NoSQL design

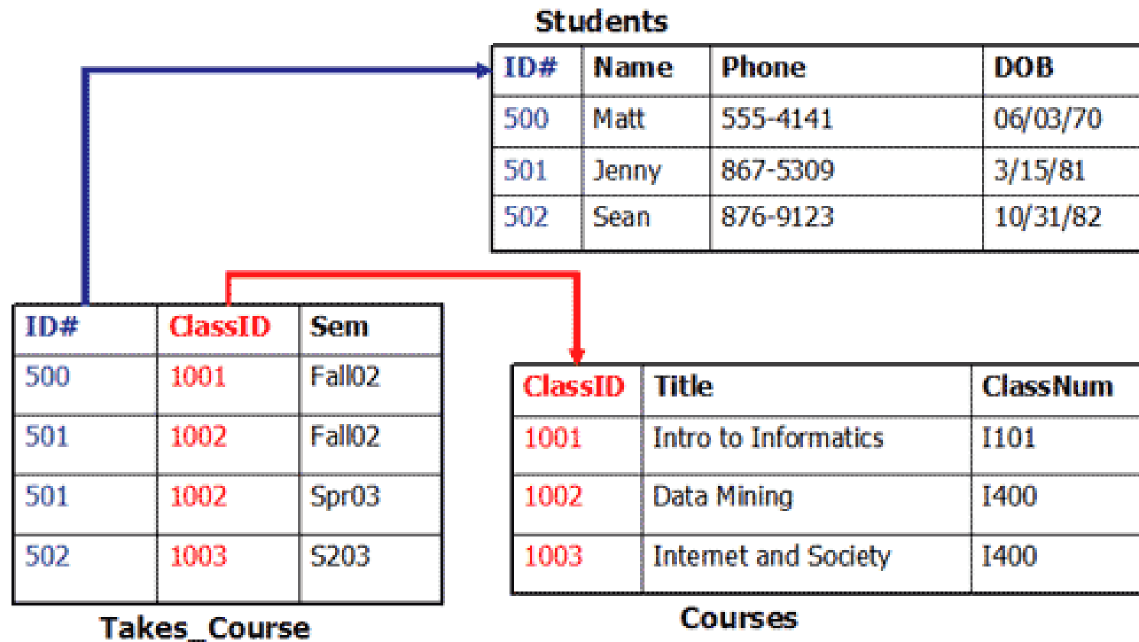
Data Management and Visualization

Politecnico di Torino

Relational Database Management Systems

1. Data structures are broken into the **smallest units**
   - **normalization** of database schema
     - because the data structure is known in advance
     - and users/applications query the data in different ways
   - database **schema** is rigid
2. Queries merge the data from different tables (**joins**)
3. Write operations are **simple**, search can be **slower**
4. Strong guarantees for **transactional** processing

RDBMS Example



Students

ID#	Name	Phone	DOB
500	Matt	555-4141	06/03/70
501	Jenny	867-5309	3/15/81
502	Sean	876-9123	10/31/82

ID#	ClassID	Sem
500	1001	Fall02
501	1002	Fall02
501	1002	Spr03
502	1003	S203

Takes_Course

ClassID	Title	ClassNum
1001	Intro to Informatics	I101
1002	Data Mining	I400
1003	Internet and Society	I400

Courses

SELECT Name
FROM Students S, Takes_Course T
WHERE S.ID=T.ID AND ClassID = 1001

source: https://github.com/talhafazal/DataBase/wiki/Home-Work-%23-3-Relational-Data-vs-Non-Relational-Databases

From RDBMS to NoSQL

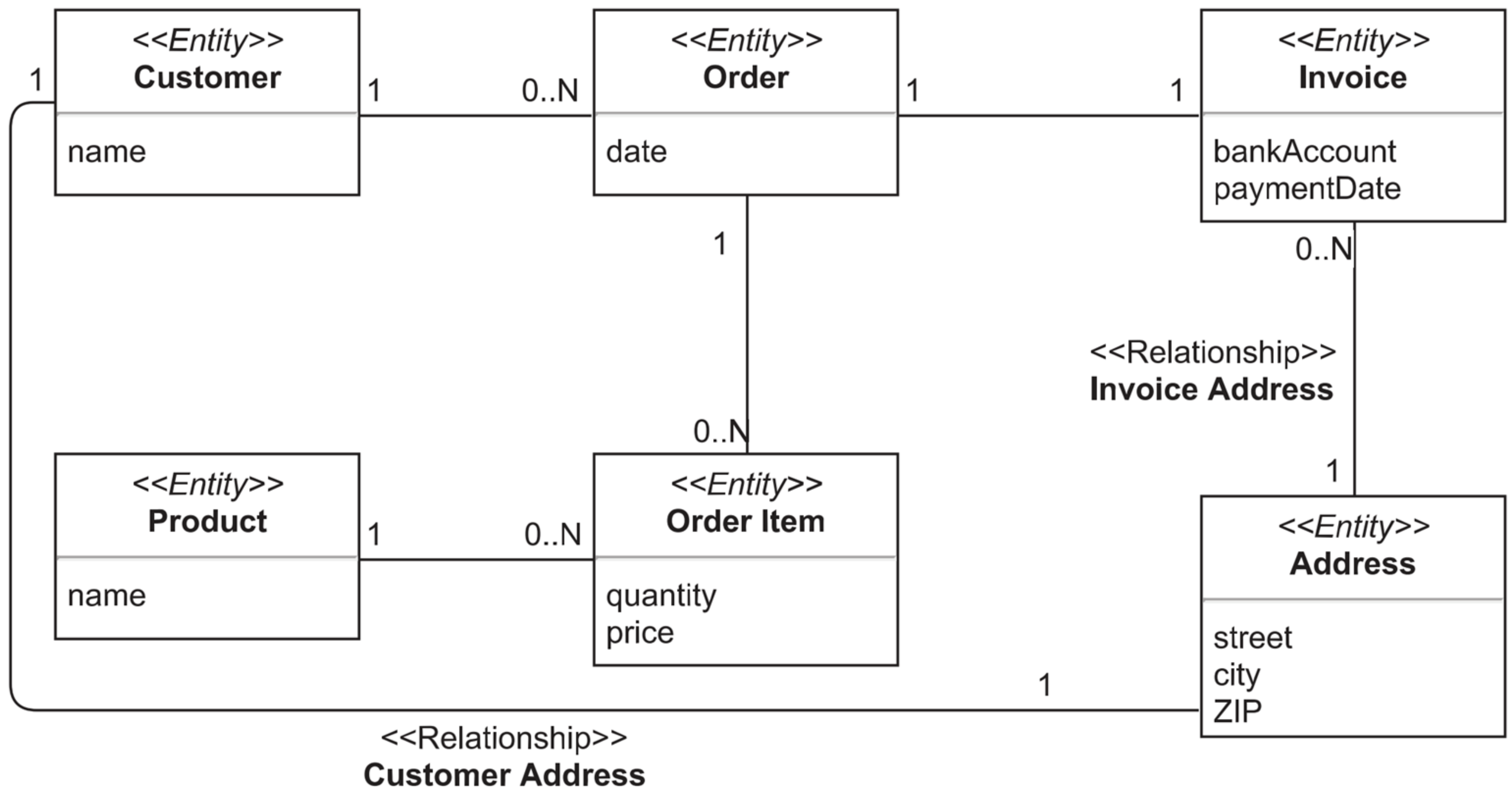
Efficient implementations of table **joins** and of **transactional** processing require a **centralized** system.

NoSQL Databases

- Database **schema** tailored for specific application
  - **keep together data pieces that are often accessed together**
- Write operations might be slower but read is fast
- Weaker consistency guarantees

Result: **efficiency** and horizontal **scalability**

Example: UML Model



source: Holubová, Kosek, Minařík, Novák. Big Data a NoSQL databáze. 2015.

Example: Relational Model

Customer

customerID

name

addressID (FK)

Order

orderNumber

date

customerID (FK)

Invoice

invoiceID

bankAccount

paymentDate

addressID (FK)

orderNumber (FK)

Product

productID

name

OrderItem

orderNumber (FK)

productID (FK)

quantity

price

Address

addressID

street

city

ZIP

source: Holubová, Kosek, Minařík, Novák. Big Data a NoSQL databáze. 2015.

Data Model

- The model by which the database organizes data
- Each NoSQL DB type has a different data model
  - Key-value, document, column-family, graph
  - The first three are oriented on **aggregates**

Data Models

- (**Logical**) Data model
  - It is a set of constructs for representing the information
- **Storage** data model
  - How the DBMS stores and manipulates the data internally
- A data model is usually **independent** of the storage model
  - In practice we need at least some insight to achieve good performances

Data Models

- Data model for relational systems
  - Relational model
    - tables, columns and rows
- Data models for NoSQL systems
  - **Aggregate** models
    - key-value based model
    - **Document** based model
    - column-family based model
  - Graph-based models

Relational Model - limitations

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

Aggregate Models

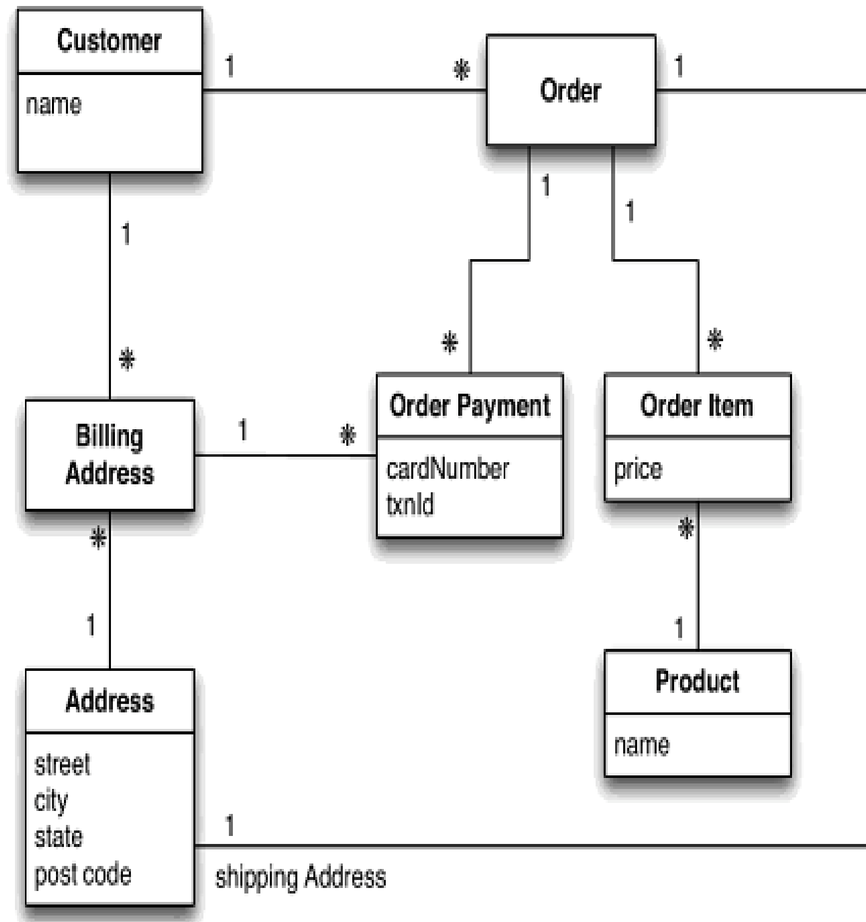
- Data are modeled as units that have a **complex structure**
  - A more complex structure than just a set of tuples
  - Complex records with
    - Simple fields
    - Lists
    - Maps
    - Records nested inside other records

Aggregate Models

- Aggregate is a term coming from Domain-Driven Design
  - An aggregate is **a collection of related objects that we wish to treat as a unit f**or data manipulation, management, and consistency
- We work with data in terms of aggregates
- We like to update aggregates with **atomic** operations
- With aggregates we can easier work on a cluster
  - They are "independent" units
- Aggregates are also easier for application programmer to work since solve the **impedance mismatch** problem of relational databases
  - There is a strict "matching" between the objects used inside programs and the "units/complex records" stored in the databases

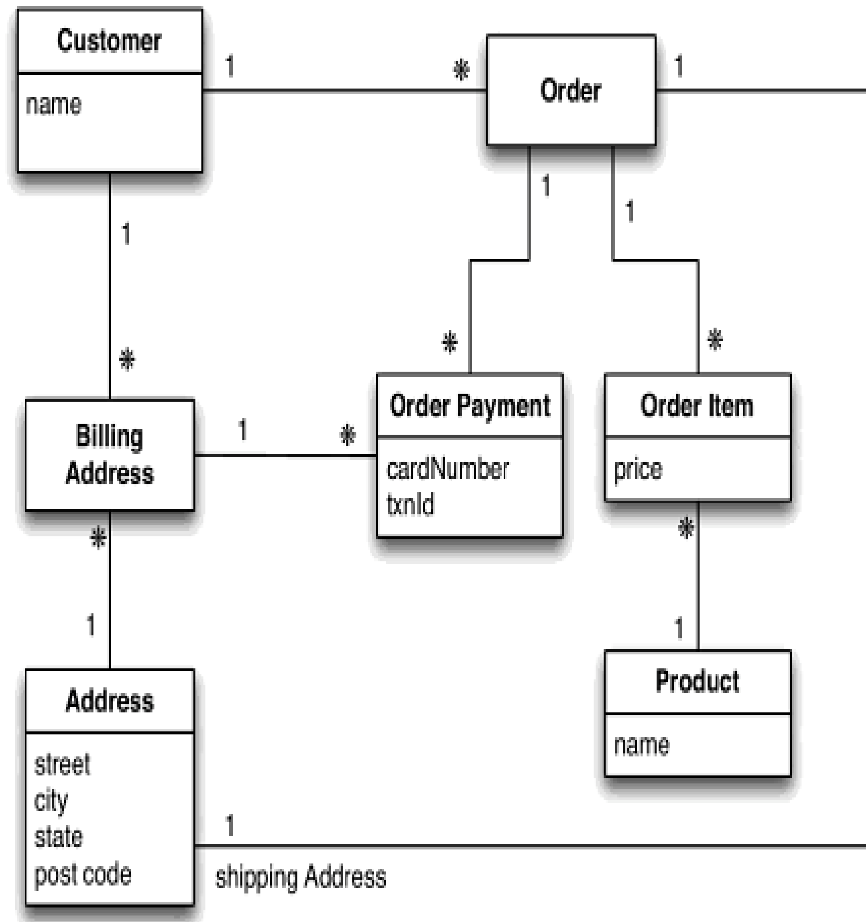
Example

- We are building an e-commerce website
- Stored information
  - Users
  - Products
  - Orders
  - Shipping addresses
  - Billing addresses
  - Payment data



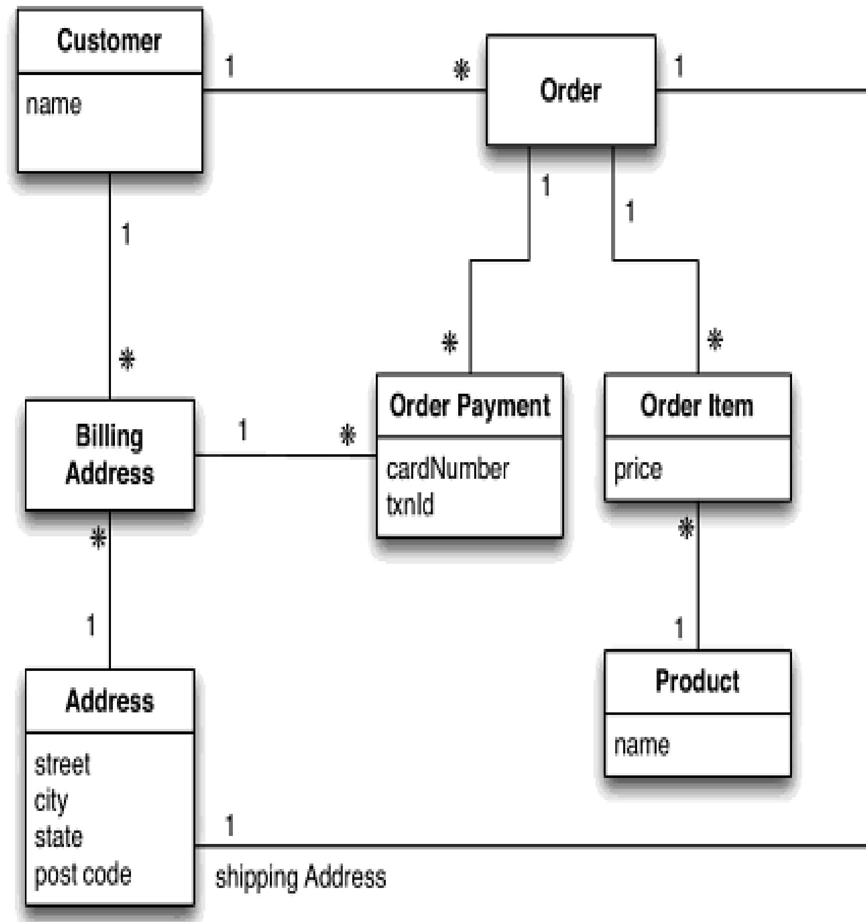
Example of Relational Model

- Relational model
  - Everything is normalized
  - No data is repeated in multiple tables
  - We have referential integrity



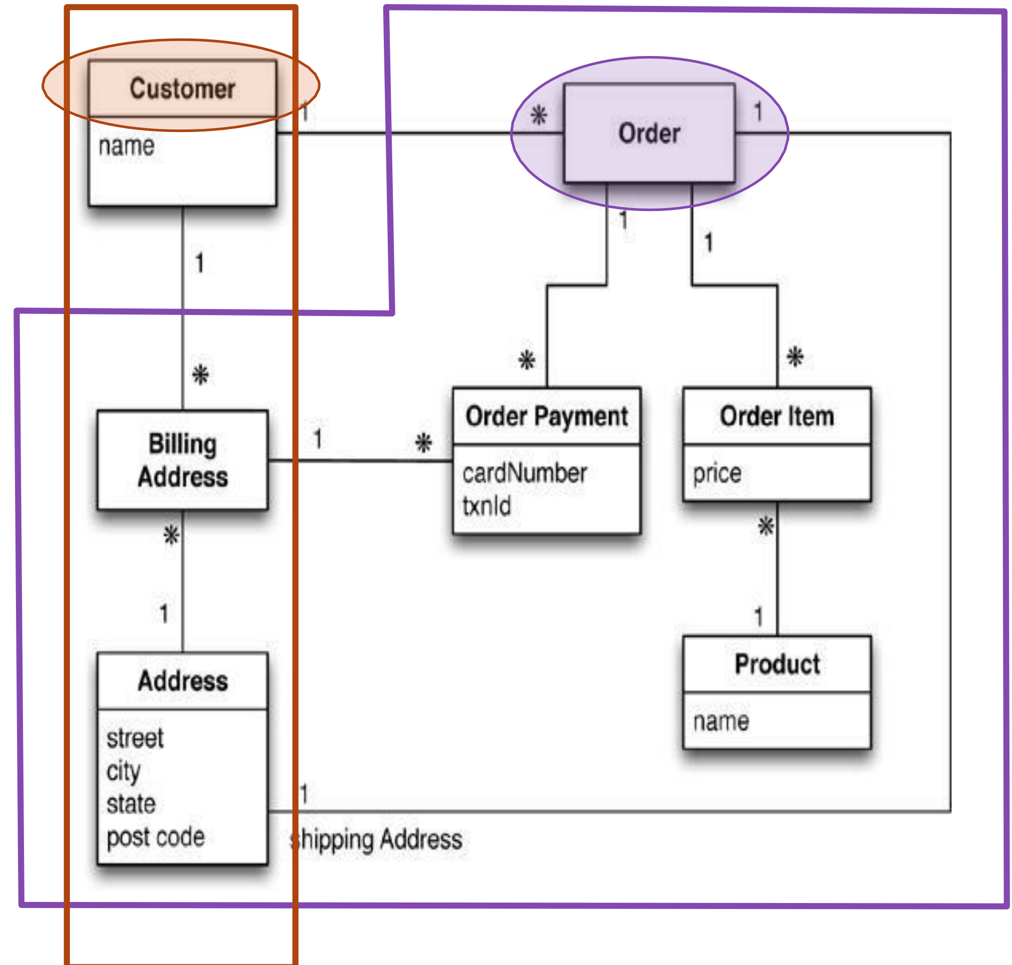
Exercise: domain-driven data access requests

- The billing address of customers is typically accessed together with the customer name
- When an order is accessed, all its items with the corresponding product names, and the payment information with the billing address of the paying customer are requested.
- The shipping address of the order is requested when accessing an order.



Aggregate Model

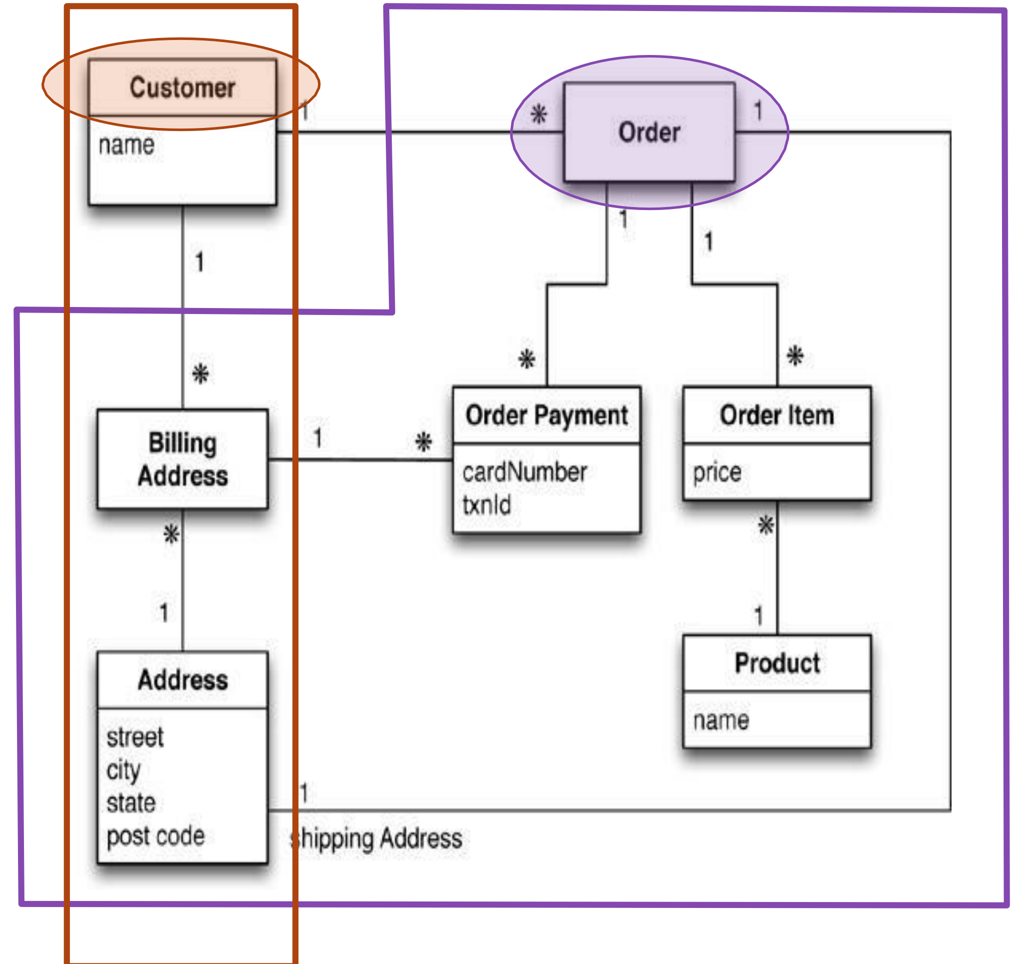
- We have **two aggregates** in this example model
  - Customers
  - Orders



Solution

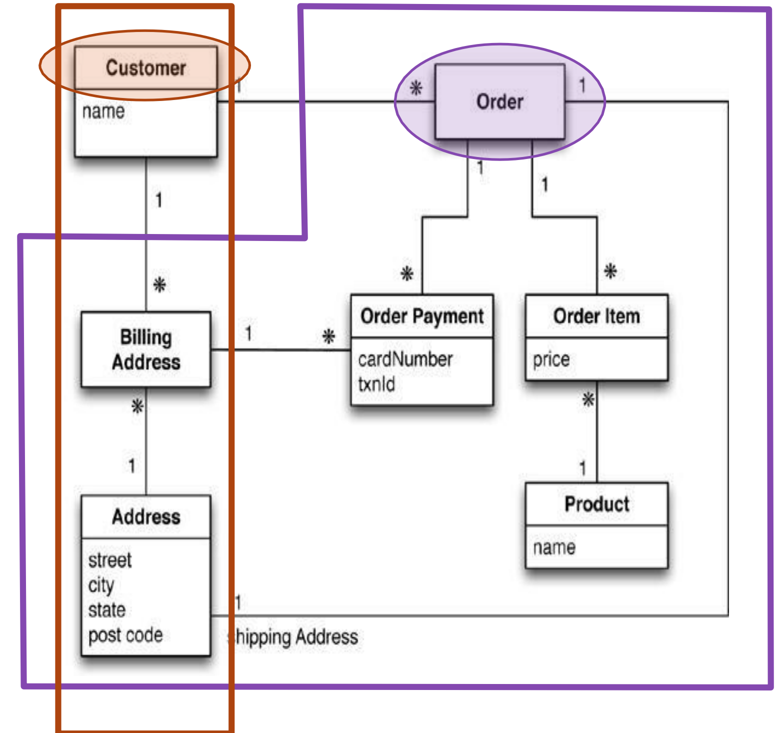
```
// (Single) Customer
{
    "id": 1,
    "name": "Fabio",
    "billingAddress": [
        {
            "street": "via Eco",
            "city": "Bari",
            "state": "IT",
            …
        },

        {
            "street": "via Ugo",
            "city": "Torino",
            "state": "IT",
            …
        },
    ]
}
```



Solution

```
//(Single) Order
{
    "id": 99,
    "customerId": 1,
    "orderItems": [
      {"productId": 27,
       "price": 34,
       "productName": “Data Mngm book”
       }, {…} ],
     "shippingAddress": {"city": "Bari”, … },
     "orderPayment": [
          { "ccinfo": "100-432423-545-134",
            "txnId": "afdfsdfsd",
            "billingAddress":
                  {"city": "Bari”, … }
           }, {…} ]
}
```



Aggregate implementation

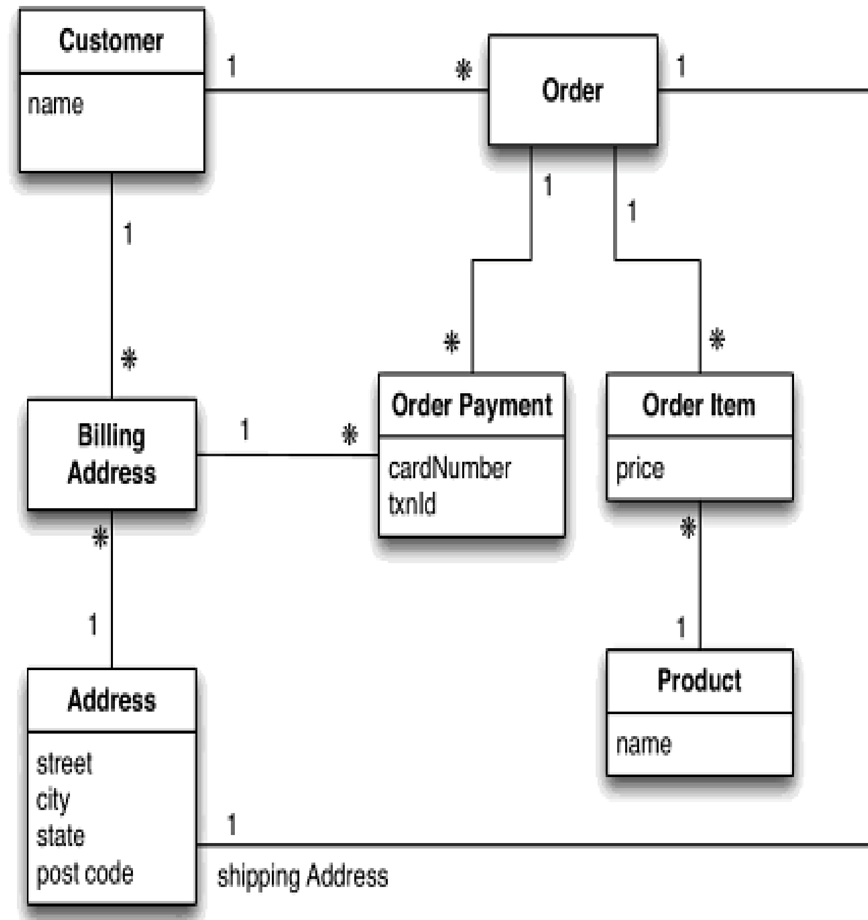
- In the provided aggregate model there are two “complex types” of records
  - **Customer**
    - Each customer record contains the customer profile, including his/her billing addresses
  - **Order**
    - Each order record contains all the data about one order
- Data are **denormalized** and some information is **replicated**

Aggregate implementation

- The solution (data model) is domain-driven
  - The aggregates are related to the expected usage of the data
- In the reported example we suppose to frequently read/write
  - Customer profiles (including shipping addresses)
  - Orders, with all the related information

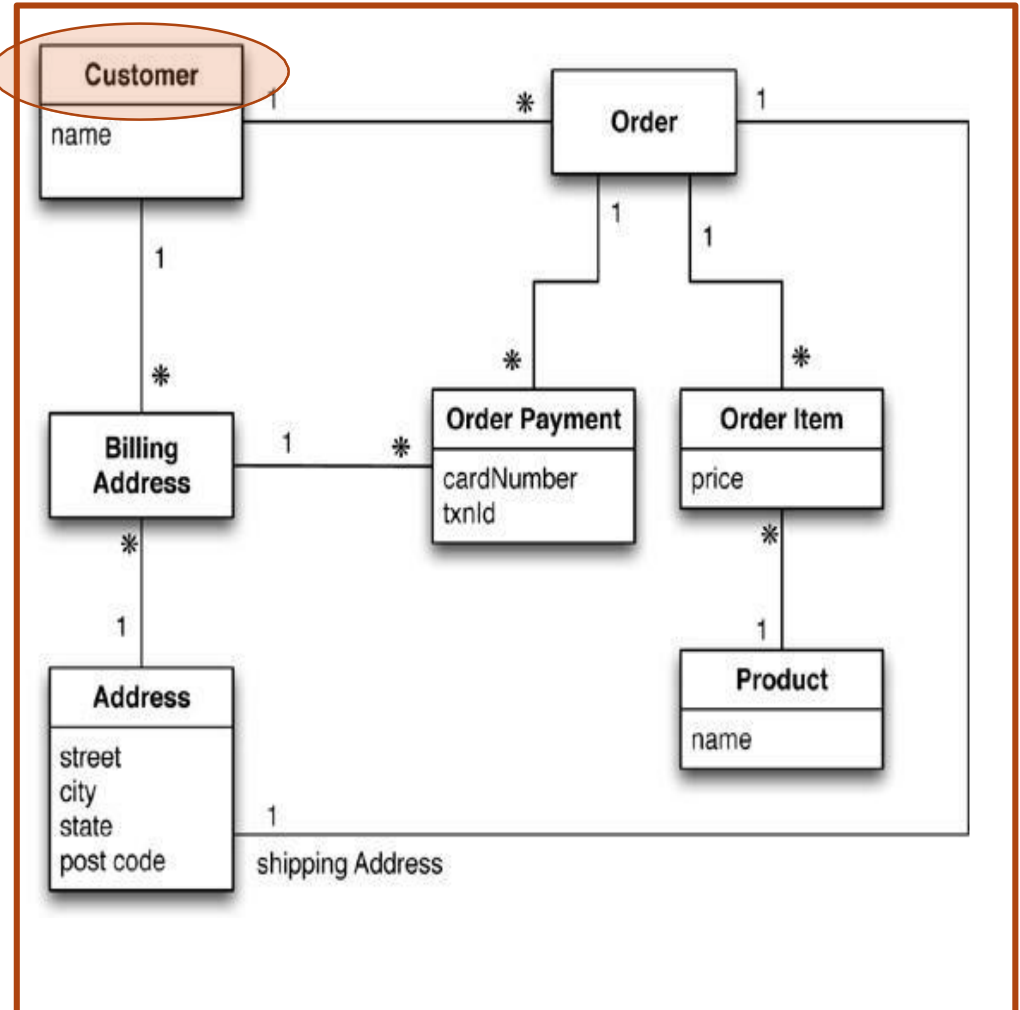
Exercise 2: domain-driven data access requests

- The billing address of customers is typically accessed together with the customer name
- When an order is accessed, all its items with the corresponding product names, and the payment information with the billing address of the paying customer are requested
- The shipping address of the order is requested when accessing an order
- **When a customer is accessed, all her/his orders are requested**



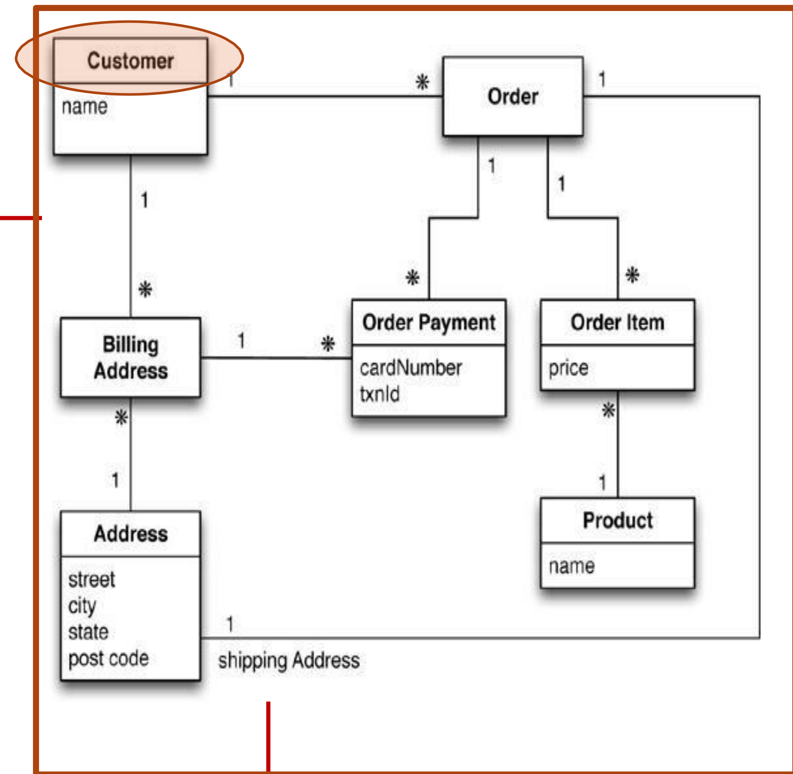
Aggregation 2

- We have only one aggregate in this model: customers and their orders, together



Solution 2

```
// (Single) Customer
{
    "id": 1,
    "name": "Fabio",
    "billingAddresses": [
        {
            "city": "Bari", …
        }
    ]
    "orders": [
      {
      "id": 99,
      "customerId": 1,
      "orderItems": [
        {"productId": 27,
         "price": 34,
         "productName": "Data Mngm book"
           }, {…} ],
       "shippingAddress": {"city": "Bari", … },
       "orderPayment": [
            { "ccinfo": "100-432423-545-134",
              "txnId": "afdfsdfsd",
              "billingAddress":
                  {"city": "Bari", … }
           }, {…} ]
       }, {…}, {…}, …]
}
```



Design strategy

- No universal answer for how to draw aggregate boundaries
- It depends entirely on how you tend to manipulate data
  - Accesses on a single order at a time and a single customer at a time
    - First solution
  - Accesses on one customer at a time with all her orders
    - Second solution
- Context-specific
  - Some applications will prefer one or the other

Aggregate Model

- The focus is on the unit(s) of interaction with the data storage
- Pros:
  - It helps greatly when running on a cluster of nodes
    - The data of each “complex record” will be manipulated together, and thus should the stored on the same node
- Cons:
  - An aggregate structure may help with some data interactions but be an obstacle for others

Aggregates

An aggregate

- A data unit with a complex structure
  - Not simply a tuple (a table row) like in RDBMS
- A collection of related objects treated as a unit
  - unit for data manipulation and management of consistency

- Relational model is aggregate-ignorant
  - It is not a bad thing, it is a feature
  - Allows to easily look at the data in different ways
  - Best choice when there is no primary structure for data manipulation