N
ALLOW FILTERING ;
```

columns, aggregates, functions

one table per query

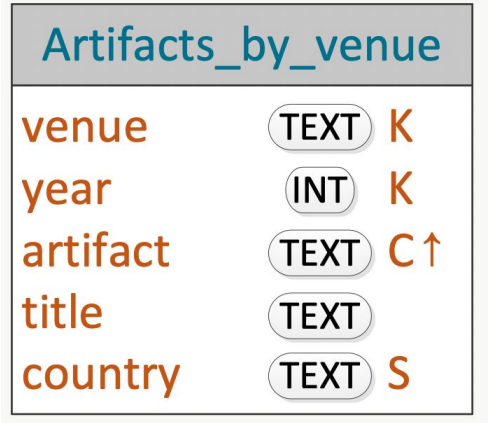
restricted to primary key columns

danger

DATASTAX

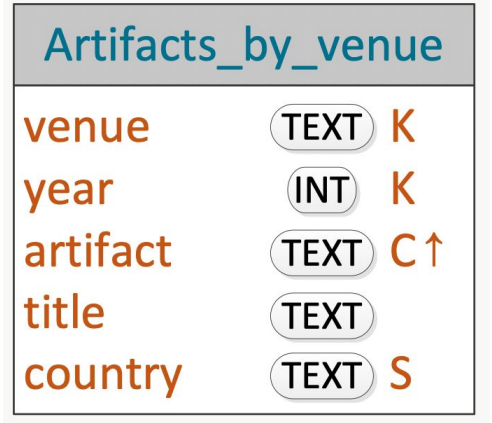
Sample CQL Queries

```
SELECT *
FROM artifacts_by_venue

WHERE venue = ? AND year = ?;

WHERE venue = ? AND year = ? AND artifact = ?;

WHERE venue = ? AND year = ? AND
      artifact > ? AND artifact < ?
ORDER BY artifact DESC;
```



Invalid CQL Queries

```
SELECT *
FROM artifacts_by_venue

WHERE venue = ?;

WHERE venue = ? AND artifact = ?;

WHERE artifact > ? AND artifact < ?

WHERE venue = ? AND year = ? AND title = ?;

WHERE country = ?;
```



Important Implications for Data Modeling

Data

- Primary keys define data uniqueness
- Partition keys define data distribution
- Partition keys affect partition sizes
- Clustering keys define row ordering

Query

- Primary keys define how data is retrieved
- Partition keys allow equality predicates
- Clustering keys allow inequality predicates and ordering
- Only one table per query, no joins

Data Modeling

- Collection and analysis of data requirements
- Identification of participating entities and relationships
- Identification of data access patterns
- A particular way of organizing and structuring data
- Design and specification of a database schema
- Schema optimization and data indexing techniques

Data Quality: completeness consistency accuracy
Data Access: queryability efficiency scalability

Key Data Modeling Steps

Understand the data

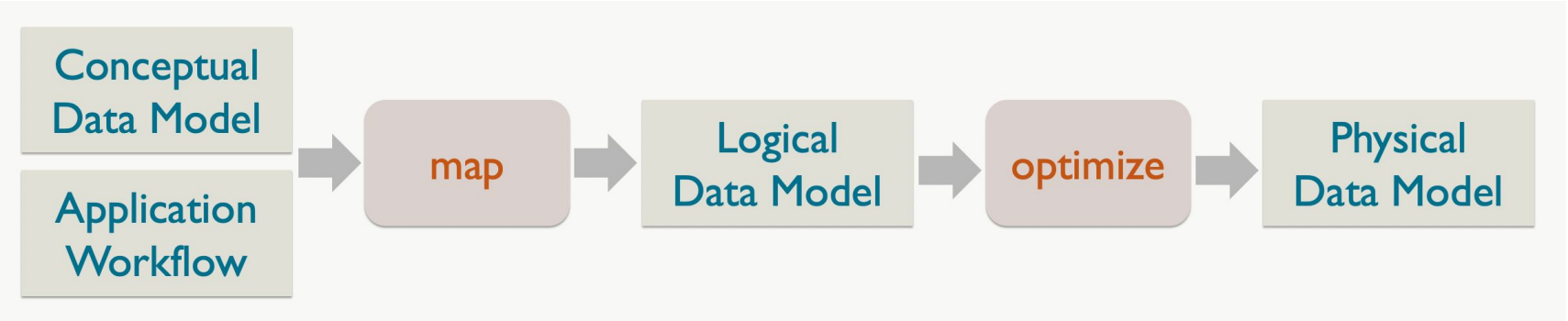
Identify access patterns

Apply a query-first approach

Optimize and implement

The Data Modeling Framework

Defines Models and Transitions

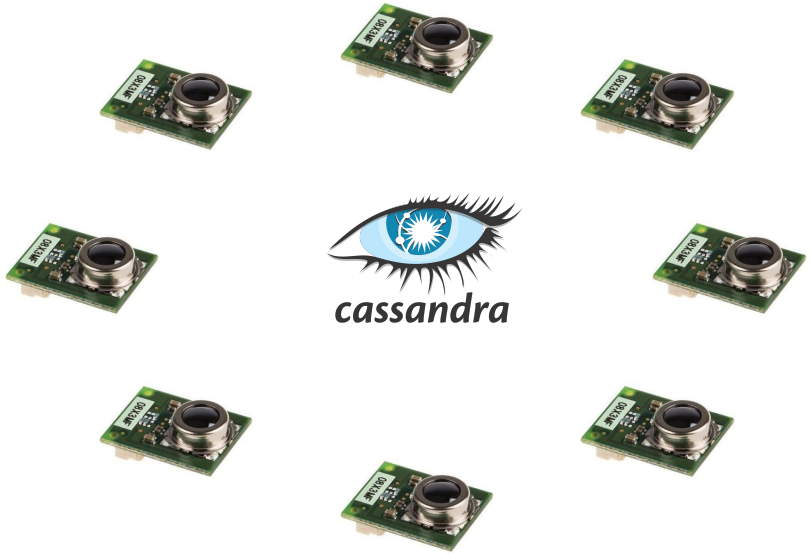


IoT Data

Sensor Network

Learn by Example! In this one, we learn how to create an efficient data model for global temperature monitoring sensor networks.

We cover a conceptual data model, application workflow, logical data model, physical data model, final CQL schema and query design.

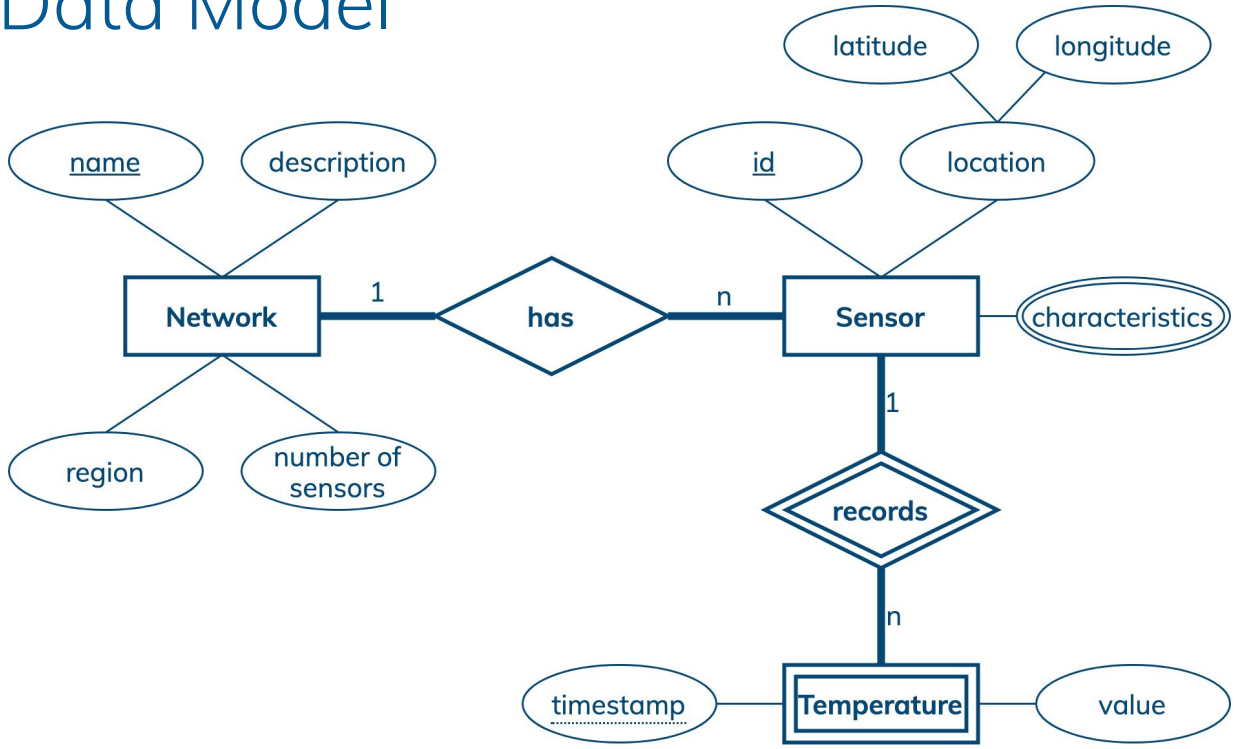


IoT: Conceptual Data Model

The conceptual data model for sensor data features sensor networks, sensors, and temperature measurements.

Each network has a unique name, description, region, and number of sensors. A sensor is described by a unique id, location (latitude and longitude), multiple sensor characteristics. A temperature measurement has a timestamp and value, and is uniquely identified by a sensor id and a measurement timestamp.

While a network can have many sensors, each sensor can only belong to one network.



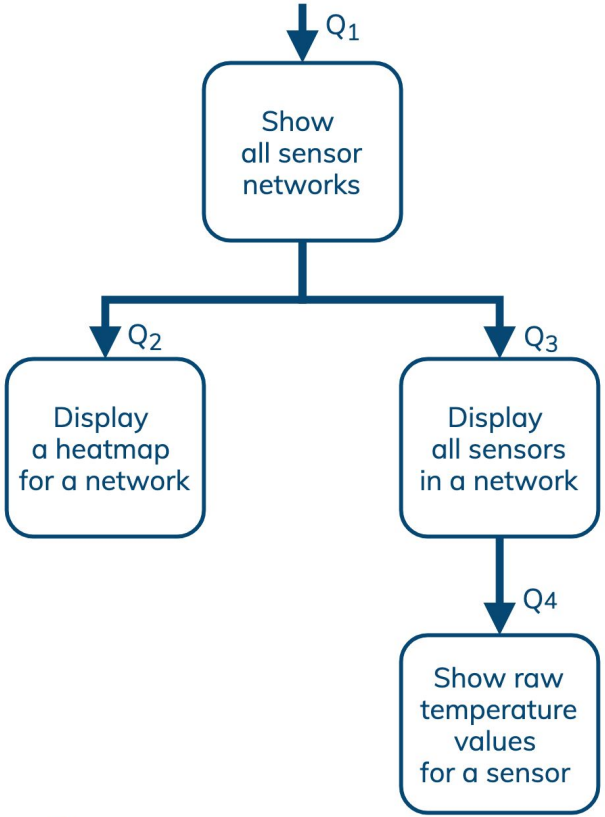
Entity-Relationship Diagram

IoT: Application Workflow

An application workflow is designed with the goal of understanding data access patterns. Its visual representation consists of application tasks, dependencies among tasks, and data access patterns. Ideally, each data access pattern should specify what attributes to search for, search on, order by, or do aggregation on.

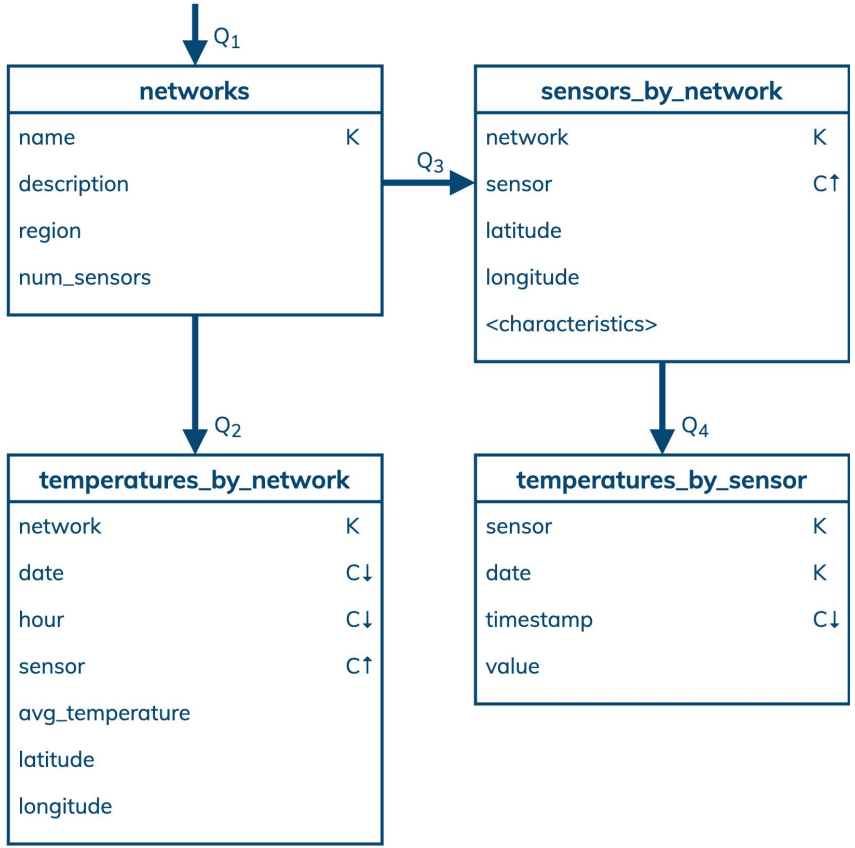
Data access patterns

Q_1: Find information about all networks; order by name (asc)
Q_2: Find hourly average temperatures for every sensor in a specified network for a given date range; order by date (desc) and hour (desc)
Q_3: Find information about all sensors in a specified network
Q_4: Find raw measurements for a particular sensor on a specified date; order by timestamp (desc)



IoT: Logical Data Model

A logical data model results from a conceptual data model by *organizing data into Cassandra-specific data structures based on data access patterns* identified by an application workflow. Logical data models can be conveniently captured and visualized using Chebotko Diagrams that can feature tables, materialized views, indexes and so forth.

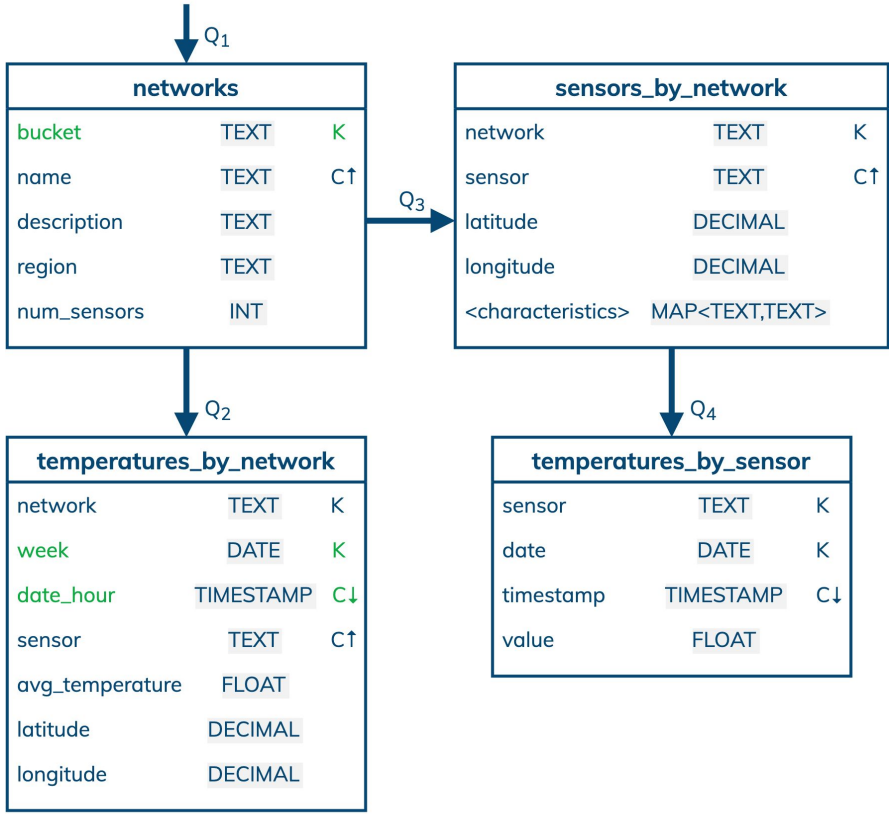


Chebotko Diagram

IoT: Physical Data Model

A physical data model is directly derived from a logical data model by analyzing and optimizing for performance. The most common type of analysis is identifying potentially large partitions.

Some common optimization techniques include splitting and merging partitions, data indexing, data aggregation and concurrent data access optimizations.



Chebotko Diagram

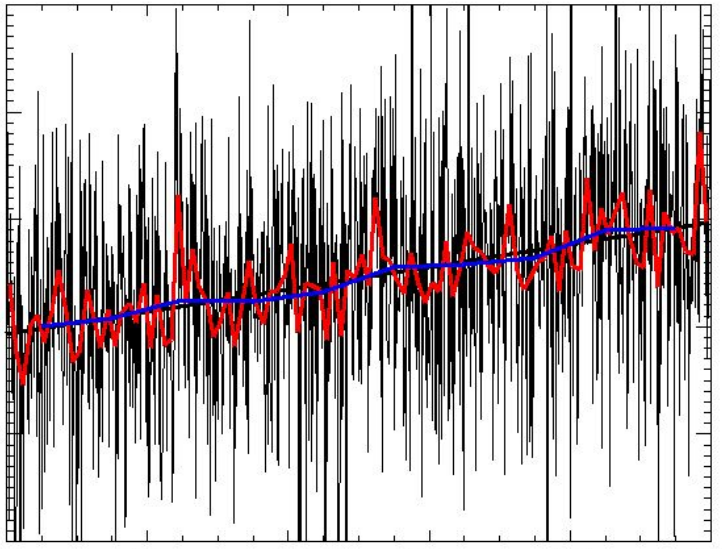
IoT Hands-On

katacoda.com/datastax/courses/cassandra-data-modeling/sensor-data

Time Series Data

Time Series Data

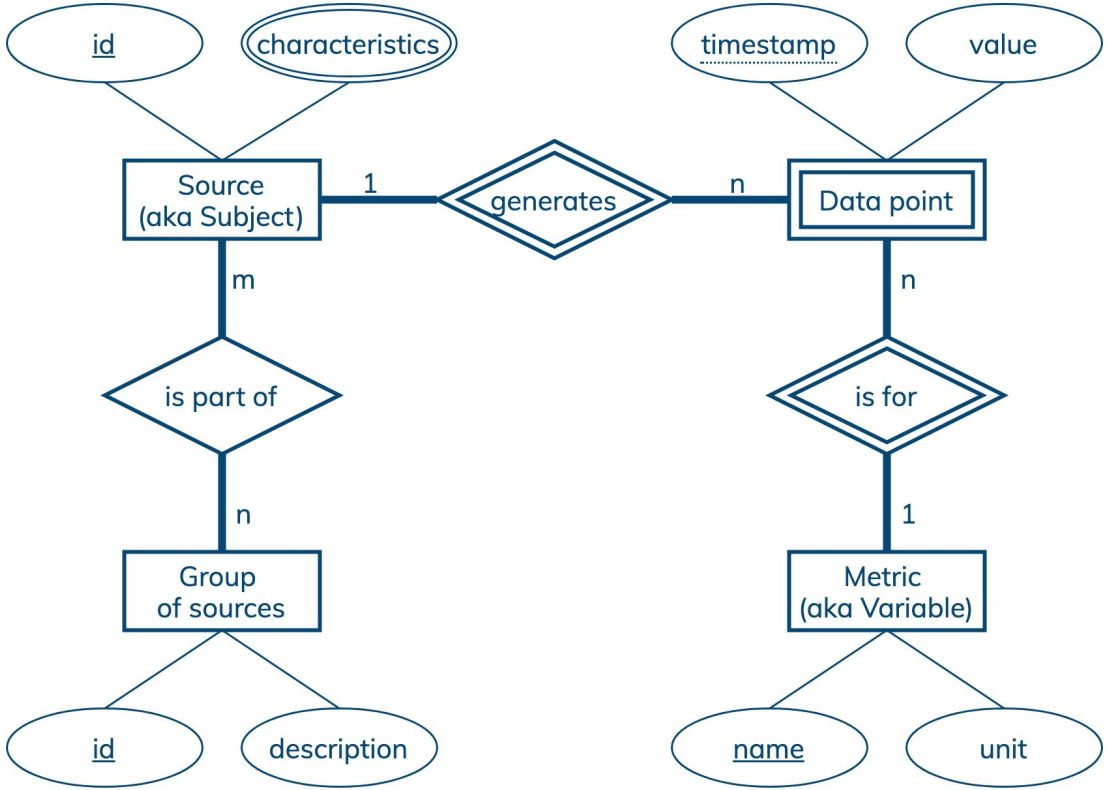
This example demonstrates how to create a data model for any time series data. It covers a conceptual data model, application workflow, logical data model, physical data model, and final CQL schema and query design. It's well generalised so it will fit to the most of time-series use-cases.



Time Series

Conceptual Data Model

A time series is a sequence of data points taken at successive and usually equally spaced out points in time. A time series is generated by a source or collected from a subject and is for a metric or variable. In the diagram, the conceptual data model for time series features data sources, groups of related sources, metrics and data points.



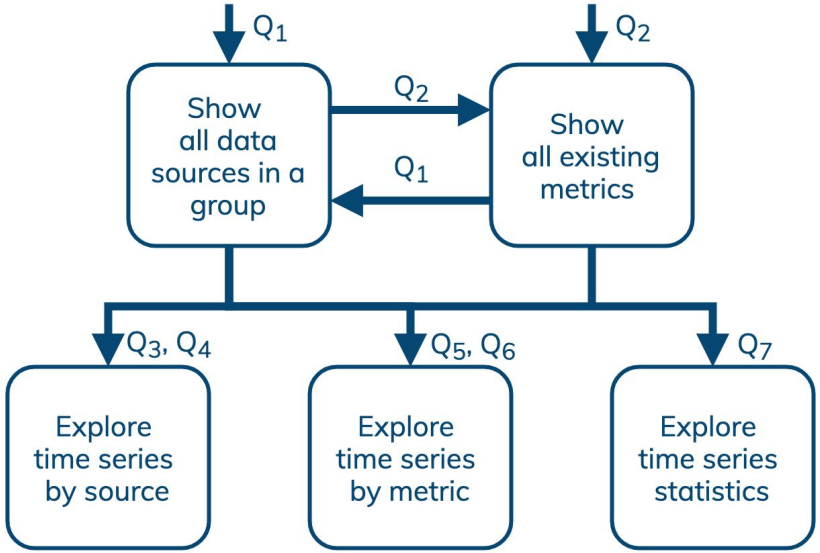
Entity-Relationship Diagram

Time Series

Application Workflow

Data access patterns

Q_1: Find information about all data sources in a particular group
Q_2: Find information about all metrics; order by metric name (asc)
Q_3: Retrieve time series with a high resolution of 60 seconds for a known group, one or more sources and time range spanning at most 10 days from now; order by source (asc), timestamp (desc) and metric (asc)
Q_4: Retrieve time series with a low resolution of 60 minutes for a known group, one or more sources and time range spanning at most 1 year; order by source (asc), timestamp (desc) and metric (asc)
Q_5: Retrieve time series with a high resolution of 60 seconds for a specified metric, group and time range spanning at most 10 days from now; order by timestamp (desc) and source (asc)
Q_6: Retrieve time series with a low resolution of 60 minutes for a specified metric, group and time range spanning at most 1 year; order by timestamp (desc) and source (asc)
Q_7: Find daily min, max, median, mean and standard deviation values for a time series with a given source, metric and date range spanning at most 1 year; order by date (desc)



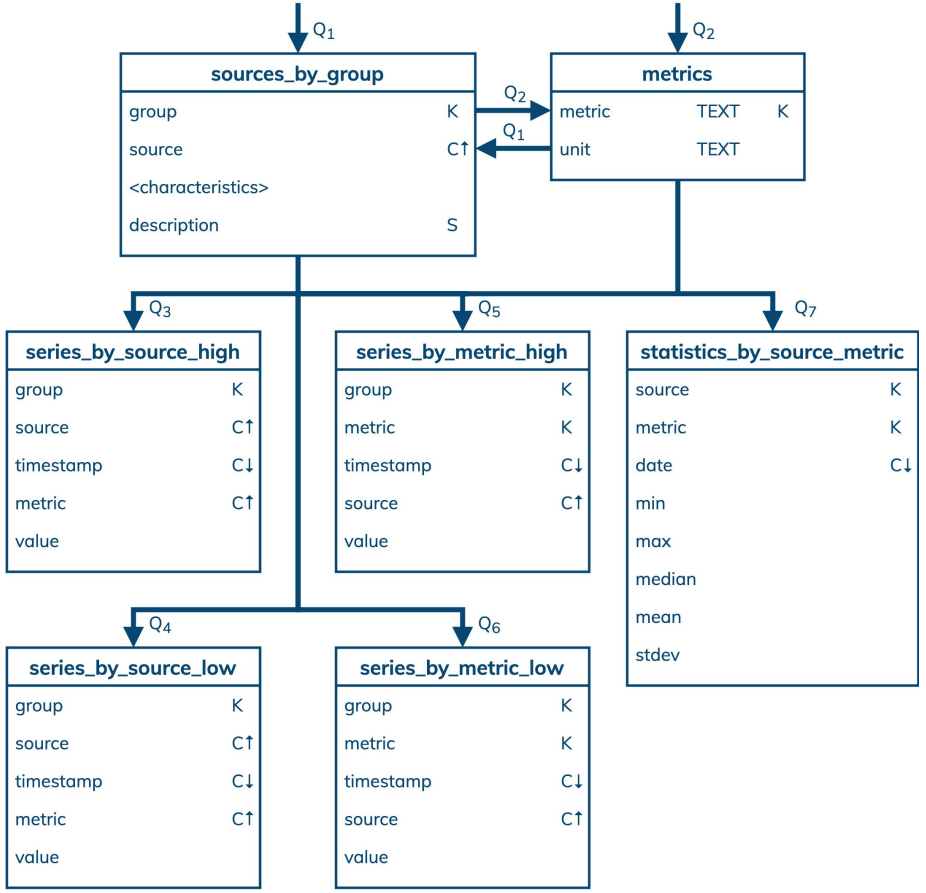
The application workflow has two entry-point tasks. The first entry-point task uses data access pattern Q1 to show all data sources from a particular group. The second entry-point task uses data access pattern Q2 to show all existing metrics. Next, an application has three options: explore time series by source using Q3 and Q4; explore time series by metric using Q5 and Q6; and explore statistics for a time series uniquely identified by a source and a metric using Q7.

Time Series

Logical Data Model

The logical data model for time series data is represented by the shown Chebotko Diagram.

There are seven tables, each one supporting a unique data access pattern. Tables *sources_by_group* and *metrics* are classic examples of tables with multi-row and single-row partitions, respectively. The former nests all sources belonging to a group as rows in a single partition and uses a static column to describe the group. The latter stores each metric in a separate partition. The remaining five tables demonstrate the three common approaches to organizing time series data: modeling by data source; modeling by metric; and modeling by both data source and metric.

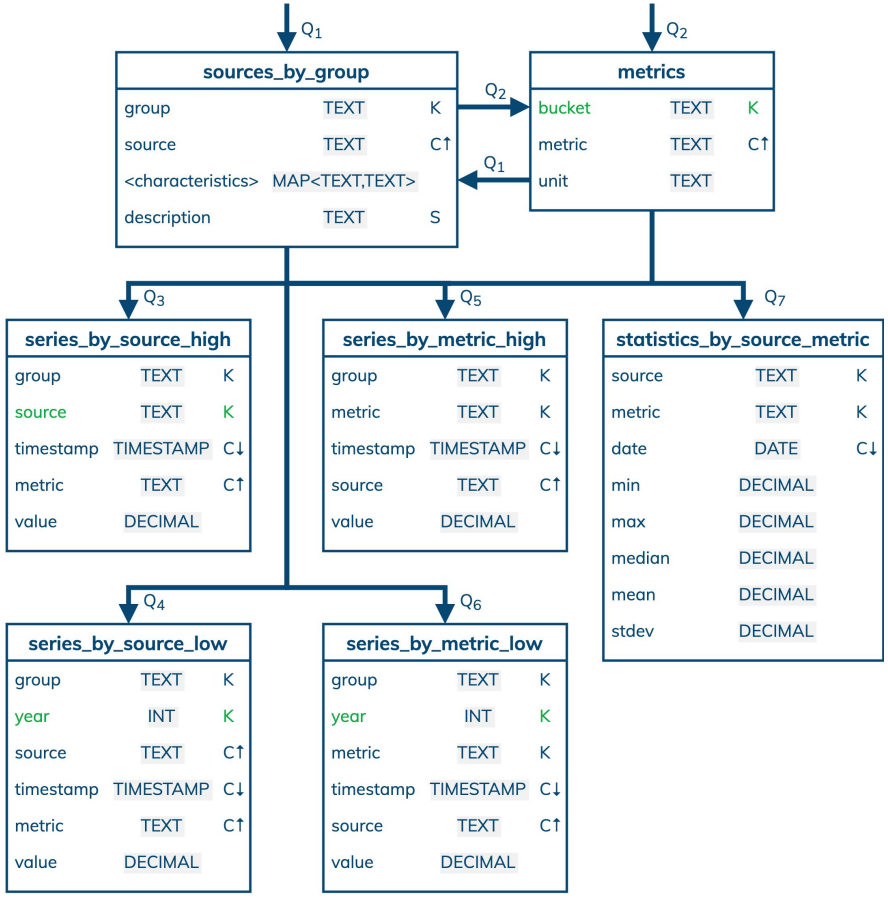


Chebotko Diagram

Time Series

Physical Data Model

All table columns have associated data types. In addition, four tables have changes in their primary keys. Table *metrics* used to be partitioned based on column *metric* and is now partitioned based on column *bucket*. The old design had single-row partitions and required retrieving rows from multiple partitions to satisfy Q2. The new design essentially merges old single-row partitions into one multi-row partition and results in much more efficient Q2. The remaining three optimizations applied to tables series_by_source_high, series_by_source_low and series_by_metric_low are all about splitting potentially large partitions.



Chebotko Diagram

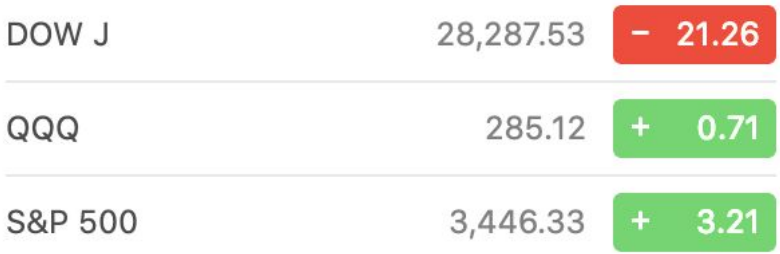
Time Series Hands-On

katacoda.com/datastax/courses/cassandra-data-modeling/time-series-data

Investment Portfolio Data

Investment Portfolio Data

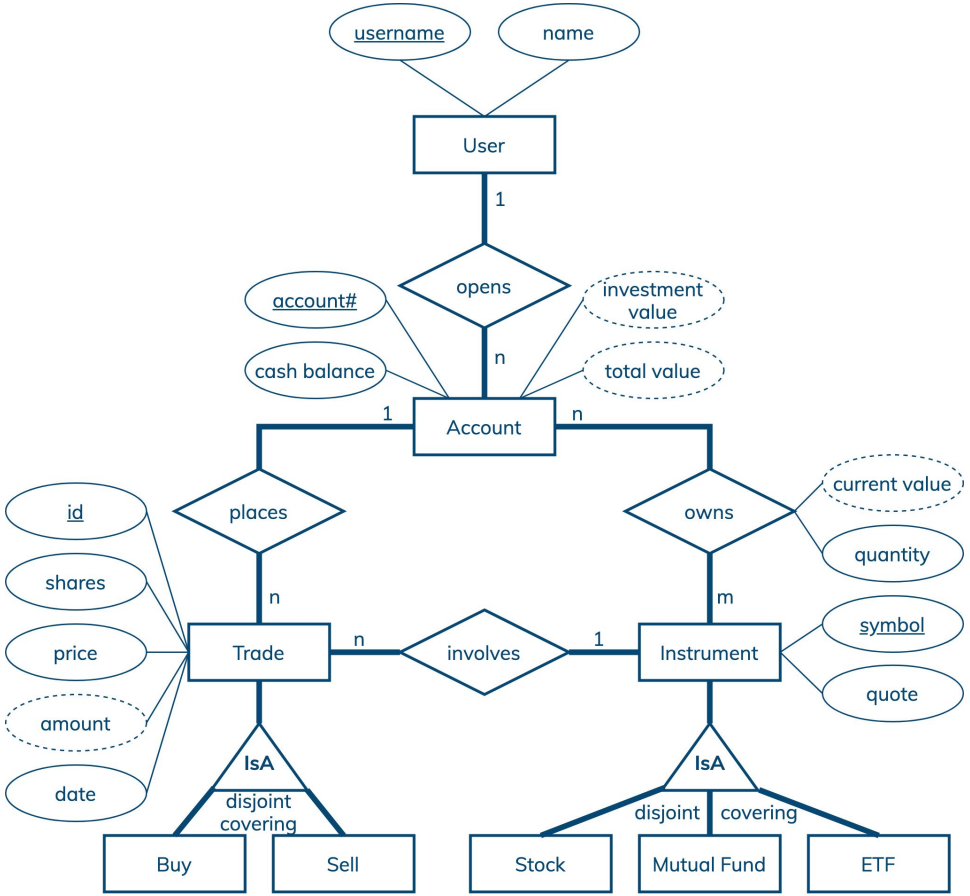
This example demonstrates how to create a data model for investment accounts or portfolios. It features users, accounts, trades and financial instruments.



Investment Portfolio

Conceptual Data Model

The conceptual data model for investment portfolio data features users, accounts, trades and instruments. A user has a unique username and may have other attributes like name. An account has a unique number, cash balance, investment value and total value. A trade is uniquely identified by an id and can be either a buy transaction or a sell transaction. Other trade attribute types include a trade date, number of shares, price per share and total amount. An instrument has a unique symbol and a current quote. Stocks, mutual funds and exchange-traded funds (ETFs) are all types of instruments supported in this example.



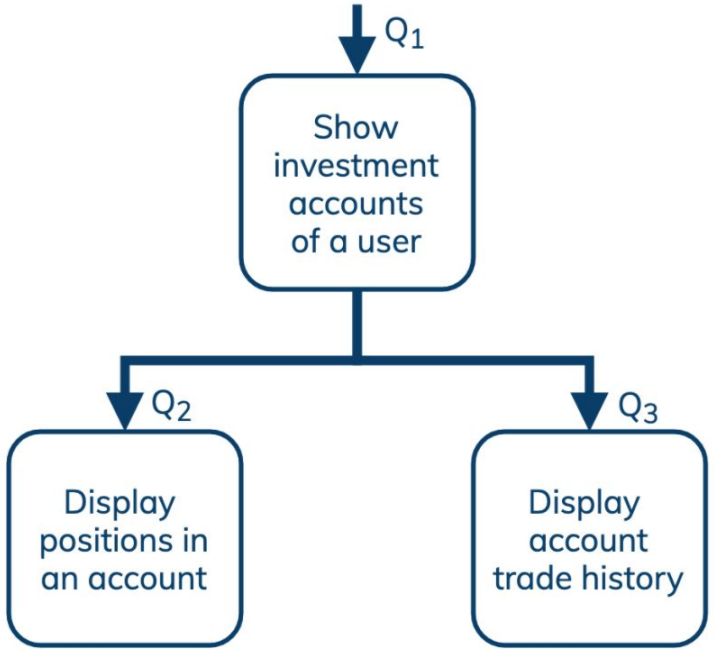
Entity-Relationship Diagram

Investment Portfolio

Application Workflow

Data access patterns

Q_1: Find information about all investment accounts of a user
Q_2: Find all positions in an account; order by instrument symbol (asc)
Q_3: Find all trades for an account and, optionally, a known date range, transaction type (buy/sell), and stock symbol; order by trade date (desc)
  $Q_{3.1}$: Find all trades for an account; order by trade date (desc)
  $Q_{3.2}$: Find all trades for an account and date range; order by trade date (desc)
  $Q_{3.3}$: Find all trades for an account, date range and transaction type; order by trade date (desc)
  $Q_{3.4}$: Find all trades for an account, date range, transaction type and instrument symbol; order by trade date (desc)
  $Q_{3.5}$: Find all trades for an account, date range and instrument symbol; order by trade date (desc)



The application workflow has an entry-point task that shows all investment accounts of a user. This task requires data access pattern Q1 as shown on the diagram. Next, an application can either display current positions in a selected account, which requires data access pattern Q2, or display information about trade history for a selected account, which requires data access pattern Q3. Furthermore, Q3 is broken down into five more manageable data access patterns Q3.1, Q3.2, Q3.3, Q3.4 and Q3.5. All in all, there are seven data access patterns for a database to support.

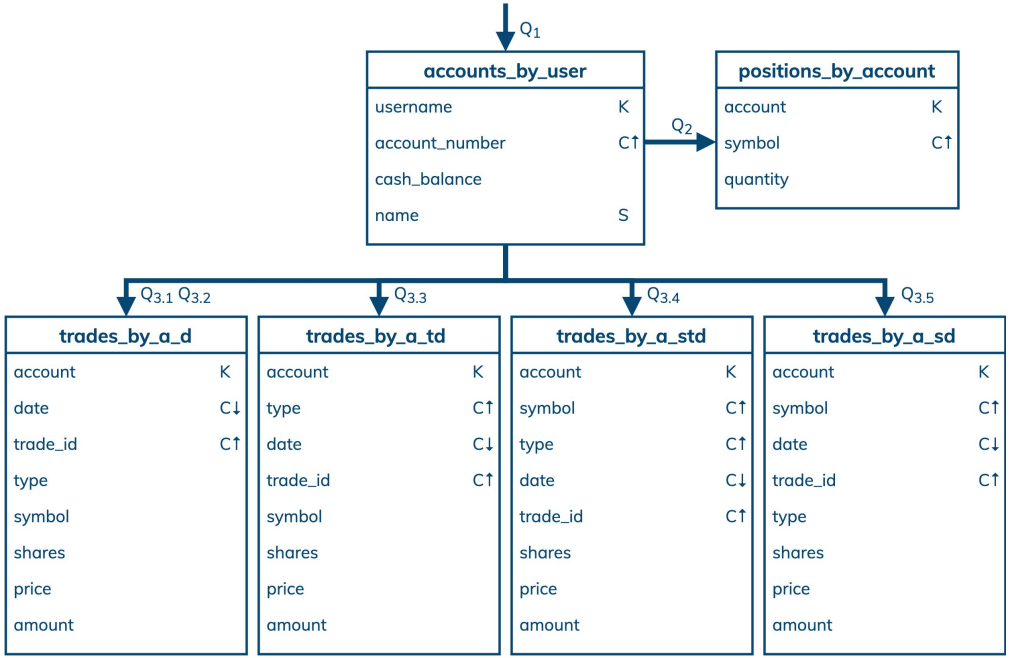
Investment Portfolio

Logical Data Model

There are six tables that are designed to support seven data access patterns. All the tables have compound primary keys, consisting of partition and clustering keys, and therefore can store multiple rows per partition.

Table *accounts_by_user* is designed to have one partition per user, where each row in a partition corresponds to a user account. Column name is a static column as it describes a user who is uniquely identified by the table partition key.

Table *positions_by_account* is designed to efficiently support Q2. All positions in a particular account can be retrieved from one partition with multiple rows. Next, tables *trades_by_a_d*, *trades_by_a_td*, *trades_by_a_std* and *trades_by_a_sd* store the same data about trade transactions, but they organize rows differently and support different data access patterns.

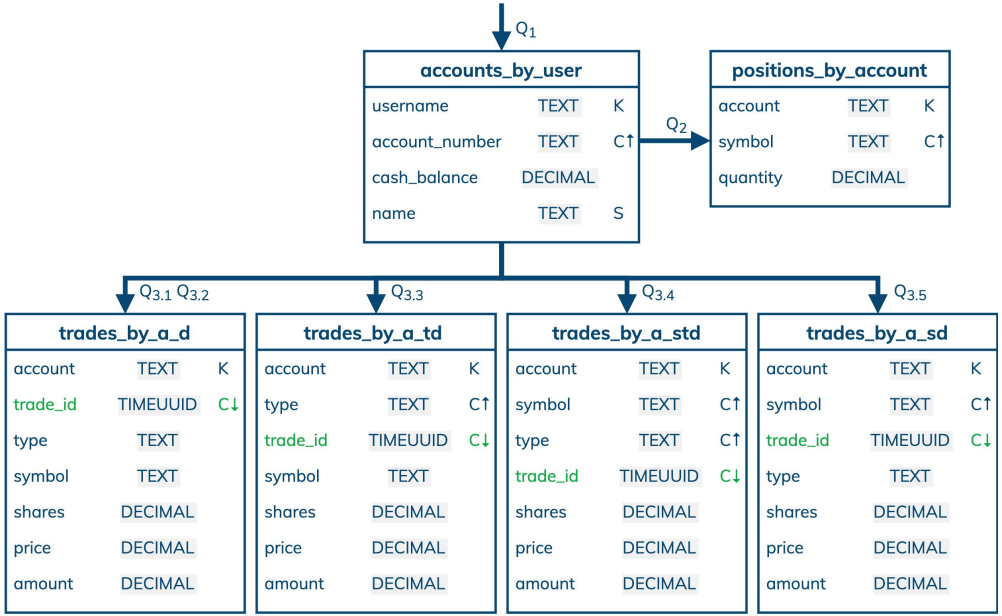


Chebotko Diagram

Investment Portfolio

Physical Data Model

Assuming the use case does not cover robo-advisors and algorithmic trading, it should be evident that none of the tables will have very large partitions. Any given user will at most have a few accounts. An account will normally have at most dozens of positions and possibly hundreds or thousands of trades over its lifetime.



Chebotko Diagram

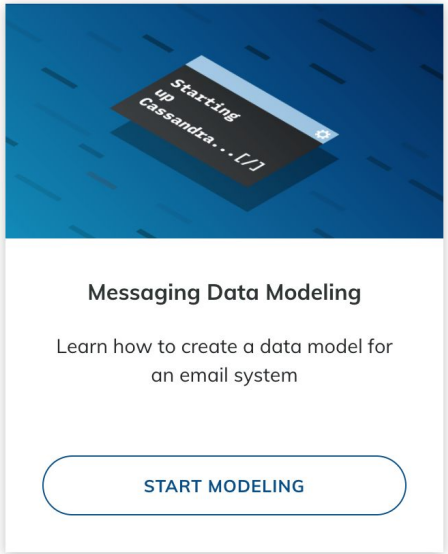
Investment Portfolio Hands-On

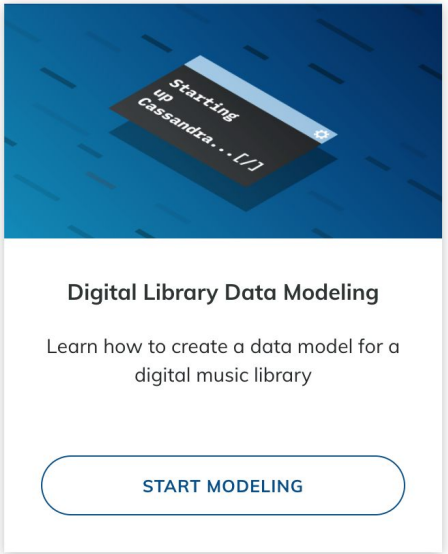
katacoda.com/datastax/courses/cassandra-data-modeling/investment-data

Workshop Assignment

What's Next?

1. Walk through 2 more examples, **Messaging** and **Digital Library**.
   datastax.com/learn/data-modeling-by-example
2. Ask your questions
   a. community.datastax.com
   b. Discord discord.gg/pPjPcZN
3. Get your free voucher, pass the exam and become **Cassandra Certified Developer!**
   bit.ly/GET-VOUCHER





DATASTAX

Thank You!

YOU ARE AMAZING!