Aggregate Data Models

Dr. Fabio Fumarola

Agenda

- Data Model Evolution
- Relational Model vs Aggregate Model
- Consequences of Aggregate Models
- Aggregates and Transactions
- Aggregates Models on NoSQL
  - Key-value and Document
  - Column-Family Stores
- Summarizing Aggregate-Oriented databases

Data Model

- A data model is a representation that we use to perceive and manipulate our data.
- It allows us to:
  - Represent the data elements under analysis, and
  - How these are related to each others
- This representation depends on our perception.

Data Model: Database View

- In the database field, it describes how we interact with the data in the database.
- This is distinct from the storage model:
  - It describes how the database stores and manipulate the data internally.
- In an ideal worlds:
  - We should be ignorant of the storage model, but
  - In practice we need at least some insight to achieve a decent performance

Data Models: Example

- A Data model is the model of the specific data in an application
- A developer might point to an entity-relationship diagram and refer it as the data model containing
  - customers,
  - orders and
  - products

Data Model: Definition

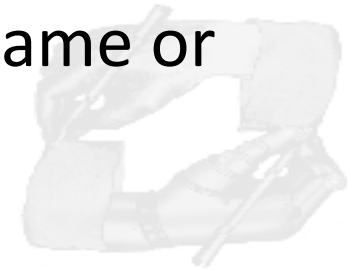
In this course we will refer “data model” as the model by which the database organize data.

It can be more formally defined as meta-model

Last Decades Data Model

- The dominant data model of the last decades what the relational data model.

1. It can be represented as a set of tables.
2. Each table has rows, with each row representing some entity of interest.
3. We describe entities through columns (???)
4. A column may refer to another row in the same or different table (relationship).



NoSQL Data Model

- It moves away from the relational data model
- Each NoSQL database has a different model
  - Key-value,
  - Document,
  - Column-family,
  - Graph, and
  - Sparse (Index based)
- Of these, the first three share a common characteristic (Aggregate Orientation).

RELATIONAL MODEL
VS
AGGREGATE MODEL

Relational Model

- The relational model takes the information that we want to store and divides it into tuples (rows).
- However, a tuple is a limited data structure.
- It captures a set of values.
- So, we can't nest one tuple within another to get nested records.
- Nor we can put a list of values or tuple within another.

Relational Model

- This simplicity characterize the relational model
- It allows us to think on data manipulation as operation that have:
  - As input tuples, and
  - Return tuples
- Aggregate orientation takes a different approach.

Aggregate Model

- It recognizes that, you want to operate on data unit having a more complex structure than a set of tuples.
- We can think on term of complex record that allows:
  - List,
  - Map,
  - And other data structures to be nested inside it
- Key-Value, document, and column-family databases uses this complex structure.

Aggregate Model

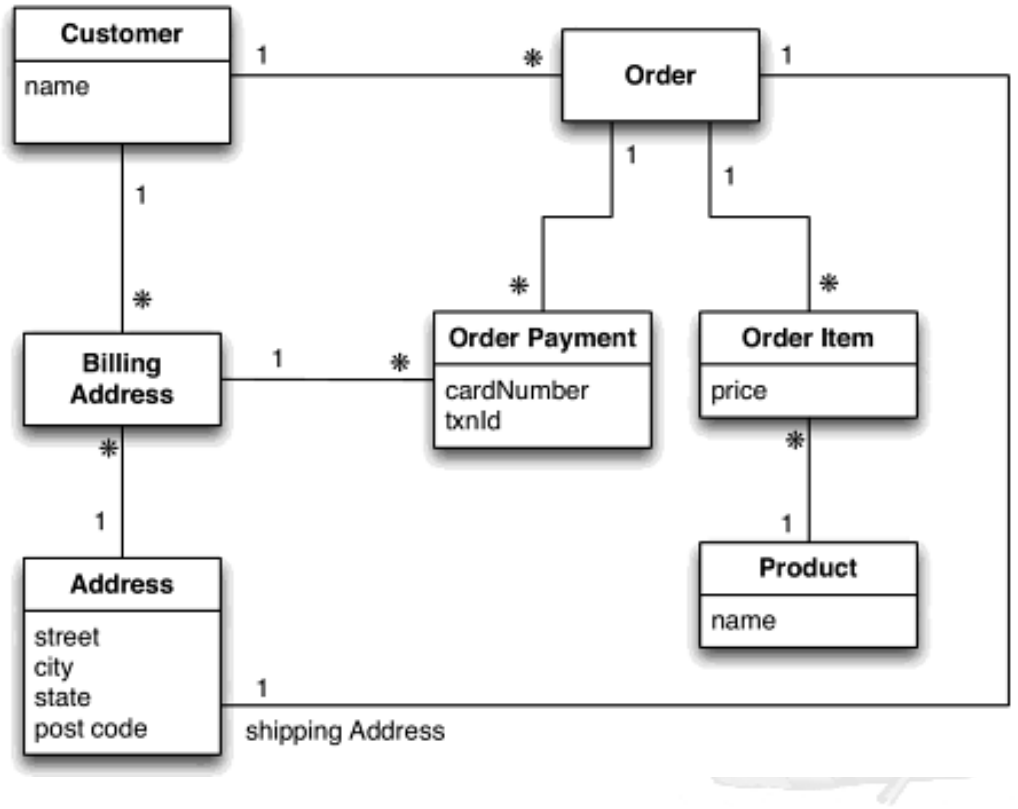
- Aggregate is a term coming from Domain-Driven Design [Evans03]
  - An aggregate is a collection of related objects that we wish to treat as a unit for data manipulation, management a consistency.
- We like to update aggregates with atomic operation
- We like to communicate with our data storage in terms of aggregates

Aggregate Models

- This definition matches really with how key-value, document, and column-family databases works.
- With aggregates we can easier work on a cluster, since they are unit for replication and sharding.
- Aggregates are also easier for application programmer to work since solve the impedance mismatch problem of relational databases.

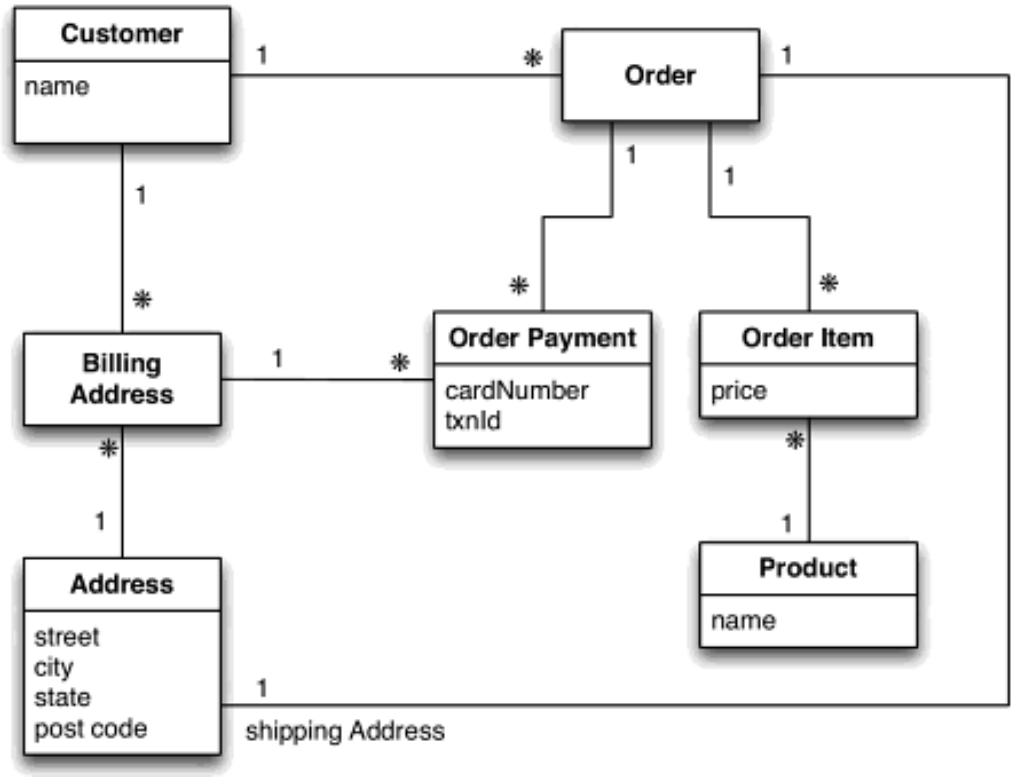
Example of Relational Model

- Assume we are building an e-commerce website;
- We have to store information about: users, products, orders, shipping addresses, billing addresses, and payment data.



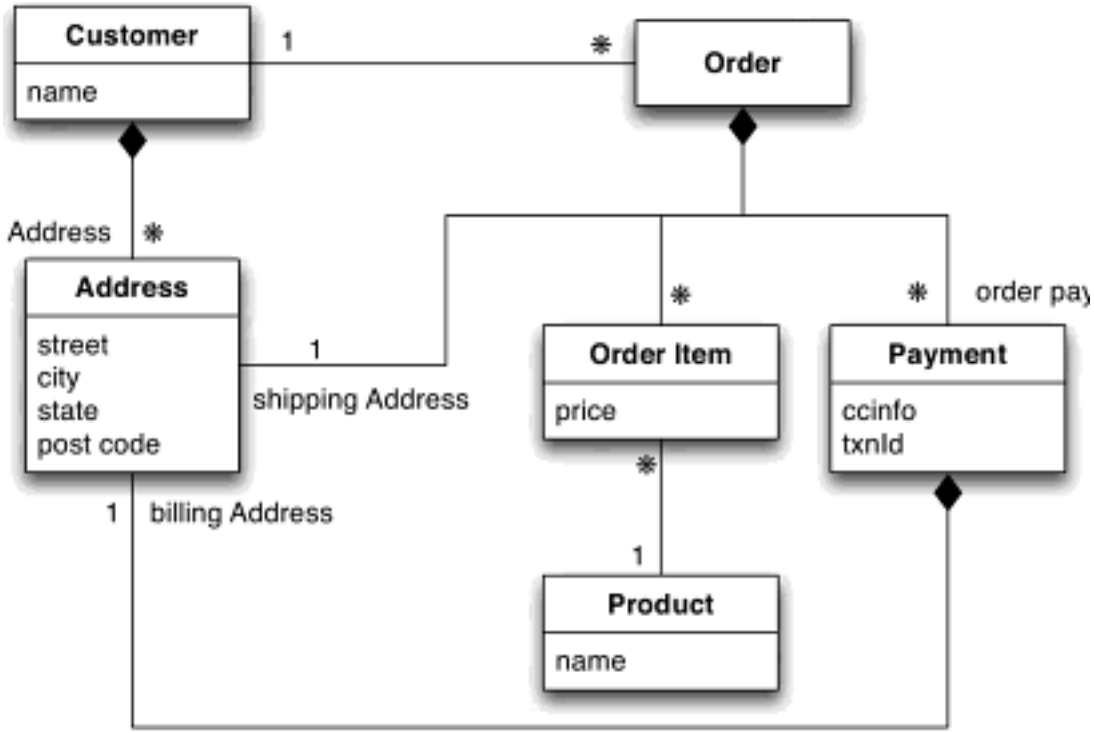
Example of Relational Model

- As we are good relational soldier:
  - Everything is normalized
  - No data is repeated in multiple tables.
  - We have referential integrity



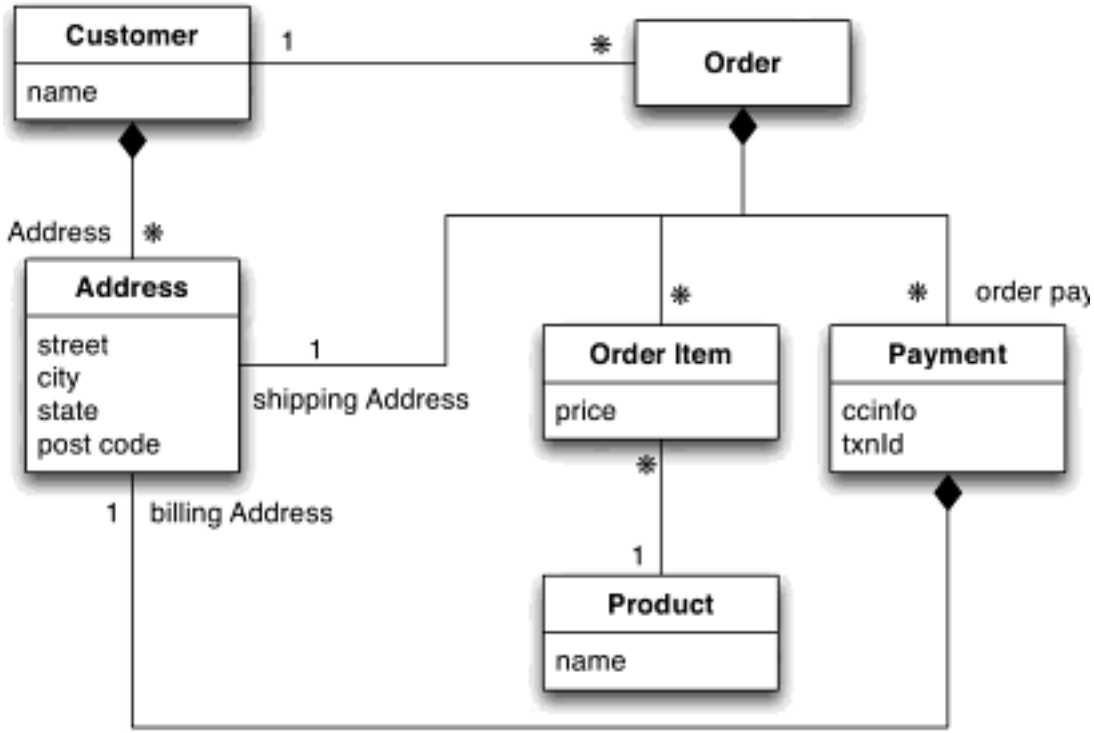
Example of Aggregate Model

- We have two aggregates:
  - Customers and
  - Orders
- We us the black-diamond composition to show how data fits into the aggregate structure



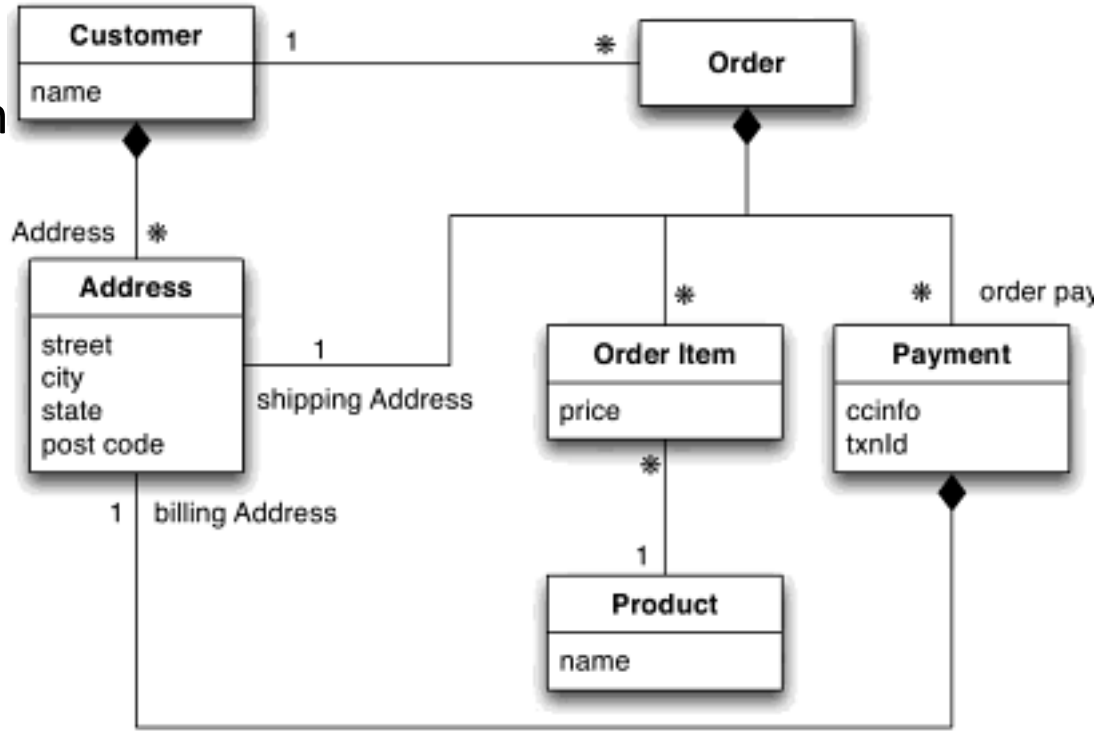
Example of Aggregate Model

- The customer contains a list of billing addresses;
- The order contains a list of:
  - order items,
  - a shipping address,
  - and payments
- The payment itself contains a billing address for that payment

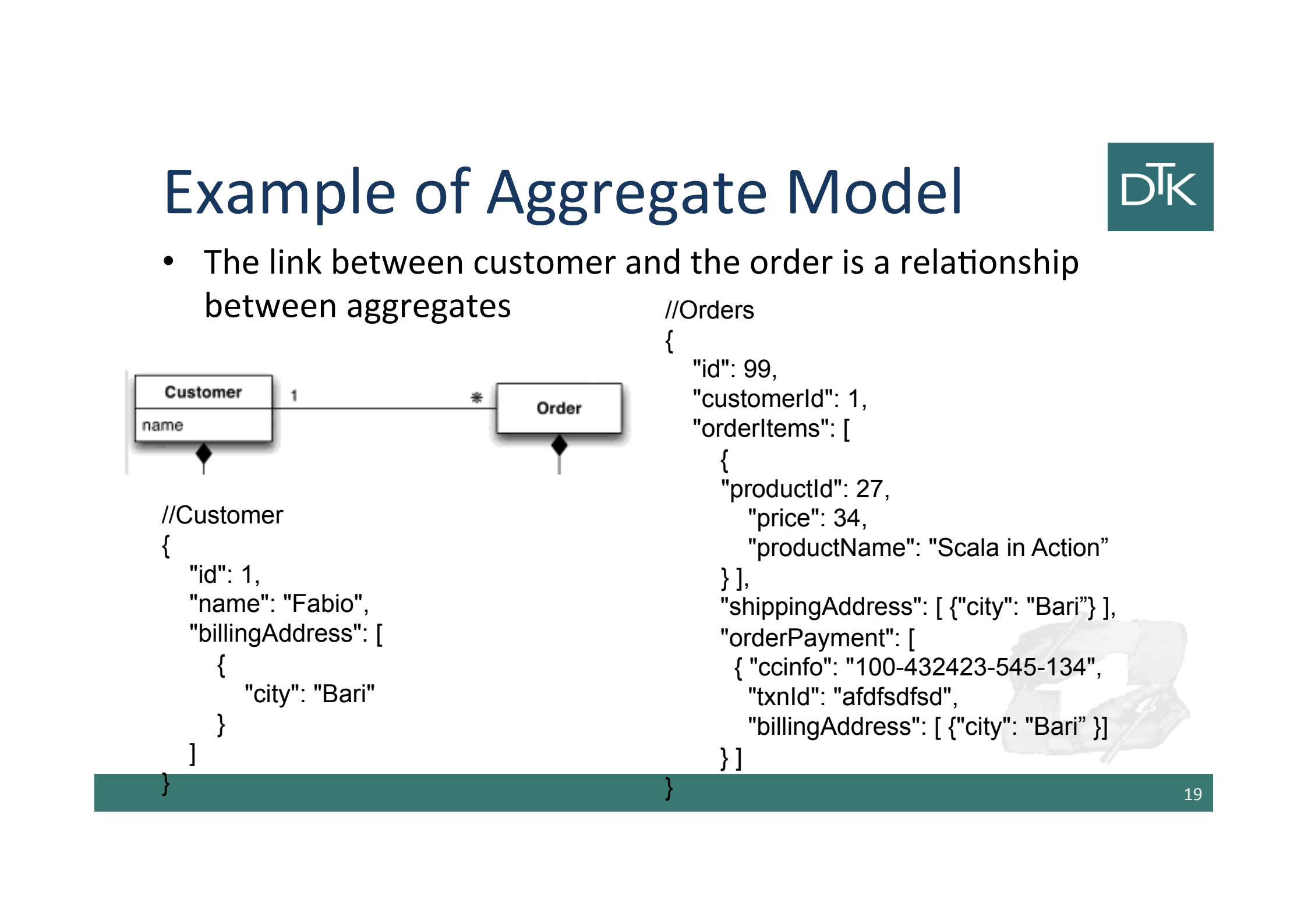


Example of Aggregate Model

- A single address appears 3 times, but instead of using an id it is copied each time
- This fits a domain where we don't want shipping, payment and billing address to change
- What is the difference w.r.t a relational representation?



Example of Aggregate Model



- The link between customer and the order is a relationship between aggregates


```
//Customer
{
    "id": 1,
    "name": "Fabio",
    "billingAddress": [
        {
            "city": "Bari"
        }
    ]
}
```

```
//Orders
{
    "id": 99,
    "customerId": 1,
    "orderItems": [
        {
        "productId": 27,
            "price": 34,
            "productName": "Scala in Action”
        } ],
        "shippingAddress": [ {"city": "Bari”} ],
        "orderPayment": [
          { "ccinfo": "100-432423-545-134",
            "txnId": "afdfsdfsd",
            "billingAddress": [ {"city": "Bari” }]
        } ]
}
```

Orders Details

```
//Orders
{
   "id": 99,
   "customerId": 1,
   "orderItems": [
      {
         "productId": 27,
         "price": 34,
         "productName": "Scala in Action”
      } ],
      "shippingAddress": [ {"city": "Bari”} ],
      "orderPayment": [
       { "ccinfo": "100-432423-545-134",
         "txnId": "afdfsdfsd",
         "billingAddress": [ {"city": "Bari” }]
      } ]
}
```

- There is the customer id
- The product name is a part of the ordered Items
- The product id is part of the ordered items
- The address is stored several times

Example: Rationale

- We aggregate to minimize the number of aggregates we access during data interaction
- The important think to notice is that,
  1. We have to think about accessing that data
  2. We make this part of our thinking when developing the application data model
- We could drawn our aggregate differently, but it really depends on the “data accessing models”

Example: Rationale

- Like most thinks in modeling, there is no an universal answer on how we draw aggregate boundaries.
- It depends on how we have to manipulate our data
  1. If we tend to access a customer with all its orders at once, then we should prefer a single aggregate.
  2. If we tend to access a single order at time, then we should prefer having separate aggregates for each order.

CONSEQUENCES OF AGGREGATE MODELS

No Distributable Storage

- Relational mapping can captures data elements and their relationship well.
- It does not need any notion of aggregate entity, because it uses foreign key relationship.
- But we cannot distinguish for a relationship that represent aggregations from those that don't.
- As result we cannot take advantage of that knowledge to store and distribute our data.

Marking Aggregate Tools

- Many data modeling techniques provides way to mark aggregate structures in relational models
- However, they do not provide semantic that helps in distinguish relationships
- When working with aggregate-oriented databases, we have a clear views of the semantic of the data.
- We can focus on the unit of interaction with the data storage.

Aggregate Ignorant

- Relational database are **aggregate-ignorant**, since they don't have concept of aggregate
- Also graph database are aggregate-ignorant.
- This is not always bad.
- It domains where it is difficult to draw aggregate boundaries aggregate-ignorant databases are useful.

Aggregate and Operations

- An order is a good aggregate when:
  - A customer is making and reviewing an order, and
  - When the retailer is processing orders
- However, when the retailer want to analyze its product sales over the last months, then aggregate are trouble.
- We need to analyze each aggregate to extract sales history.

Aggregate and Operations

- Aggregate may help in some operation and not in others.
- In cases where there is not a clear view aggregate-ignorant database are the best option.
- But, remember the point that drove us to aggregate models (cluster distribution).
- Running databases on a cluster is need when dealing with huge quantities of data.

Running on a Cluster

- It gives several advantages on computation power and data distribution
- However, it requires to minimize the number of nodes to query when gathering data
- By explicitly including aggregates, we give the database an important of which information should be stored together
- But, still we have the problem on querying historical data, do we have any solution?

AGGREGATES AND TRANSACTIONS

ACID transactions

- Relational database allow us to manipulate any combination of rows from any table in a single transaction.
- ACID transactions:
  - Atomic,
  - Consistent,
  - Isolated, and
  - Durable

  have the main point in Atomicity.

Atomicity & RDBMS

- Many rows spanning many tables are updated into an Atomic operation
- It may succeeded or failed entirely
- Concurrently operations are isolated and we cannot see partial updates
- However relational database still fail.

Atomicity & NoSQL

- NoSQL don't support Atomicity that spans multiple aggregates.
- This means that if we need to update multiple aggregates we have to manage that in the application code.
- Thus the Atomicity is one of the consideration for deciding how to divide up our data into aggregates

Key-value and Document

AGGREGATES MODELS ON NOSQL

Key-Value and Document

- Key-value and Document databases are strongly aggregate-oriented.
- Both of these types of databases consists of lot of aggregates with a key used to get the data.
- The two type of databases differ in that:
  - In a key-value stores the aggregate is opaque (Blob)
  - In a document database we can see a structure in the aggregate.

Key-Value and Document

- The advantage of key-value is that we can store any type of object
- The database may impose some size limit, but we have freedom
- A document store imposes limits on what we can place in it, defining a structure on the data.
  - In return we a language to query documents.

Key-Value and Document

- With a key-value we can only access by its key
- With document:
  - We can submit queries based on fields,
  - We can retrieve part of the aggregate, and
  - The database can create index based on the fields of the aggregate.
- But in practice they are used differently

Key-Value and Document

- People use document as key-value
- Riak (key-value) allows you to add metadata to aggregates for indexing
- Redis allows you to break aggregates into lists, sets or maps.
- You can support queries by integrating search tools like Solr. (Riak include solr for searching data stored as XML or JSON).

Key-Value and Document

Despite this the general distinction still holds.

- With key-value databases we expect aggregates using a key
- With document databases, we mostly expect to submit some form of query on the internal structure of the documents.

Column-Family Stores

AGGREGATES MODELS ON NOSQL

Column-Family Stores

- One of the most influential NoSQL databases was Google's BigTable [Chang et al.]
- Its name derives from its structure composed by sparse columns and no schema.
- We don't have to think to this structure as a table, but to a two-level map.
- BigTable models influenced the design the open source HBase and Cassandra.

Column-Family Stores

- These BigTable-style data model are referred to as column stores.
- Pre-NoSQL column stores like C-Store used SQL and the relational model.
- What make NoSQL columns store different is how physically they store data.
- Most databases has rows as unit of storage, which helps in writing performances

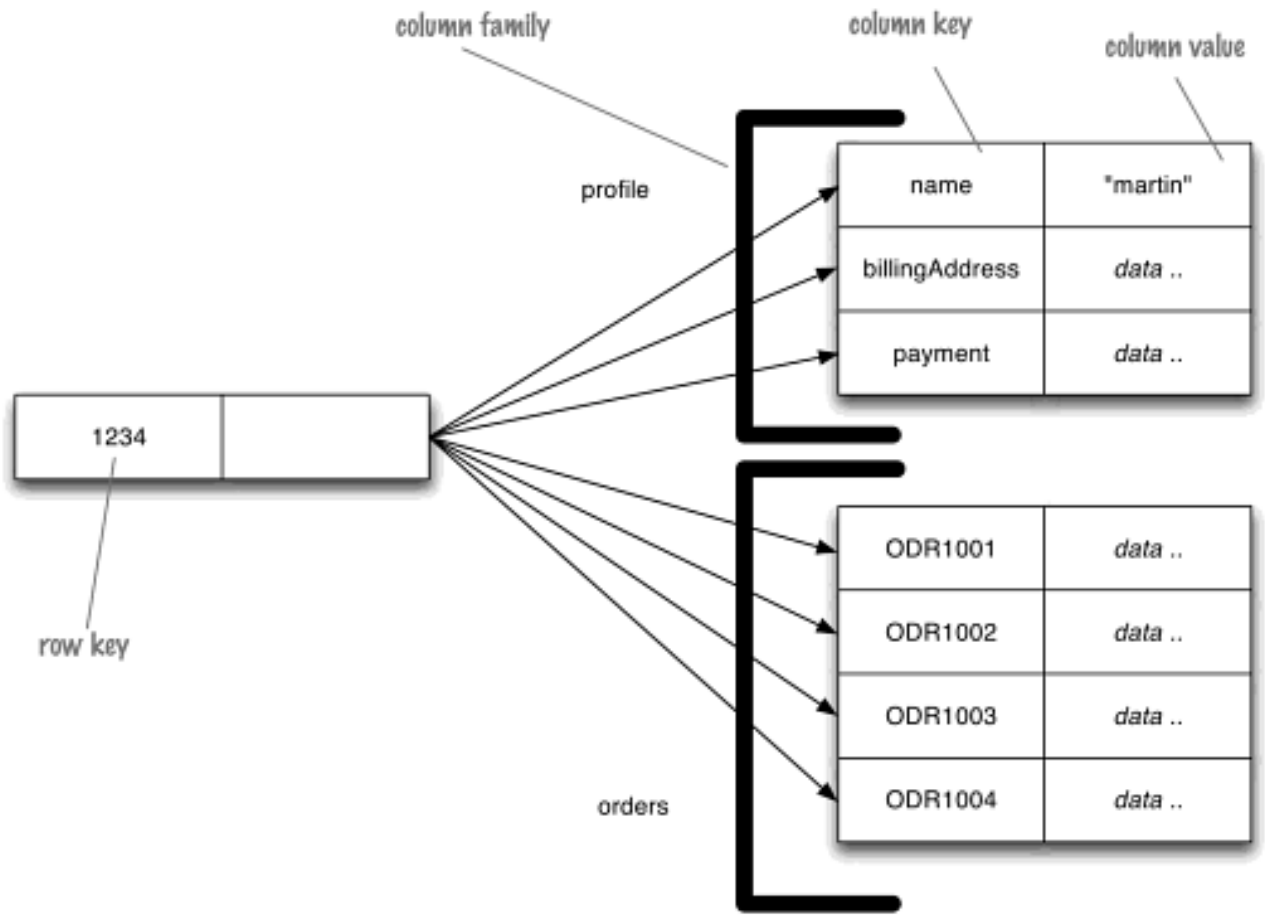
Column-Family Stores

- However, there are many scenarios where:
  - Write are rares, but
  - You need to read a few columns of many rows at once
- In this situations, it's better to store groups of columns for all rows as the basic storage unit.
- These kind of databases are called column stores or column-family databases

Column-Family Stores

- Column-family databases have a two-level aggregate structure.
- Similarly to key-value the first key is the row identifier.
- The difference is that retrieving a key return a Map of more detailed values.
- These second-level values are defined to as columns.
- Fixing a row we can access to all the column-families or to a particular element.

Example of Column Model

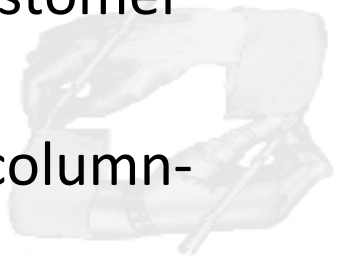


Column-Family Stores

- They organize their columns into families.
- Each column is a part of a family, and column family acts as unit of access.
- Then the data for a particular column family are accessed together.

Column-Family Stores: How to structure data

- In row-oriented:
  - each row is an aggregate (For example the customer with id 456),
  - with column families representing useful chinks of data (profile, order history) within that aggregate
- In column-oriented:
  - each column family defines a record type (e.g. customer profiles) with rows for each of the records.
  - You can this of a row as the join of records in all column-families

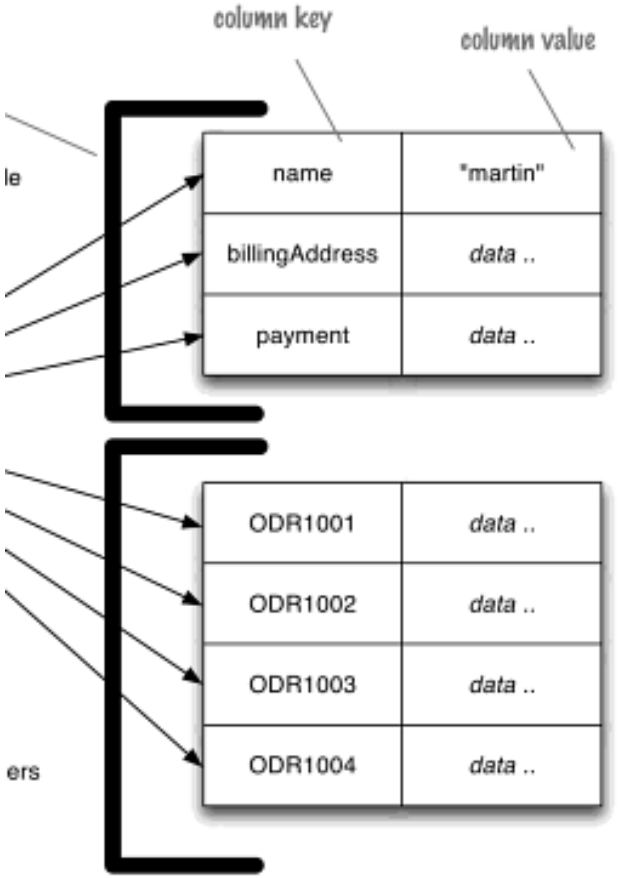


Column Family: Storage Insights

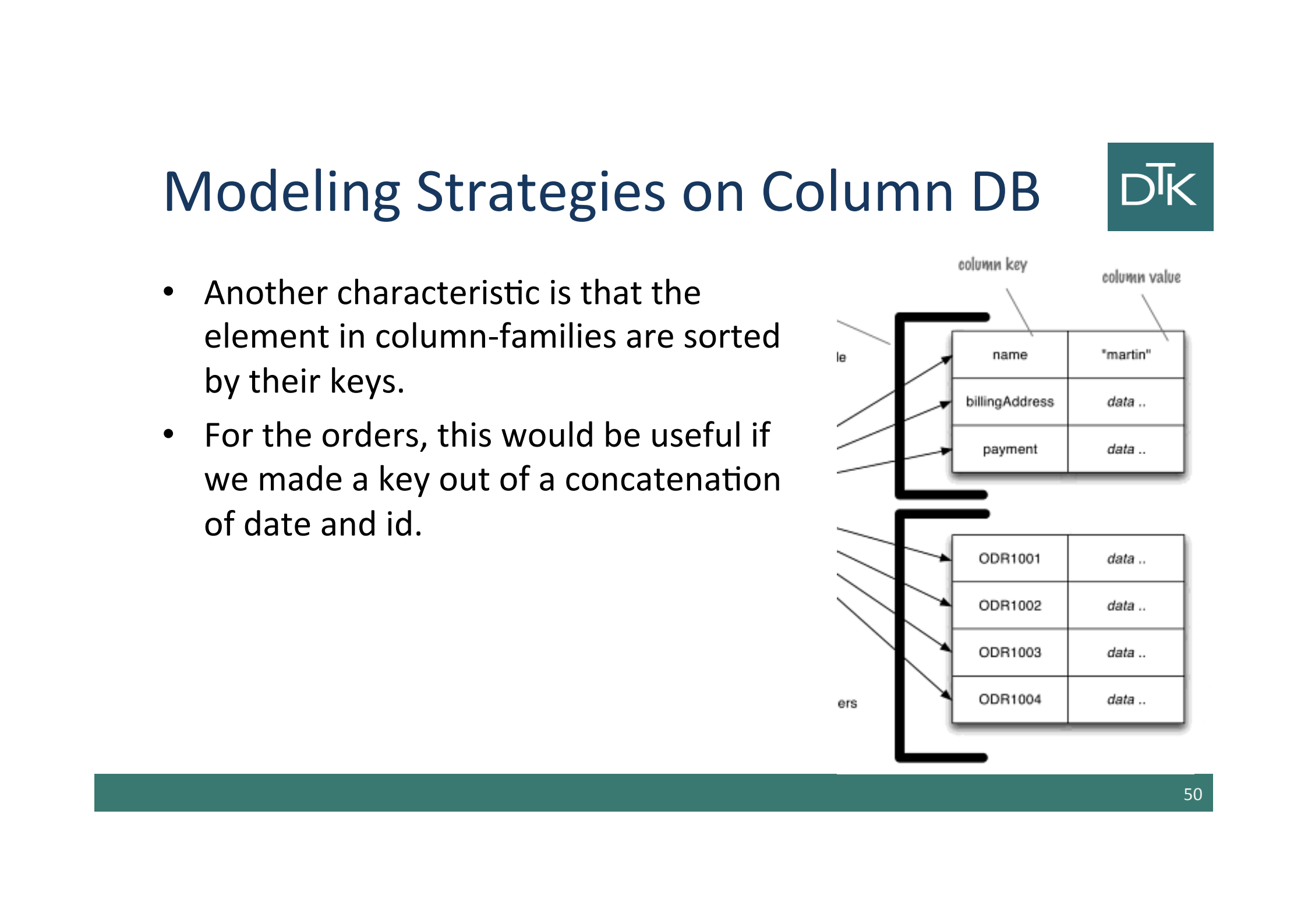
- Since the database knows about column grouping, it uses this information for storage and access behavior.
- If we consider Document store, also if a document as an array of elements the document is a single unit of storage.
- Column families give a two-level dimension of stored data.

Modeling Strategies on Column DB

- We can model list based elements as a column-family.
- Thus we can have column-families with many elements and other with few element.
- This may let to high fragmentation on data stored.
- In Cassandra this problem is faced by defining: Skinny and Wide rows.



Modeling Strategies on Column DB



- Another characteristic is that the element in column-families are sorted by their keys.
- For the orders, this would be useful if we made a key out of a concatenation of date and id.

SUMMARIZING AGGREGATE-ORIENTED DATABASES

Key Points

- All share the notion of an aggregated indexed by a key.
- The key can be used for lookup
- The aggregate is central to running on a cluster.
- The aggregates acts as the atomic unit for updates providing transaction on aggregates.
- Key-value treats the values as Blob

Key Points

- The document model makes the aggregate transparent for queries, but the document is treated as single unit of storage.
- Column-family models divide aggregates into column families, allowing the database to treat them as units of data in the aggregates.
- This imposes some structure but allows the database to take advantage of the structure to improve its accessibility.