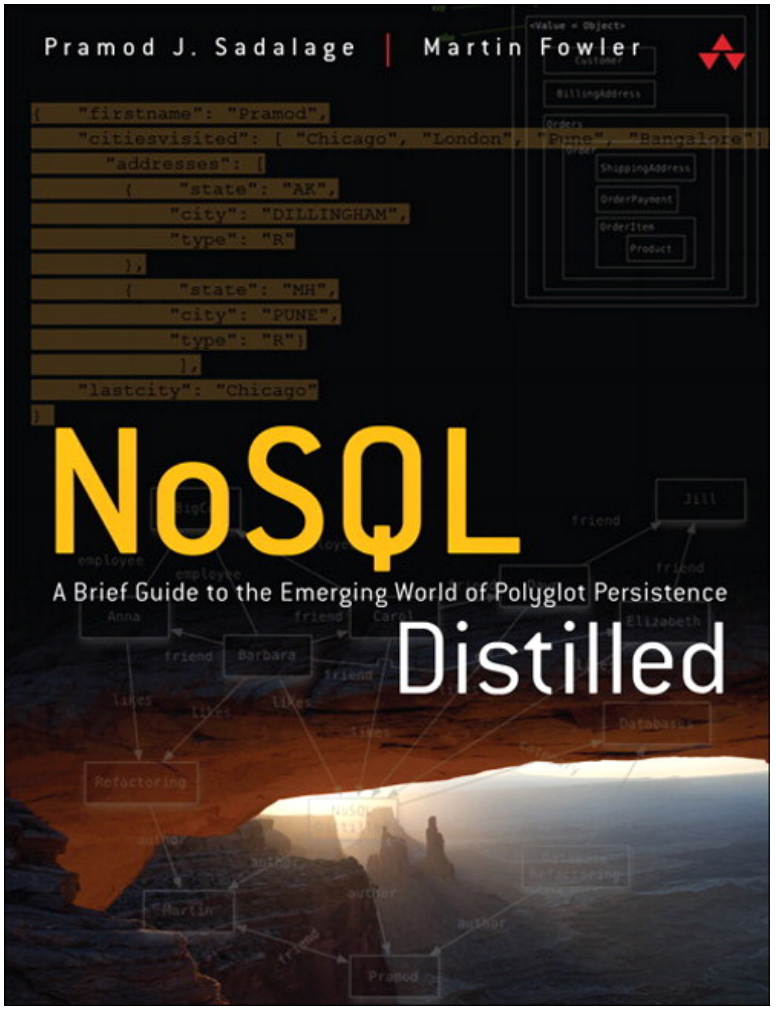
Data Models Intro

13 November 2017

Ignacio Coterillo
Computer Engineer, CERN

Reference Book



Data Model

Two overlapping point of views:

- How the data is modelled at the application level
- How the data is modelled at the database level, affecting both its physical and logical representation

Data Model: RDBMS

In a RDBMS you normally have:

- A set of tables
- Comprised by a set of rows (tuples)
- Rows have a pre-defined set of elements with fixed type and order
- Tuples can have references to other tuples, creating relationships

Data Model: RDBMS

Tuples are limited in that:

- You can't nest them
- You can't mix tuples with tuples values in structures such as lists

Oracle Nested Tables (http://www.orafaq.com/wiki/NESTED_TABLE)

MSSQL Nested Tables (https://technet.microsoft.com/en-us/library/ms175659%28v=sql.110%29.aspx)

PostgreSQL Composite Types (https://www.postgresql.org/docs/9.6/static/rowtypes.html)

Data Model: Aggregates

Instead of tuples with a fixed structure, a complex structure that can store lists and other complex structures.

```
In [8]: item
Out[8]: {'age': 1, 'name': 'test item', 'privileges': [1, 3, 34, 6, 'b']}

In [9]: item2
Out[9]:
{'age': 0.5,
 'depends': {'age': 1, 'name': 'test item', 'privileges': [1, 3, 34, 6, 'b']},
 'name': 'test item 2',
 'privileges': [1, 'c']}
```

Data Model: Aggregates

Aggregate: A collection of related objects to be treated as a unit. [Domain-Driven Design, Evans]

In particular, a unit for data manipulation and consistency management.
Typically:

- Aggregates are updated in atomic operations
- Storage layer communication is done in term of aggregates
- Dealing in aggregates makes is easier for database systems to handle operations a distributed setting, with the aggregate as the natural unit for replication and sharding

Data Model: Consequences

In the relational model Foreign Keys (external references) can be used to express relations between tuples and logically compose an *aggregate equivalent* from said tuples.

These compositions (via JOIN operations) are computationally expensive and grow complex to express and operate with the complexity of the *aggregate*.

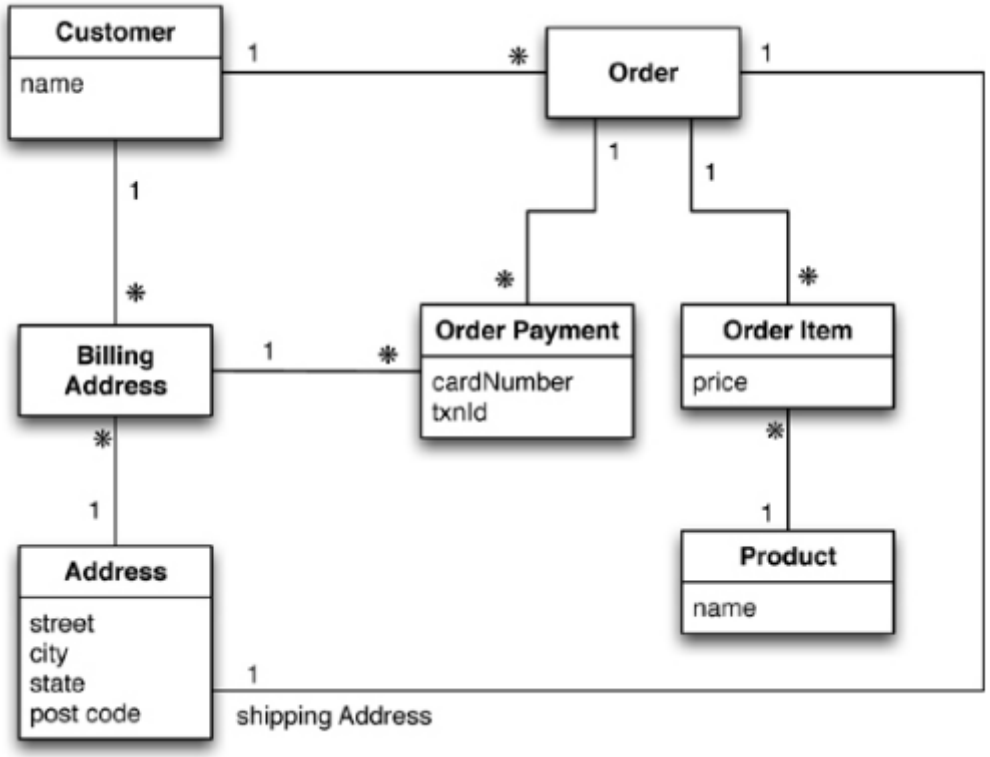
The database system is not aware of which relations indicate a conceptual entity, and can't use the information to optimize data distribution or partition for distributed access.

Data Model: Consequences

A note about transactions:

- RDMBS allow to manipulate a combination of rows from any tables in a single transaction (ACID)
- In general, aggregate oriented databases don't have ACID transactions spanning multiple aggregates but operations on a single aggregate are atomic

Data Model Example: RDBMS



Data Model Example: RDBMS

Customer

Id	Name
1	Martin

Order

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Aggregate Model (JSON)

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

Aggregate Model (JSON)

```
// in customers
{
"customer": {
"id": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"orders": [
  {
    "id":99,
    "customerId":1,
    "orderItems":[
    {
    "productId":27,
    "price": 32.45,
    "productName": "NoSQL Distilled"
    }
  ],
  "shippingAddress":[{"city":"Chicago"}]
```

Data Model: Aggregates

Aggregate oriented:

- Key-Value
- Document
- Column-Family Stores

Other models:

- Graph
- Time Series

Key-Value

Information (Values) can be stored and accessed by "name" (key).

The system is unaware of the value structure and properties:

```
aggregate=# \d+ kv
                              Table "public.kv"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 key    | character varying(32) |           | extended |              |
 value  | bytea                 |           | extended |              |
```

Most KV systems expand this functionality:

- **Riak**: Metadata links
- **Redis**: Complex values like lists or sets, plus extra functionality

Document

The system is aware of the value structure and properties:

```
aggregate=# \d+ document ;
                            Table "public.document"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 key    | character varying(32) |           | extended |              |
 value  | json                  |           | extended |              |

aggregate=# create index on document((value->>'name'));
CREATE INDEX
```

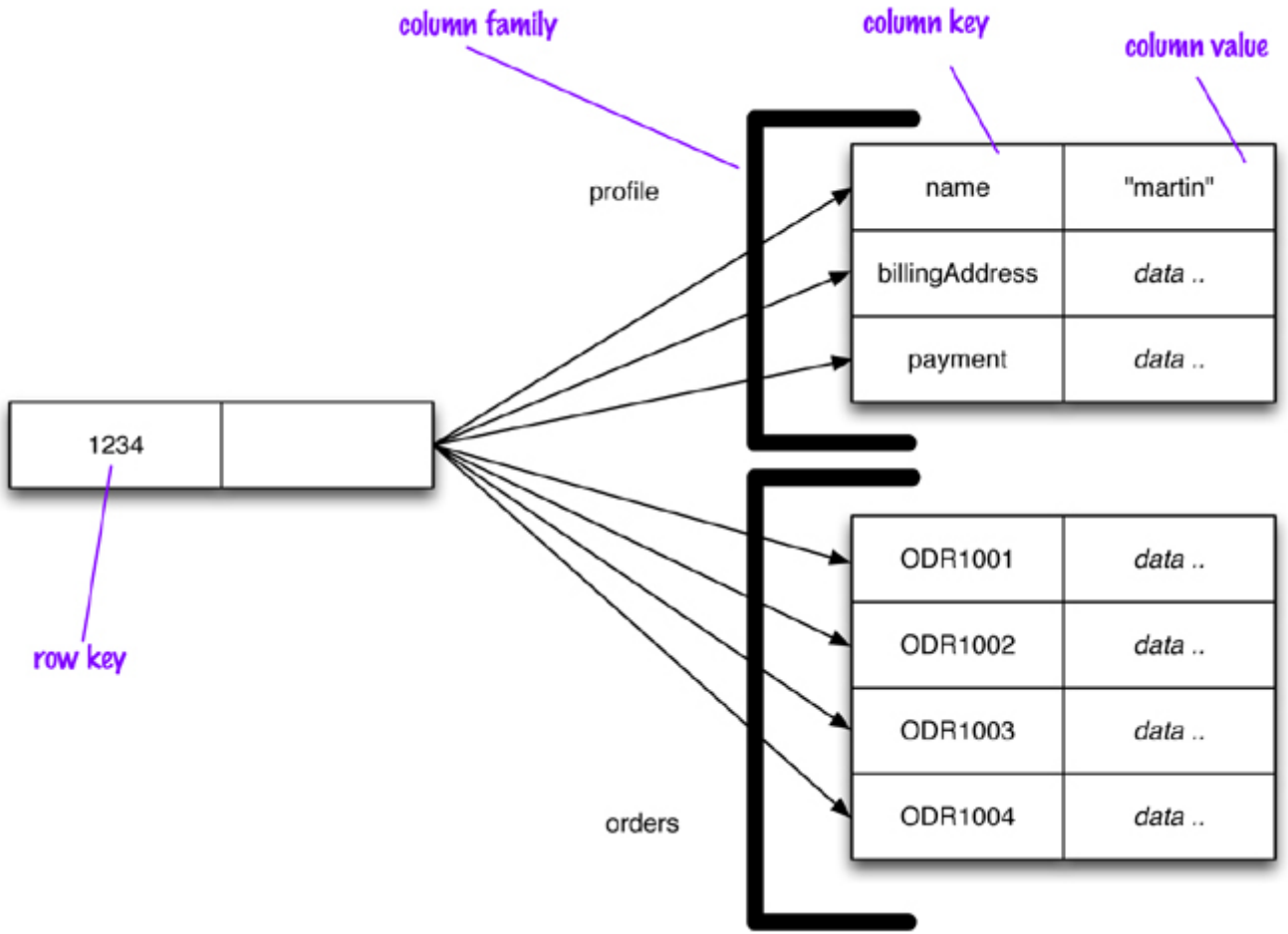
This information can be used for indexing, partitioning, optimization, etc.

Column-Family Stores

Conceptually can be thought of as a map of maps, or a two level aggregate structure.

```
{
    "1234": {
        "accesses": {
            "system1": {
                "duration": 57,
                "tstatmp": "Sat Oct 21 15:15:30 CEST 2017"
            },
            "system2": {
                "tstamp": "Sun Oct 22 18:16:30 CEST 2017"
            }
        },
        "acl": {
            "name": "user1",
            "privs": [
                "root",
                "system"
            ]
        }
    }
}
```

Column-Family Stores



Column-Family Stores

This kind of system was early defined by Google's BigTable

BigTable (https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf)

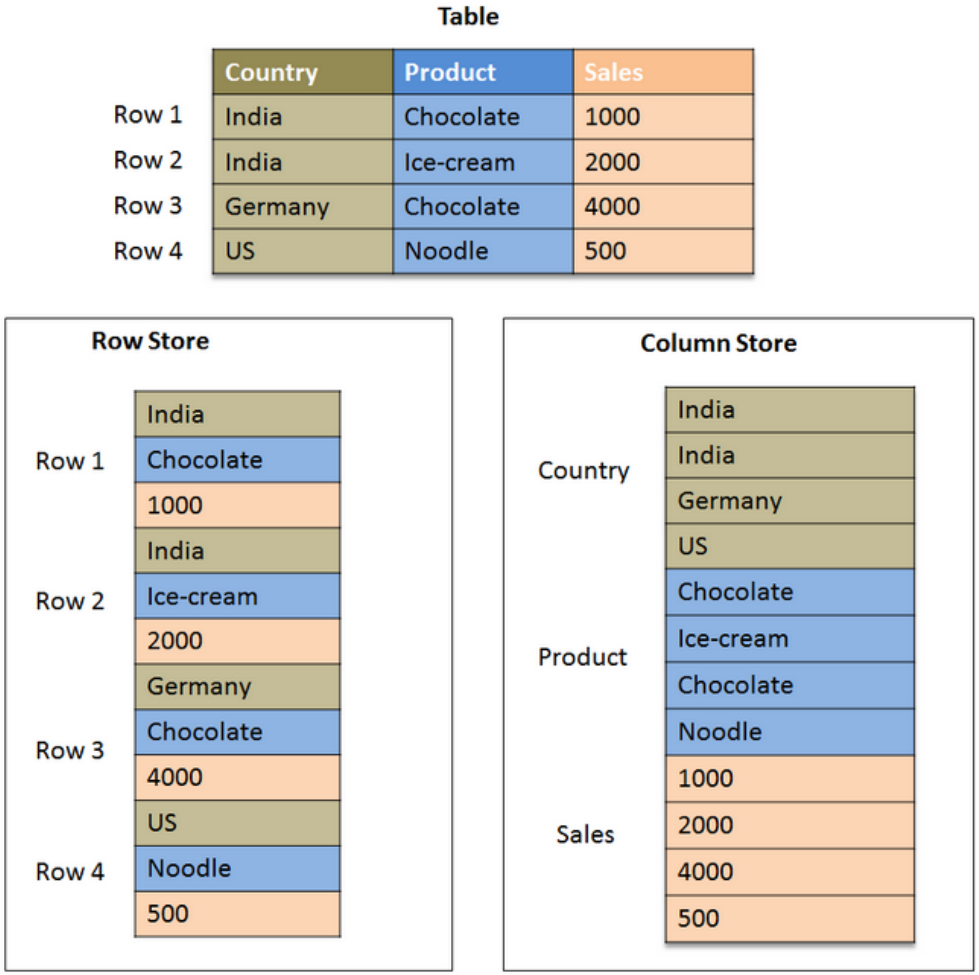
Two ways of looking at it:

- Row oriented: Each row is an aggregate formed by column families representing units of data (profile, history)
- Column oriented: Each column family defines a *record type*. A row can be seen as the *join* of all present records in the column families for a certain common key

Column families can be present or not. No fixed module and support for sparse data sets

Column-Family Stores

Not to be confused with Columnar storage in RDBMS



Graph

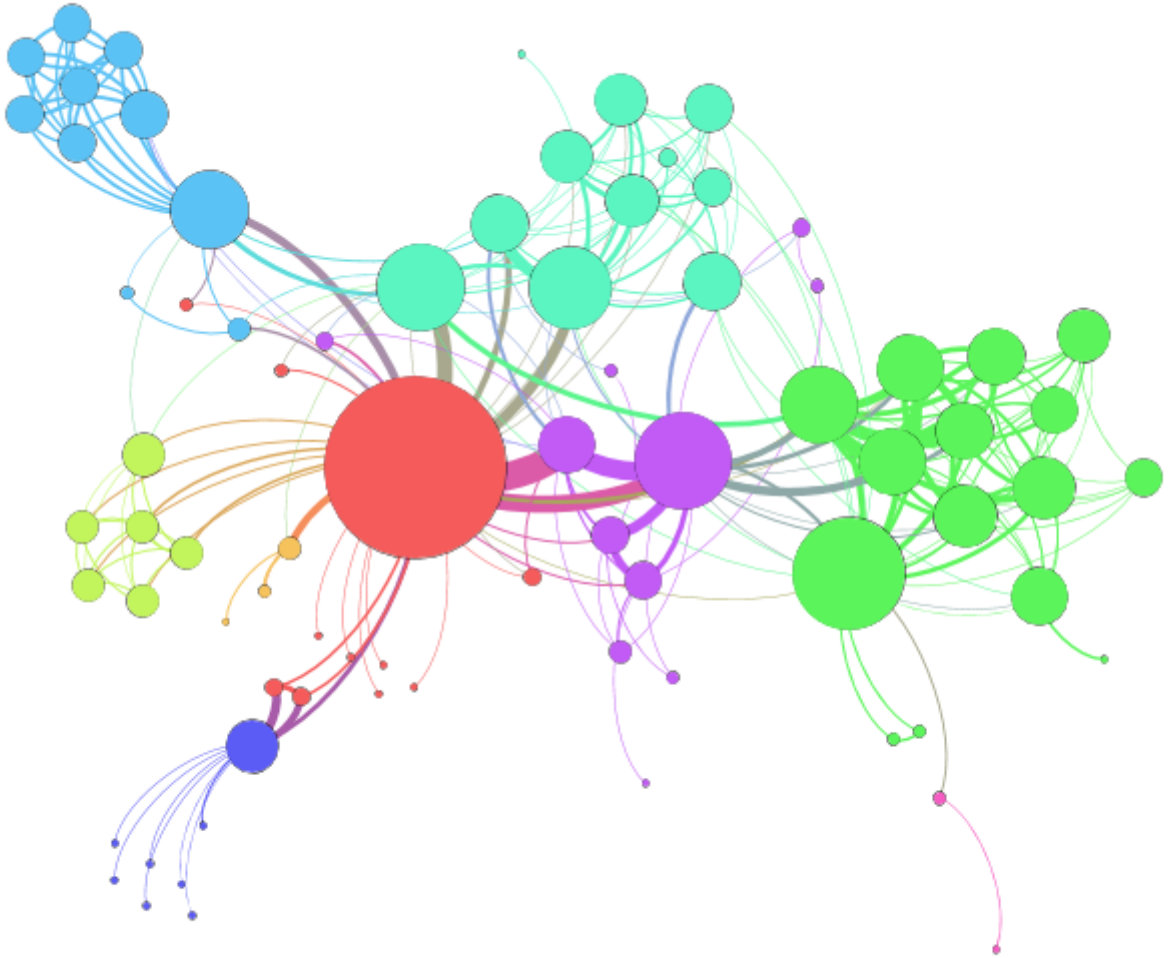
Two main concepts: **Nodes** and **edges**.

Nodes are connected by edges

Both nodes and edges can have attributes

Highly specialized. Very attractive for *social* data.

Graph



Time Series

Specialized systems for time-stamped data, mostly numeric in nature.

Typical Big Data use cases are

- To store metrics from a big number of sources (IoT).
- To store metrics generated at a very high speed (High throughput)

References

```
http://saphanatutorial.com/column-data-storage-and-row-data-storage-sap-hana/
```

Thank you

Ignacio Coterillo
Computer Engineer, CERN
ignacio.coterillo.coz@cern.ch (mailto:ignacio.coterillo.coz@cern.ch)