Lecture 7

CMPT 732 - Fall 2022

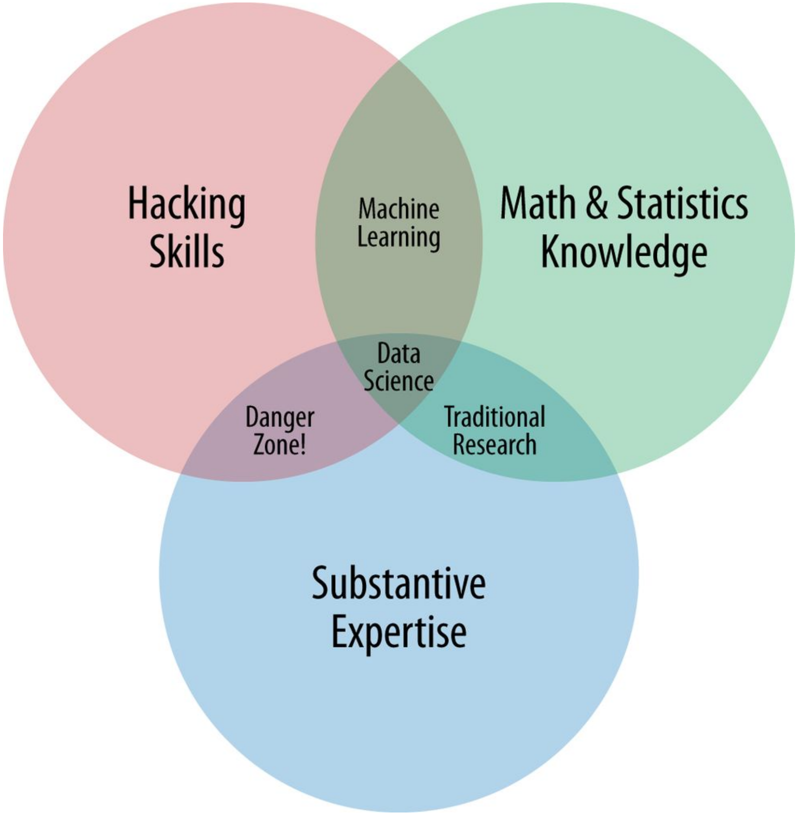
Term Project Proposals

Problem Statement

- What questions/problems do you presume your user to have?
- How does your user act on your findings/conclusions?

Descriptive vs Predictive Analytics

Data Science



https://learning.oreilly.com/library/view/the-culture-of/9781491946701/ch01.html#no_buck_rogers

21st Century Databases

CMPT 732 - Fall 2022

Agenda

- Re-examining files & HDFS
- Revisiting (Relational) Databases
- NoSQL Databases
- Cassandra*

The Story so Far: HDFS vs Conventional filesystem

HDFS

- Distributed & replicated storage
- Collocated computation
- Horizontal scaling

Conventional Filesystem

- Centralized architecture
- “Remote” computation
- Vertical scaling

HDFS in a nut-shell

A filesystem/approach for...

storing data to be processed…

in a manner suitable for parallel processing.

Database

?

Relational Databases Today

Oracle

SQL Server

DB2

Teradata

Sybase

PostgreSQL

MySQL

...

File vs Database

Features	**File**	**Database**
Schema/Structure	?	✅
Security (AAA)	✅ ? ?	✅
Life Cycle	?	✅

Relational Database: ACID

Atomicity

Consistency

Isolation

Durable

Relational data model

- Table contain *rows*
- Rows of table conform to schema of *column names* and *types*
- Queryable via language (SQL) featuring algebraic operators:
  - Set operations (union, difference, etc)
  - Projection
  - Selection
  - Rename
  - Cartesian product/join

System R: Relational Approach to Database Management

M. M. ASTRAHAN, M. W. BLASGEN, D. D. CHAMBERLIN,
K. P. ESWARAN, J. N. GRAY, P. P. GRIFFITHS,
W. F. KING, R. A. LORIE, P. R. McJONES, J. W. MEHL,
G. R. PUTZOLU, I. L. TRAIGER, B. W. WADE, AND V. WATSON

IBM Research Laboratory

System R is a database management system which provides a high level relational data interface. The system provides a high level of data independence by isolating the end user as much as possible from underlying storage structures. The system permits definition of a variety of relational views on common underlying data. Data control features are provided, including authorization, integrity assertions, triggered transactions, a logging and recovery subsystem, and facilities for maintaining data consistency in a shared-update environment.

This paper contains a description of the overall architecture and design of the system. At the present time the system is being implemented and the design evaluated. We emphasize that System R is a vehicle for research in database architecture, and is not planned as a product.

Key Words and Phrases: database, relational model, nonprocedural language, authorization, locking, recovery, data structures, index structures
CR categories: 3.74, 4.22, 4.33, 4.35

NoSQL gang (~2009)

Voldemort

Cassandra

Dynamite

HBase

Hypertable

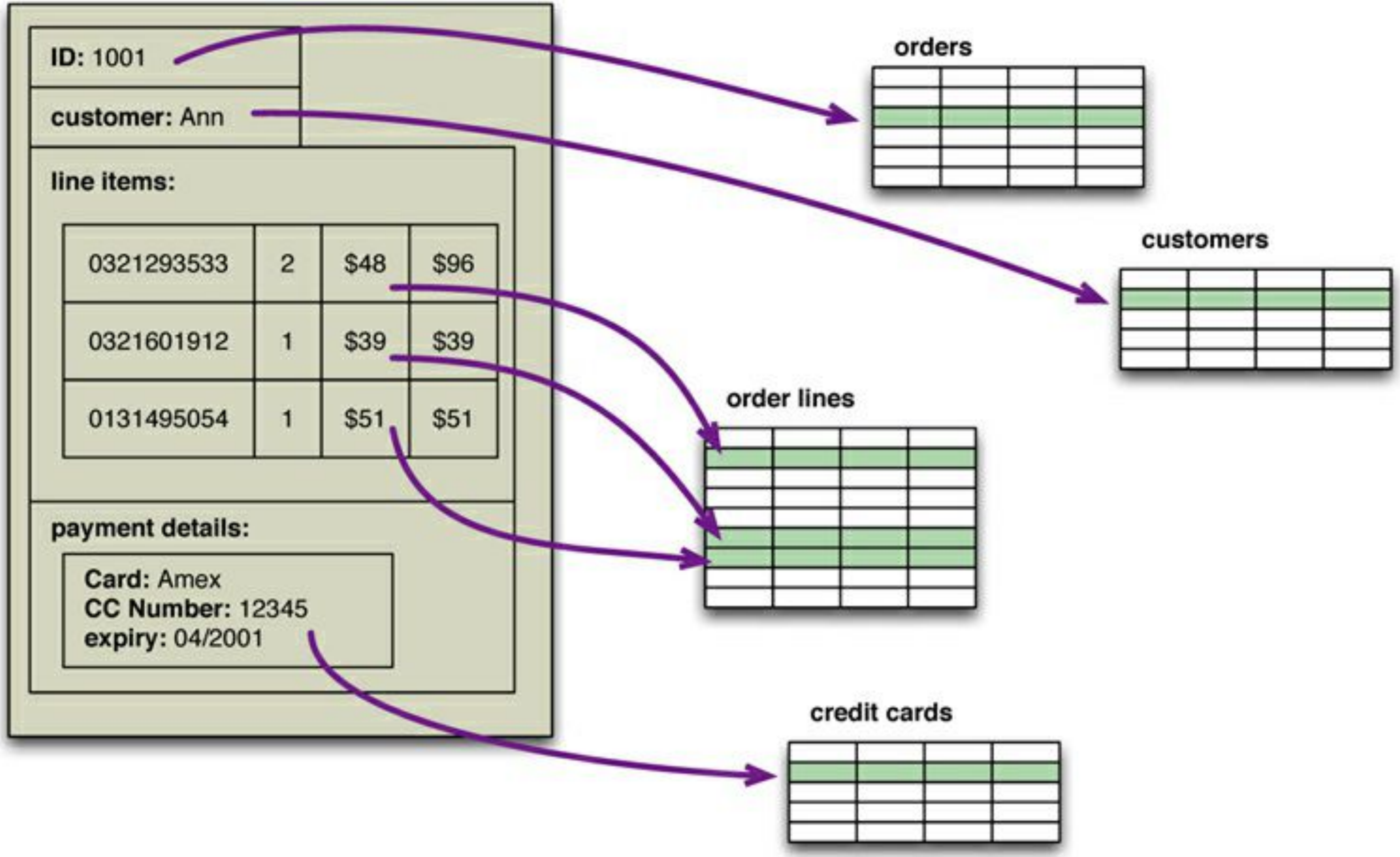
CouchDB

MongoDB

Why NoSQL?

1: Object-Relational Impedance Mismatch

On-disk storage vs In-memory representation



https://martinfowler.com/books/nosql.html

2: CAP Theorem/Brewer’s Conjecture (2000)

It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

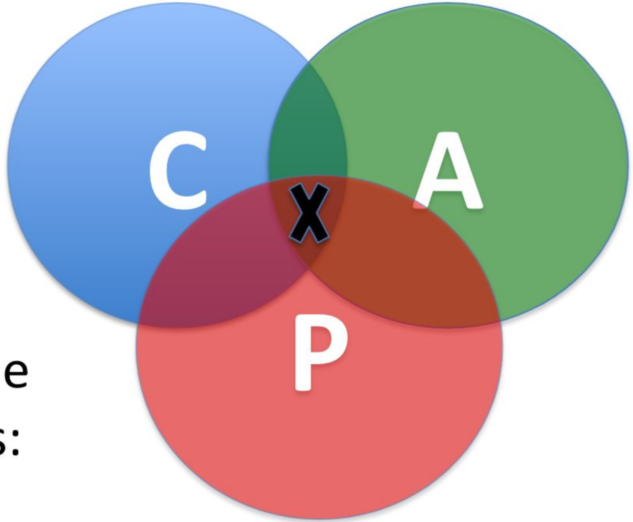
Consistency: Every read receives the most recent write or an error.

Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write.

Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

CAP => 3 types of systems

- Of the following three guarantees potentially offered by distributed systems:
  - Consistency
  - Availability
  - Partition tolerance
- Pick two
- This suggests there are three kinds of distributed systems:
  - CP
  - AP
  - CA



Reconsidering Sharding

Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

https://www.digitalocean.com/community/tutorials/understanding-database-sharding

3: Polyglot Persistence

"...a variety of different data storage technologies for different kinds of data." **Martin Fowler (~2006)**

https://martinfowler.com/bliki/PolyglotPersistence.html

https://martinfowler.com/bliki/PolyglotPersistence.html

Why NoSQL?

Object-Relational Impedance Mismatch
CAP Theorem/Brewer's Conjecture (2000)
Polyglot Persistence

Open Source

NoSQL - What’s in a Name

No SQL !

Not **O**nly

SQL

NoSQL

¯_(ツ)_/¯

NoSQL Categories

Category	Data Model	Examples
Key-value	hash/dictionary	Redis, Membase, Amazon SimpleDB
Document	Semi-structured (JSON)	MongoDB, Couchbase, Amazon DynamoDB
Wide-column	BigTable	Cassandra, HBase, BigTable
Graph	graph	Neo4j, InfoGrid
Search	Text-based document	ElasticSearch, Solr, Splunk

NoSQL Database Characteristics

- Does not use the relational model
  - Tables ✔
  - Normalization (Keys & Join) ✖

NoSQL Database Characteristics

- Does not use the relational model
- Runs well on a cluster of computers
  - Architected for horizontal scaling

NoSQL Database Characteristics

- Does not use the relational model
- Architected for cluster of computers
- Open source
  - Economical
  - Agility
  - Evolution

NoSQL Database Characteristics

- Does not use the relational model
- Architected for cluster of computers
- Open source
- Schema-less
  - Simplifies the collection of data and the operation of a database

NoSQL Categories

Category	Data Model	Examples
Key-value	hash/dictionary	Redis, Membase, Amazon SimpleDB
Document	Semi-structured (JSON)	MongoDB, Couchbase, Amazon DynamoDB
Wide-column	BigTable	Cassandra, HBase, BigTable
Graph	graph	Neo4j, InfoGrid
Search	Text-based document	ElasticSearch, Solr, Splunk

cassandra

Cassandra

Developed by Facebook

Donated to Apache Software Foundation

Championed commercially by Datastax

Cassandra

is a…

…horizontally scalable,
fault-tolerant,
distributed,
eventually consistent,
Dynamo-based,
BigTable-inspired,
sparse,
nested hash table

https://en.wikipedia.org/wiki/Apache_Cassandra

Download as PDF
Printable version

In other projects

Wikimedia Commons

Languages

Deutsch
Español
Français
한국어
Italiano
日本語
Português
Русский
中文
8 more

Edit links

License	Apache License 2.0
Website	cassandra.apache.org

History [edit]

Avinash Lakshman, one of the authors of Amazon's Dynamo, and Prashant Malik initially developed Cassandra at Facebook to power the Facebook inbox search feature. Facebook released Cassandra as an open-source project on Google code in July 2008.[4] In March 2009 it became an Apache Incubator project.[5] On February 17, 2010 it graduated to a top-level project.[6]

Facebook developers named their database after the Trojan mythological prophet Cassandra, with classical allusions to a curse on an oracle.[7]

Releases [edit]

Releases after graduation include

- 0.6, released Apr 12 2010, added support for integrated caching, and Apache Hadoop MapReduce[8]
- 0.7, released Jan 08 2011, added secondary indexes and online schema changes[9]
- 0.8, released Jun 2 2011, added the Cassandra Query Language (CQL), self-tuning memtables, and support for zero-downtime upgrades[10]
- 1.0, released Oct 17 2011, added integrated compression, leveled compaction, and improved read-performance[11]
- 1.1, released Apr 23 2012, added self-tuning caches, row-level isolation, and support for mixed ssd/spinning disk deployments[12]
- 1.2, released Jan 2 2013, added clustering across virtual nodes, inter-node communication, atomic batches, and request tracing[13]
- 2.0, released Sep 4 2013, added lightweight transactions (based on the Paxos consensus protocol), triggers, improved compactions
- 2.1 released Sep 10 2014[14]
- 2.2 released July 20, 2015
- 3.0 released November 11, 2015
- 3.1 through 3.10 releases were monthly releases using a tick-tock-like release model, with even-numbered releases providing both new features and bug fixes while odd-numbered releases will include bug fixes only.[15]
- 3.11 released June 23, 2017 as a stable 3.11 release series and bug fix from the last tick-tock feature release.
- 4.0 released July 26, 2021.
- 4.0.1 released September 7, 2021.

Version	Original release date	Latest version	Release date	Status[16]
0.6	2010-04-12	0.6.13	2011-04-18	No longer supported
0.7	2011-01-10	0.7.10	2011-10-31	No longer supported
0.8	2011-06-03	0.8.10	2012-02-13	No longer supported
1.0	2011-10-18	1.0.12	2012-10-04	No longer supported
1.1	2012-04-24	1.1.12	2013-05-27	No longer supported
1.2	2013-01-02	1.2.19	2014-09-18	No longer supported
2.0	2013-09-03	2.0.17	2015-09-21	No longer supported
2.1	2014-09-16	2.1.22	2020-08-31	No longer supported
2.2	2015-07-20	2.2.19	2020-11-04	Still supported, critical fixes only
3.0	2015-11-09	3.0.24	2021-02-28	Still supported
3.11	2017-06-23	3.11.10	2021-02-28	Still supported

Cassandra vs RDBMS

Table (was “column family” prior to CQL 3) ✅

Typed Columns (since ~v0.7) ✅

CQL (since v0.8), not SQL ✅

Table Join ❌

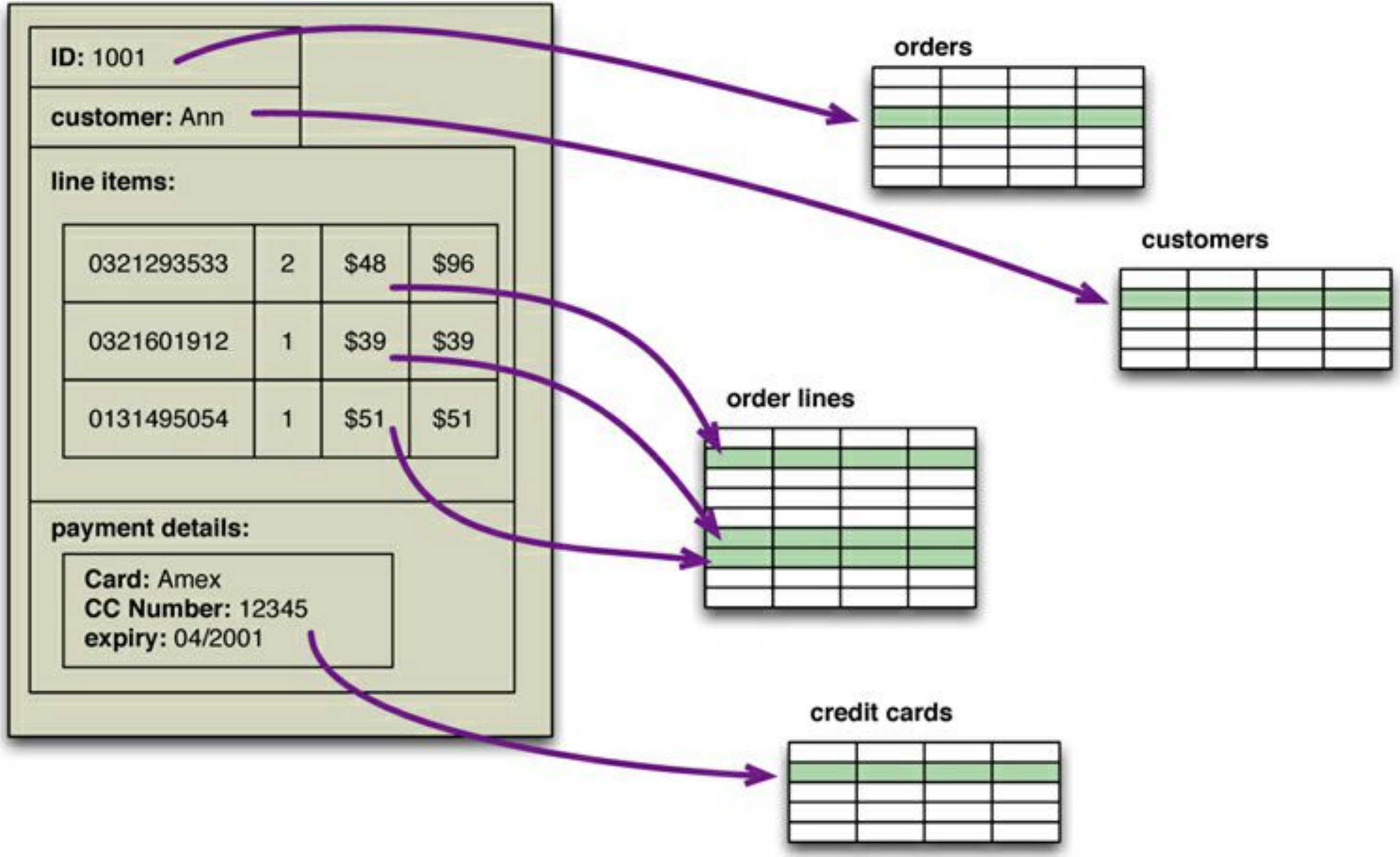
Cassandra vs RDBMS

Tables ✅

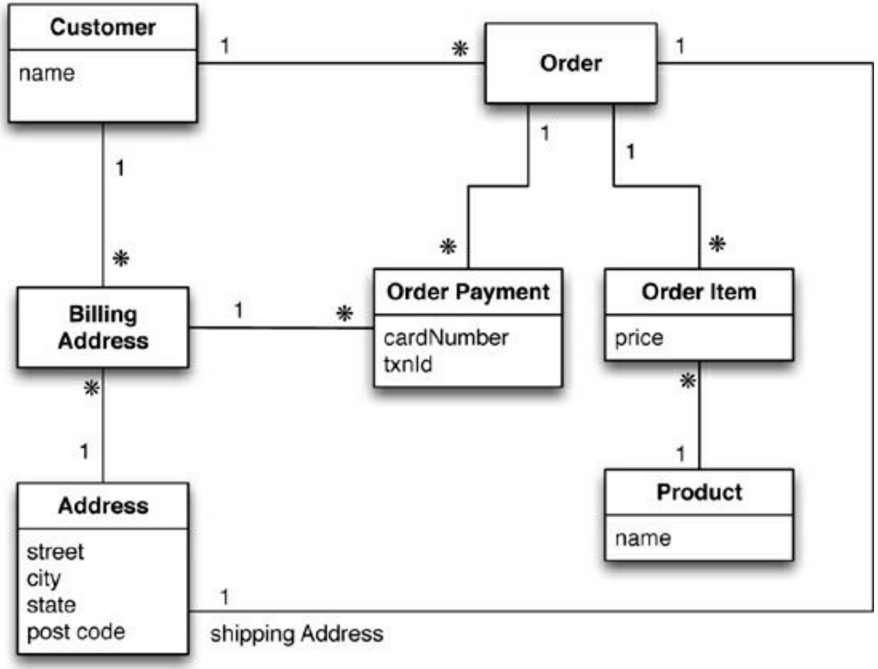
Typed columns ✅

CQL ✅

Table Join ❌

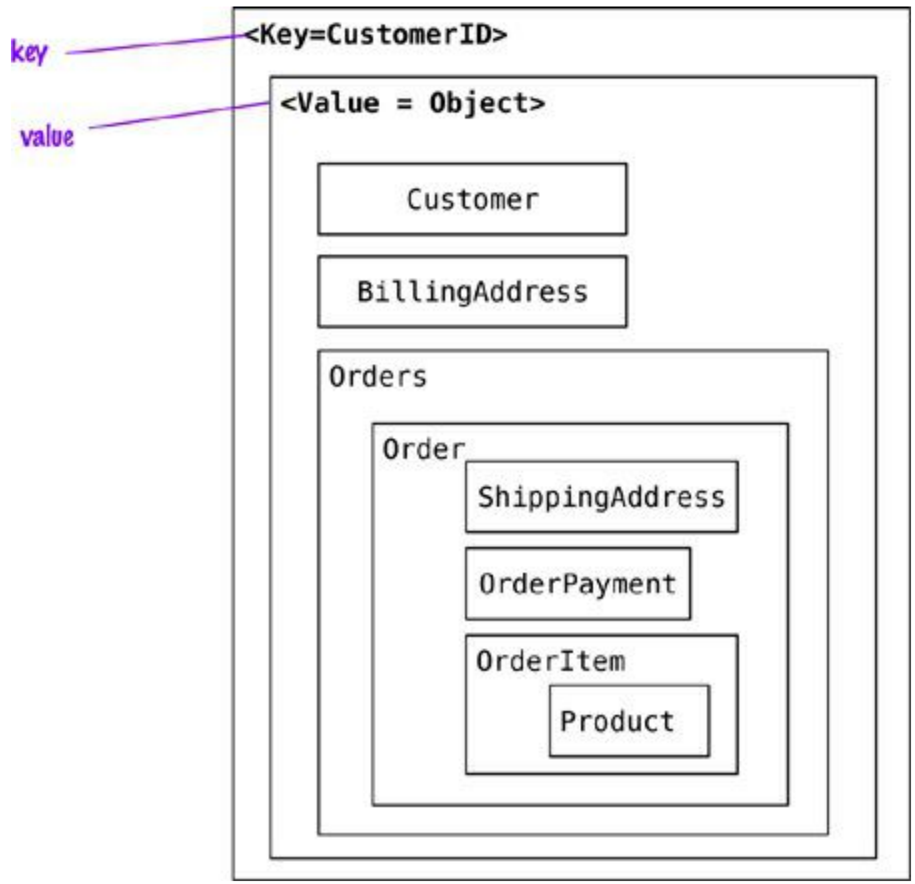


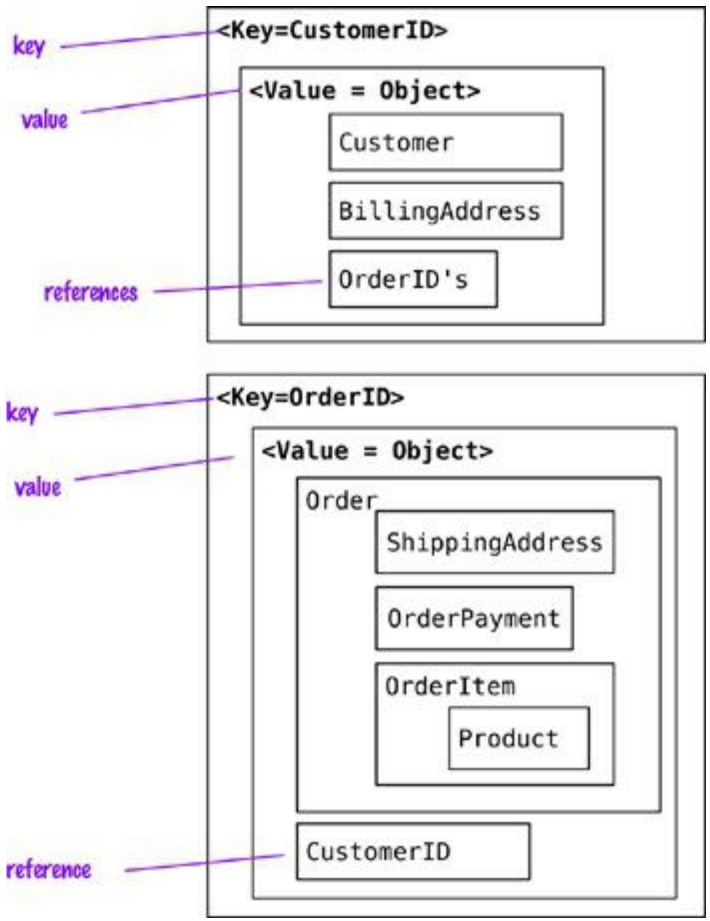
https://martinfowler.com/books/nosql.html

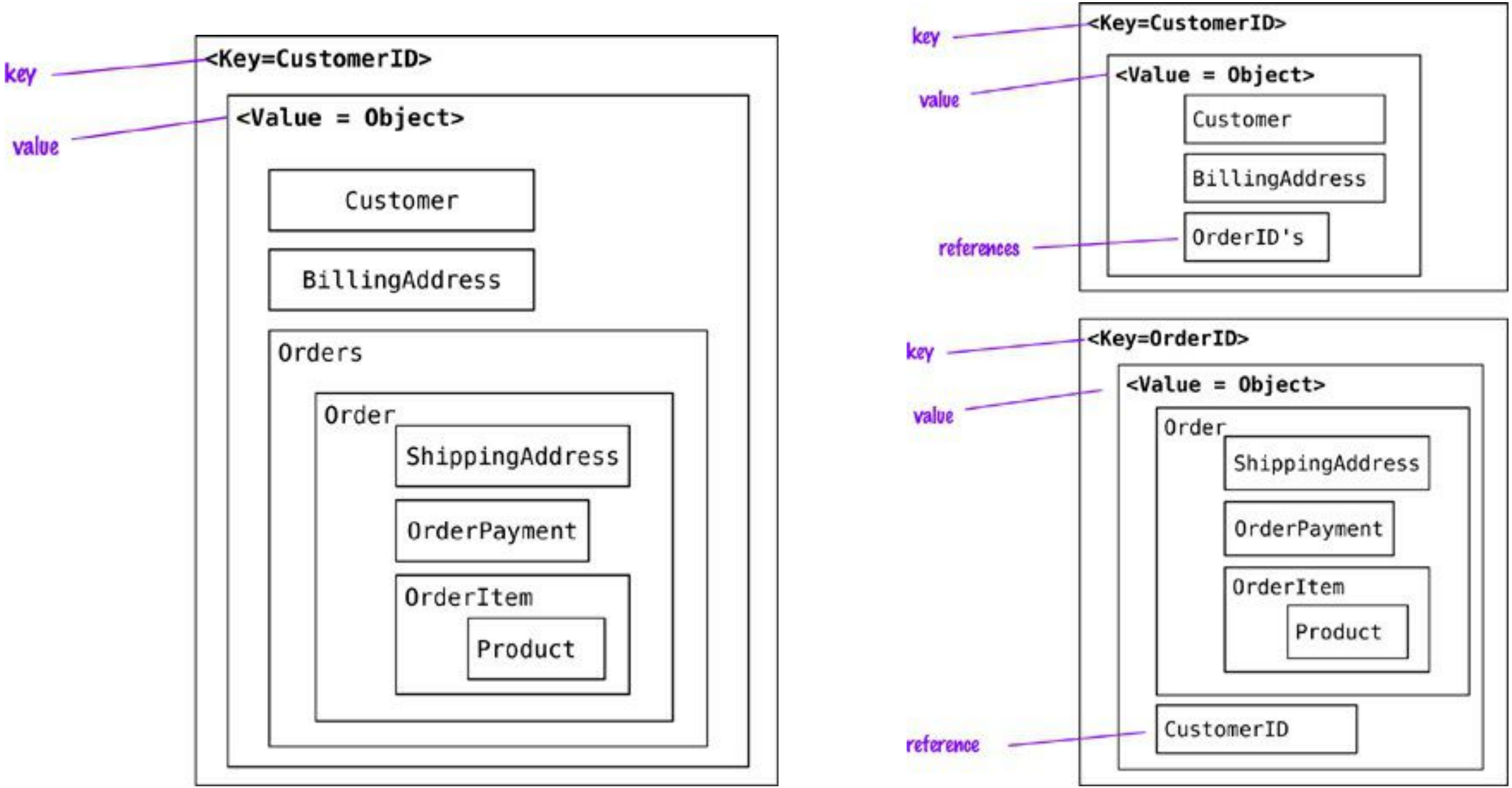



```
public class Customer
{
  private String name;
  private BillingAddr billAddr;
  private Order order;
  ...
}

public class Order
{
  private Customer customer;
  private OrderPayment orderPayment;
  private OrderItem[] orderItems;
  …
}
```


key
value
<Key=CustomerID>
<Value = Object>
Customer
BillingAddress
Orders
Order
ShippingAddress
OrderPayment
OrderItem
Product


key
<Key=CustomerID>
value
<Value = Object>
Customer
BillingAddress
references
OrderID's
key
<Key=OrderID>
value
<Value = Object>
Order
ShippingAddress
OrderPayment
OrderItem
Product
reference
CustomerID


key
value
<Key=CustomerID>
<Value = Object>
Customer
BillingAddress
Orders
Order
ShippingAddress
OrderPayment
OrderItem
Product
key
value
references
<Key=CustomerID>
<Value = Object>
Customer
BillingAddress
OrderID's
key
value
reference
<Key=OrderID>
<Value = Object>
Order
ShippingAddress
OrderPayment
OrderItem
Product
CustomerID

TL;DR: Denormalization

Model your data in Cassandra in a manner consistent with how your application operates (queries, updates) on the data.

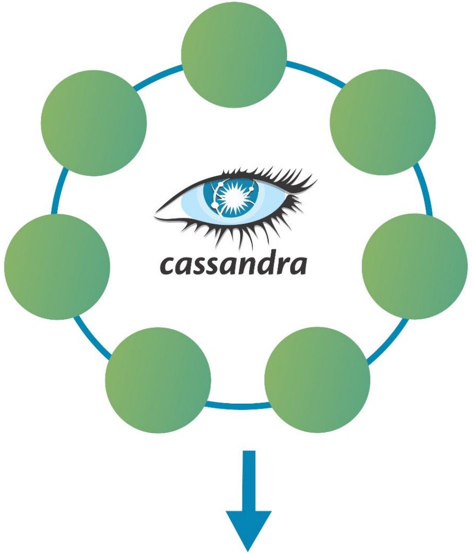
Cassandra as a NoSQL store

Distributed and decentralized ✅

Horizontally scalable ✅

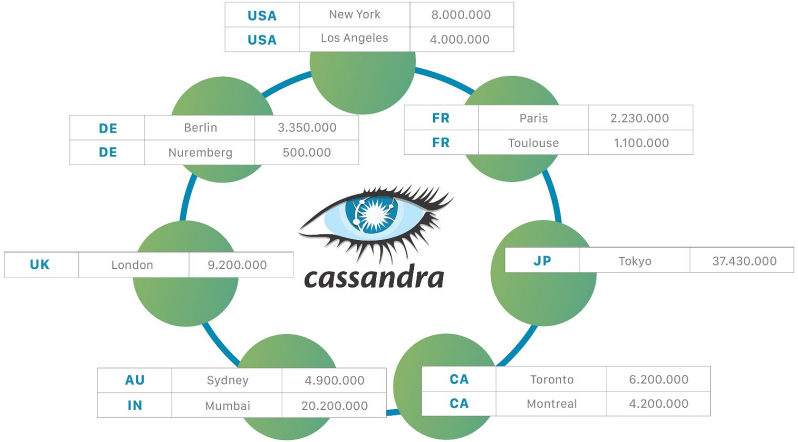
Fault tolerant ✅

Tunable consistency



COUNTRY	CITY	POPULATION
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key



https://cassandra.apache.org/_/cassandra-basics.html

Cassandra as a NoSQL store

Distributed and decentralized ✅

Horizontally scalable ✅

Fault tolerant ✅

Tunable consistency ✅

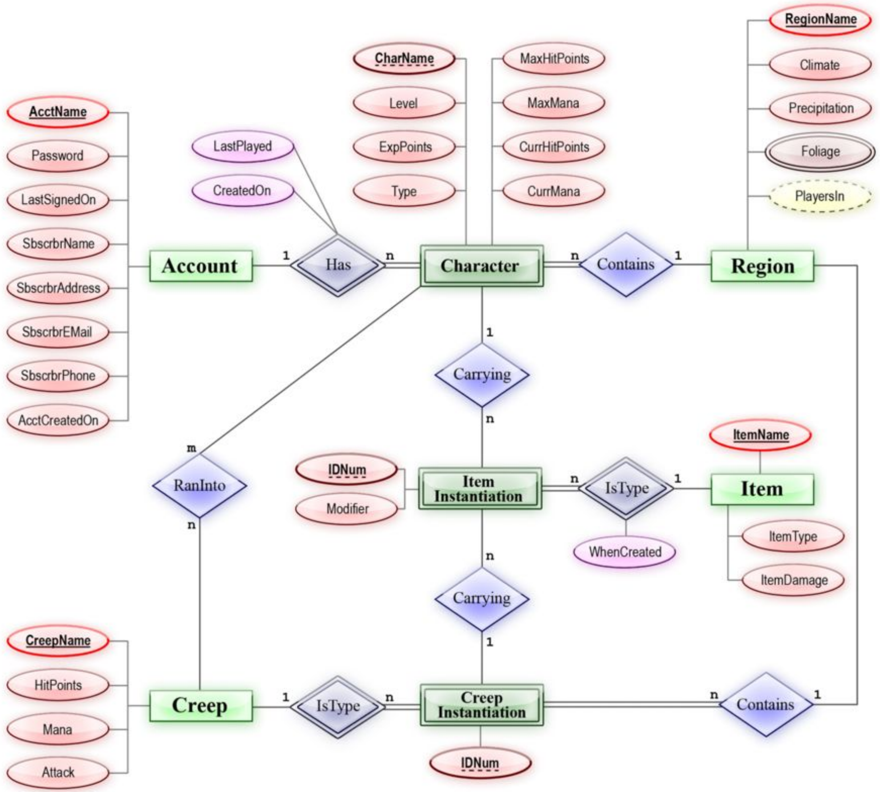
Pattern

Hadoop

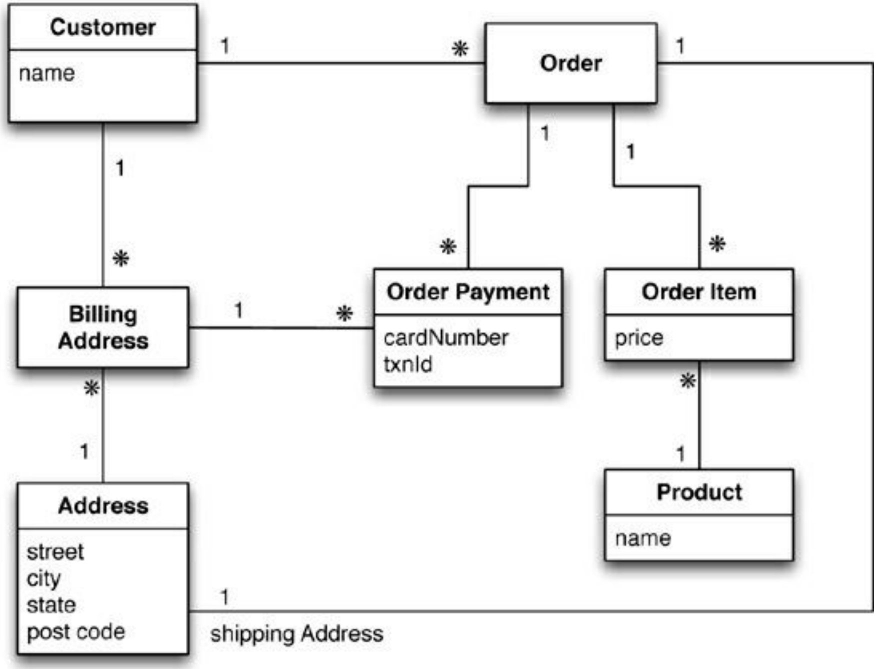
- Cluster of machines
- Distributed & replicated data
- Distributed workload
- Horizontal scaling

“Conventional Architecture”

- Single* machine
- Single* copy of data
- Centralized* workload
- Vertical scaling




```
public class MyObject
{
  private String someField1;
  private Int someField2;
  private Boolean someField3;
  private MyStructure someStruct4;
  private MyArray[] someArray5;
  ...
}
```




```
public class Customer
{
  private String name;
  private BillingAddr billAddr;
  private Order order;
  ...
}

public class Order
{
  private OrderPayment orderPayment;
  private OrderItem[] orderItems;
  …
}
```

Cassandra Data Model

Storage divided into *keyspaces* (analogous to RDBMS's catalog/schema)

Each keyspace declares how data is replicated

Each keyspace contains one or more *tables*

Each table has a declared schema (column names and data types)

Data for each table distributed across the cluster by a *hash*

Hash is computed from the table's *primary key*