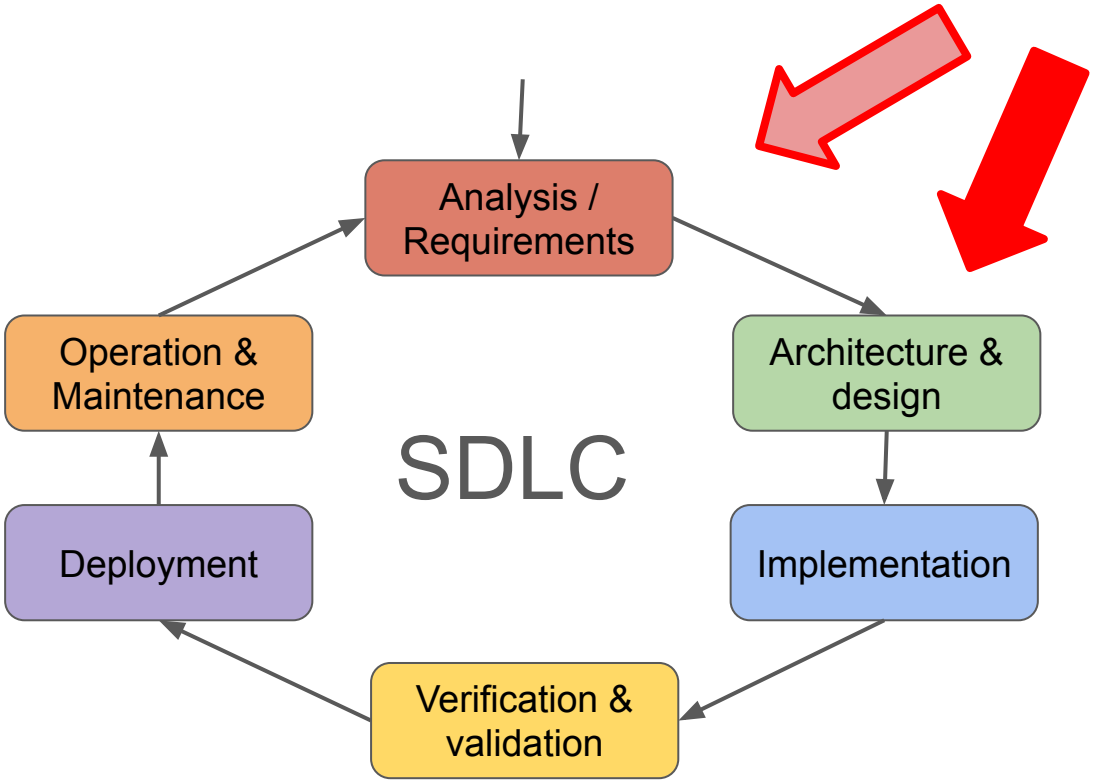
Databases
(Modeling, ORM)

Jana Kostičová, 5.11.2025


Analysis / Requirements
Architecture & design
Implementation
Verification & validation
Deployment
Operation & Maintenance
SDLC

Database vs DBMS

Database

= an organized collection of data, typically stored electronically, allowing easy retrieval, modification, and management of information

- Examples: flat or hierarchical file formats (txt, csv, tsv, json, xml, spreadsheets, …), relational tables in RDBMS

DBMS (**D**ata**B**ase **M**anagement **S**ystem)

= a software that manages and interacts with the data stored in a database

- Examples: RDBMS = Relational DBMS, NoSQL databases

Storage vs computation

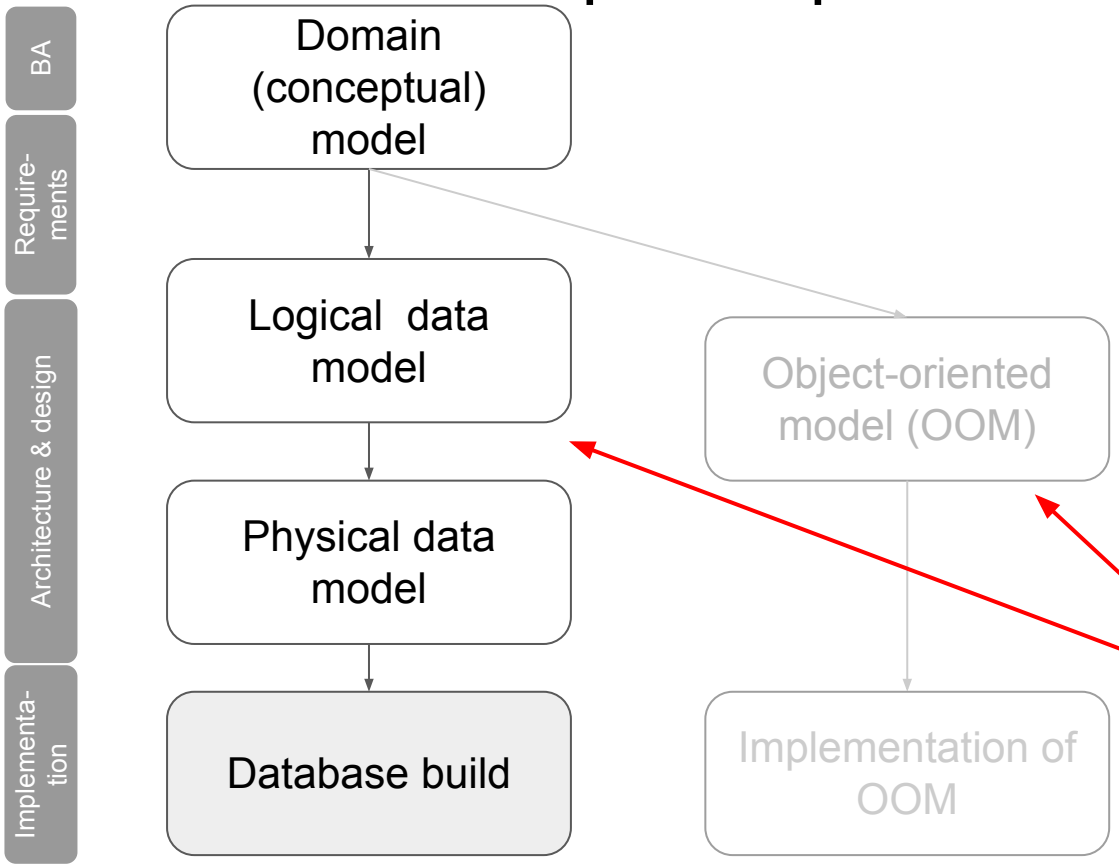
DBMSs

- Both storage and computation integrated in a single system
- Cannot be scaled separately

New trend: separating storage and computation

- Big data processing
- Distributed environment (cloud)
- Examples:
  - Storage: HDFS (Hadoop distributed file system), AWS S3, Google File System, ADLS (Azure Data Lake storage)
  - Computation: Apache Hadoop (MapReduce), Apache Spark, Apache Flink

Database development process



Data model = visual representation of data and its relationships within a database

Domain model is used to derive both more detailed data models as well as O-O models.

There is typically ORM (Object-relational mapping) between O-O model and relational data model.

Domain (conceptual) model

= High-level static model, visualization of domain concepts

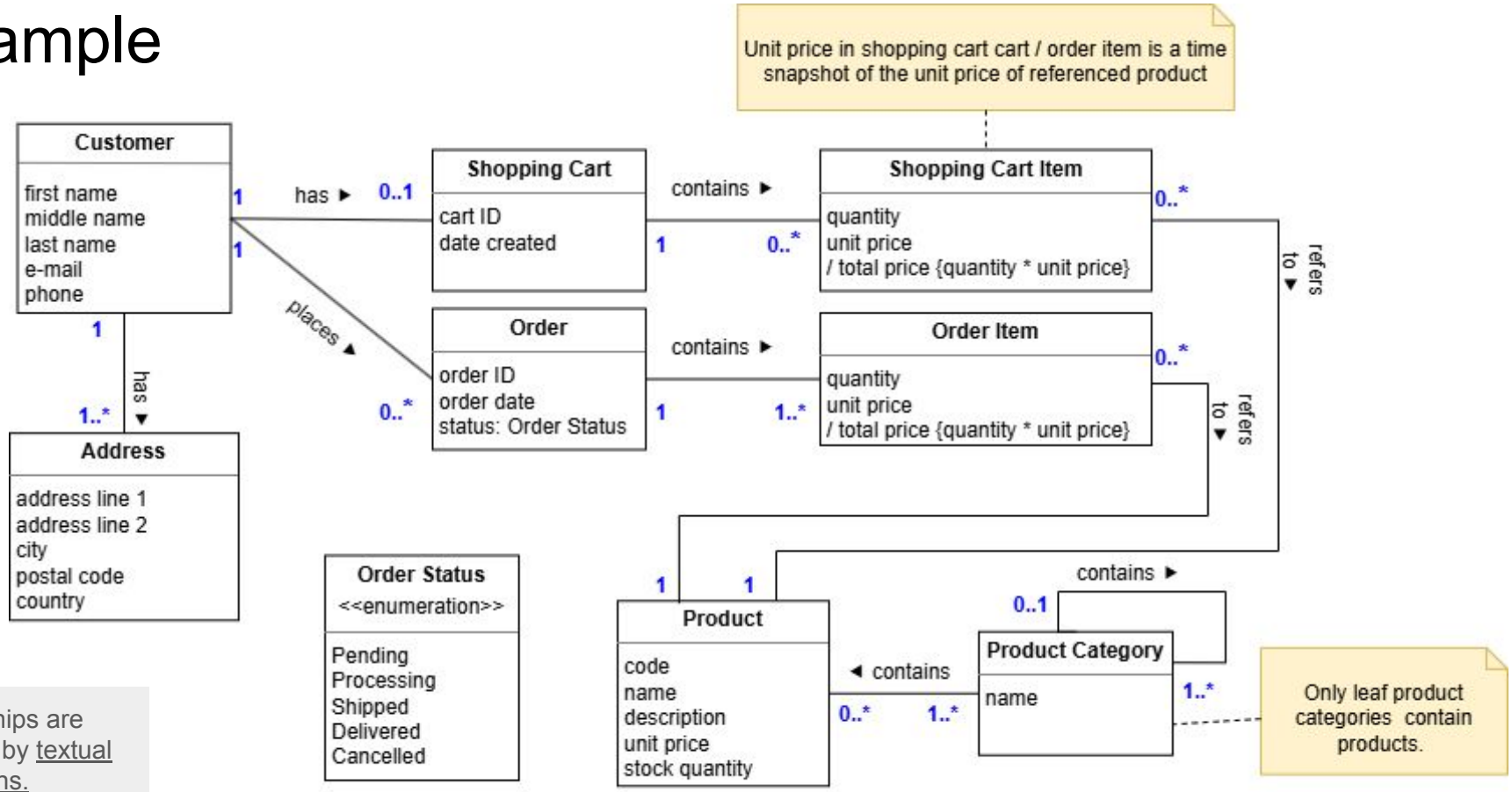
See “Requirements” lecture.

- Often supplemented with a glossary
- E-R diagram, UML class diagram or free-form

- Independent from implementation details and specific data storage mechanism
- Easily understood both for technical and non-technical people

Typically created during **Requirements phase.**

Example



Relationships are explained by textual descriptions.

Logical data model

= A more detailed view of the data, but still driven by business needs

- Describes entities, attributes, relationships (including cardinalities) and constraints
  - Typically describe abstract types for attributes and referential integrity (primary keys, foreign keys)
- E-R diagram, UML class diagram
- Data normalization (if it is in accordance with the requirements !)

- Dependent on logical data structure
- Still independent from specific DBMS
- Still understood by non-technical people

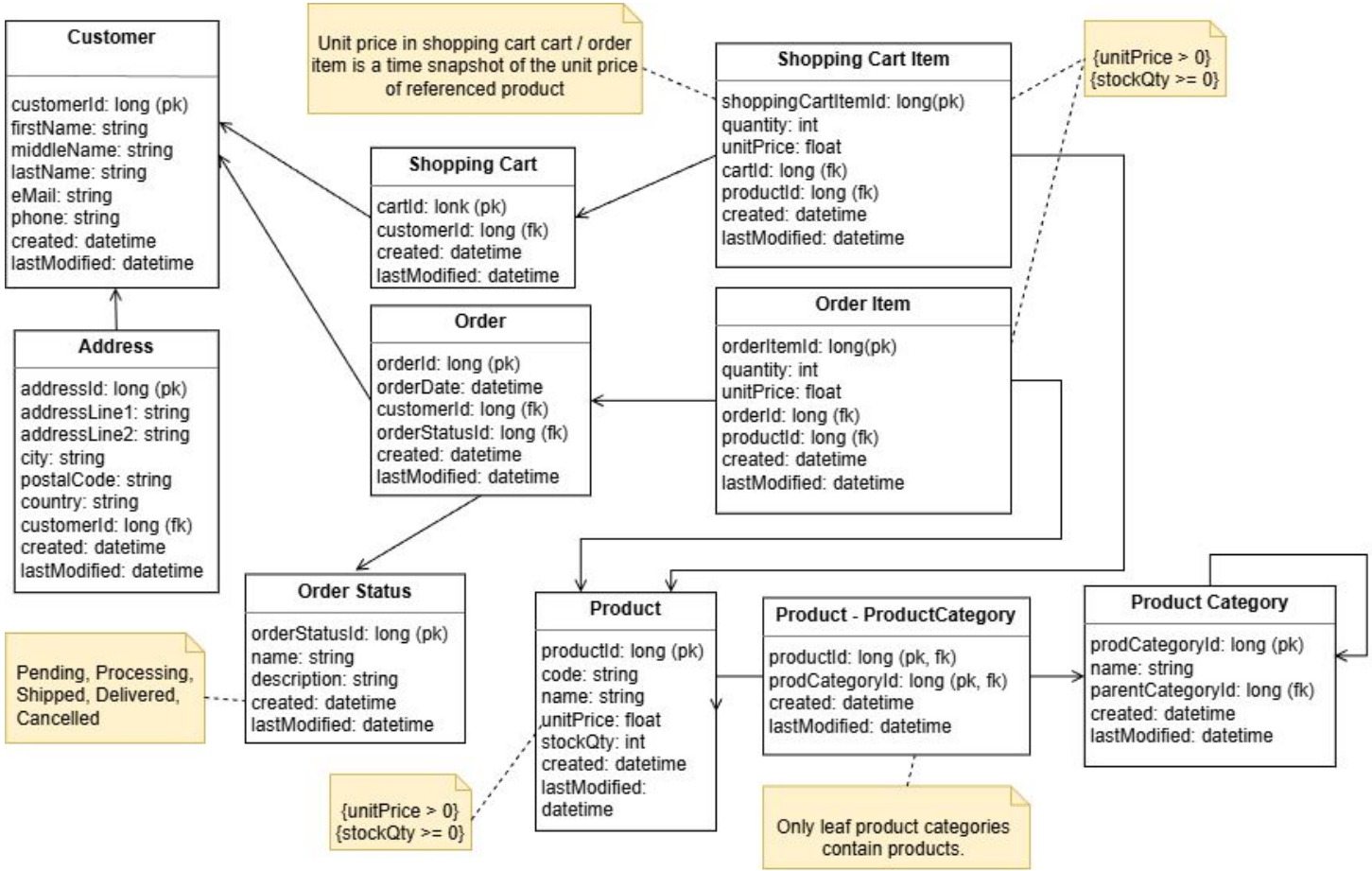
Typically created during **Design & Architecture phase.**

Example

Logical model for relational tables

This model contains explicit foreign keys so it is assumed that non-technical people understand this notation.

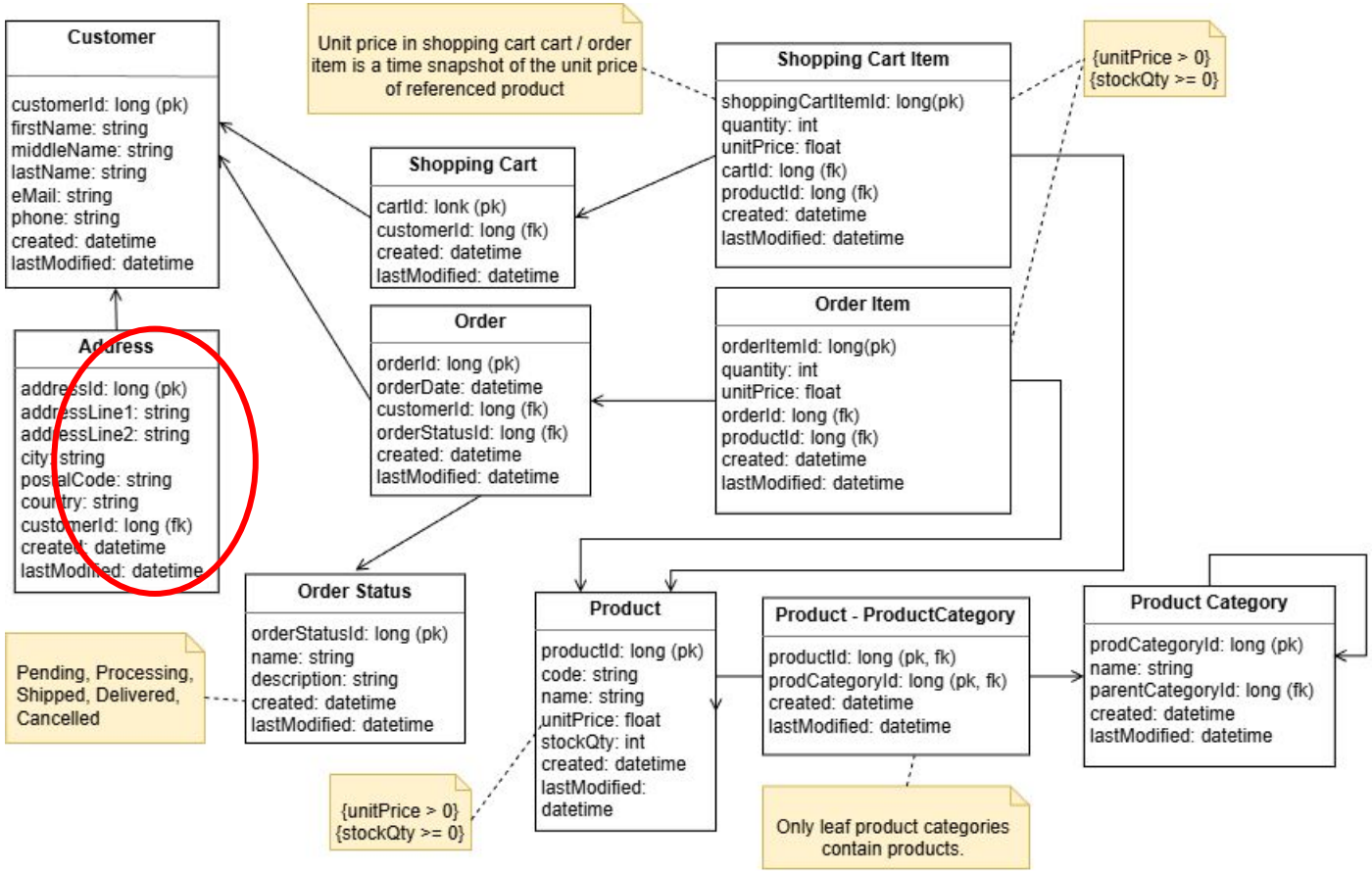
If not, it is better to keep the relationship descriptions and cardinalities used in the conceptual model.



Example

Logical model for relational tables

Abstract data types

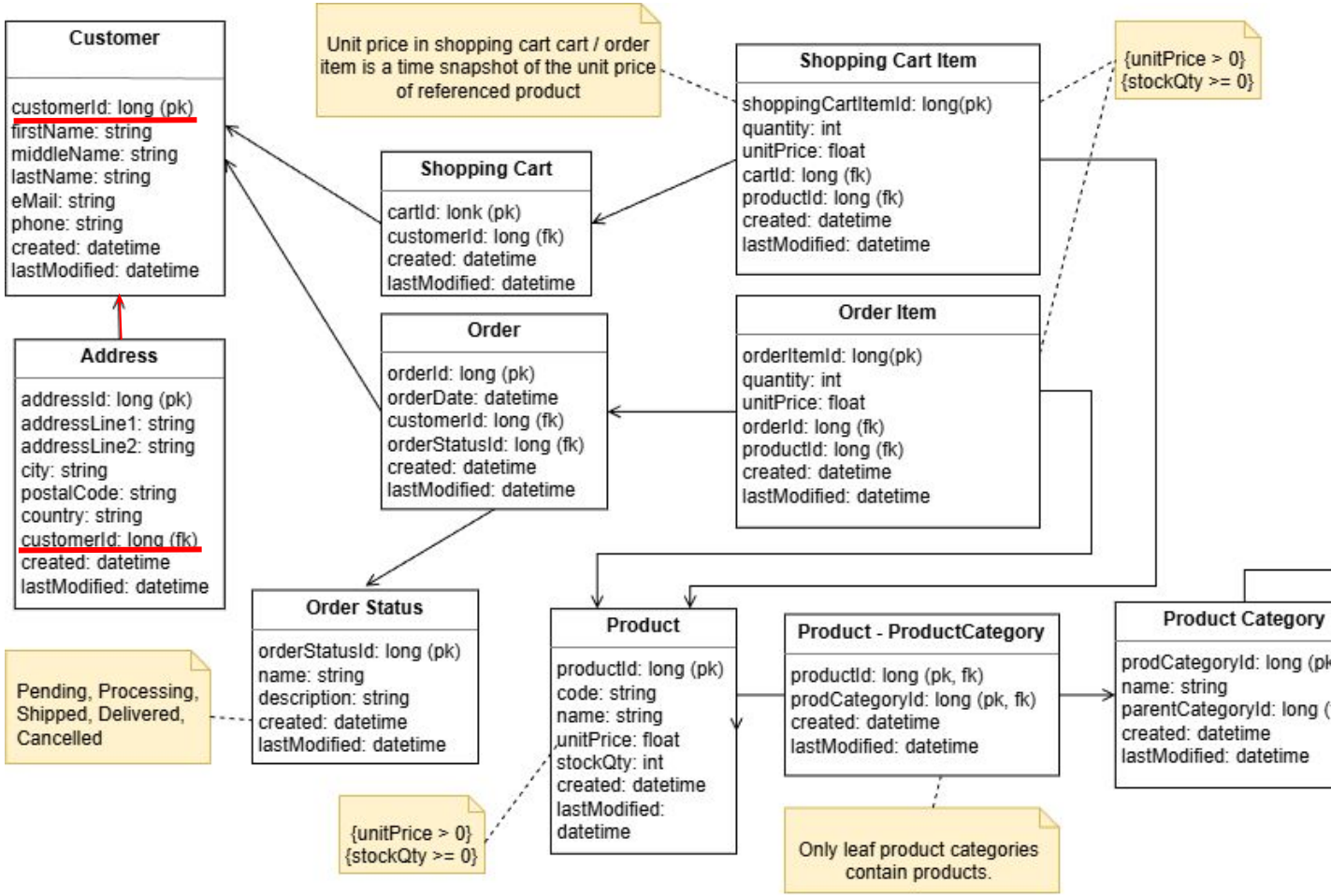


Example

Logical model for relational tables

Referential integrity (pks / fks)
Oriented associations

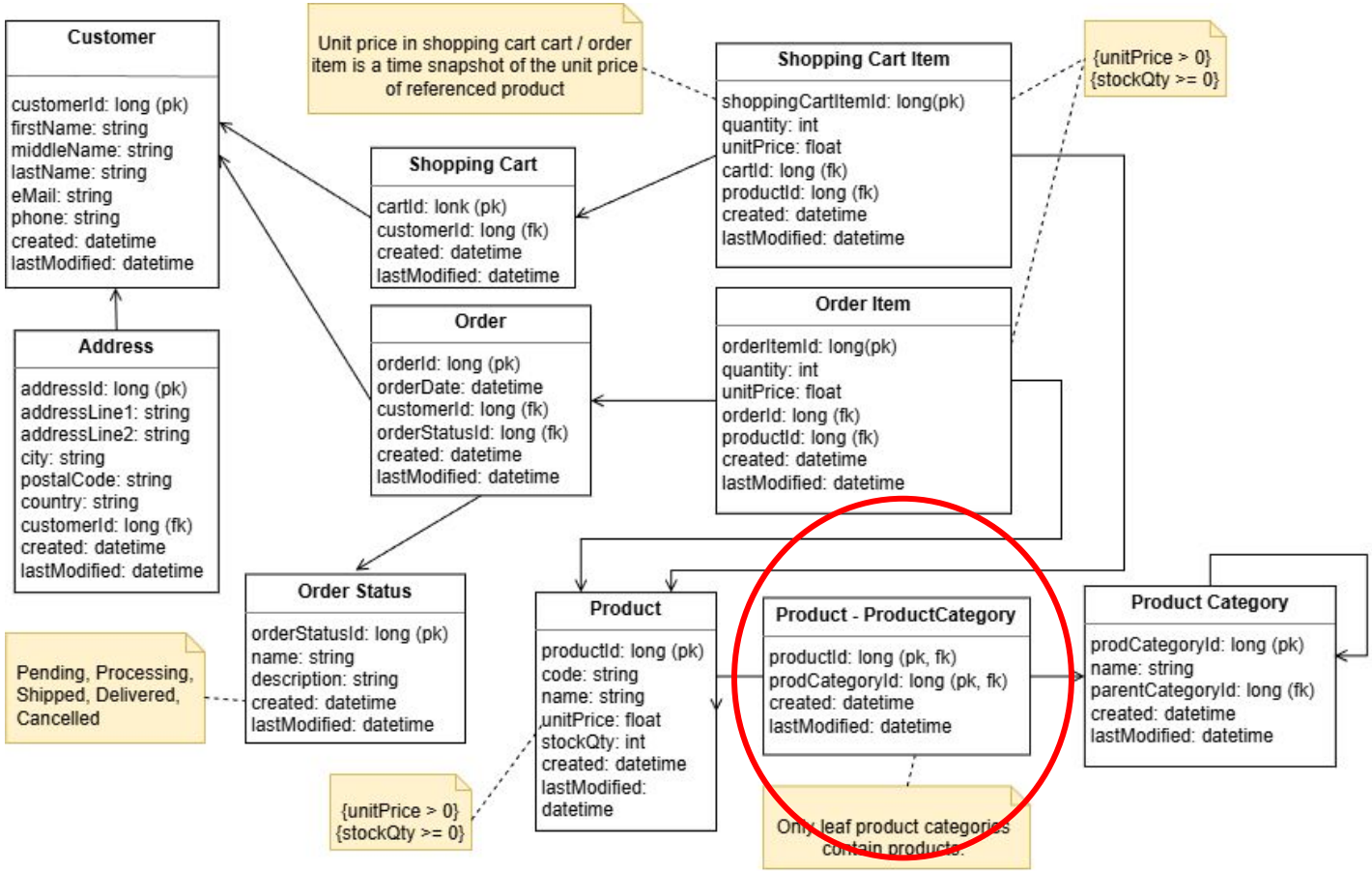
Primary keys:
Surrogate or natural (often surrogate key is used in addition to the natural one)



Example

Logical model for relational tables

New “join” tables for N:M associations

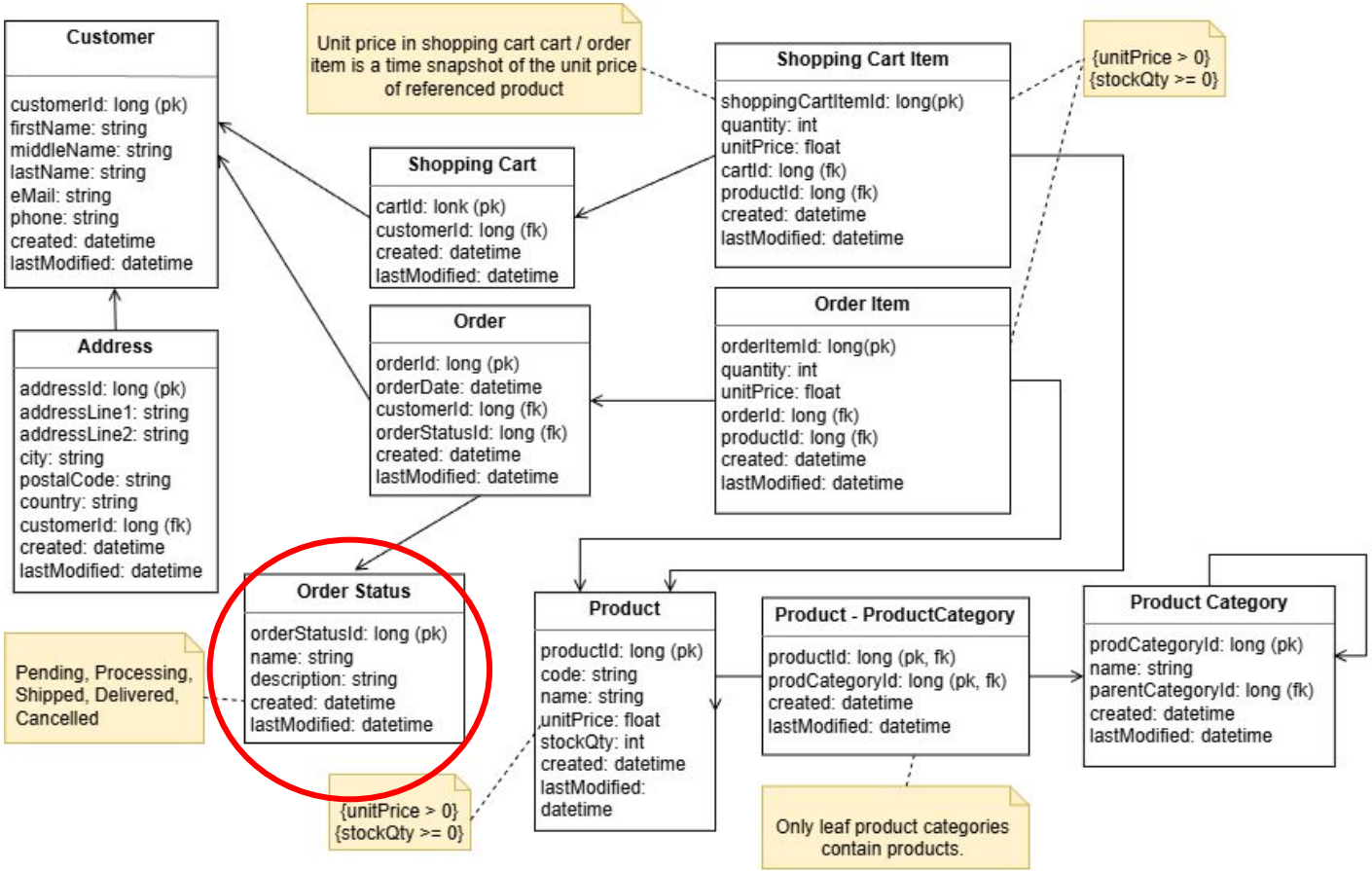


Example

Logical model for relational tables

Enumeration
→ tables
(preferred approach)

Values can be listed in a note for clarity

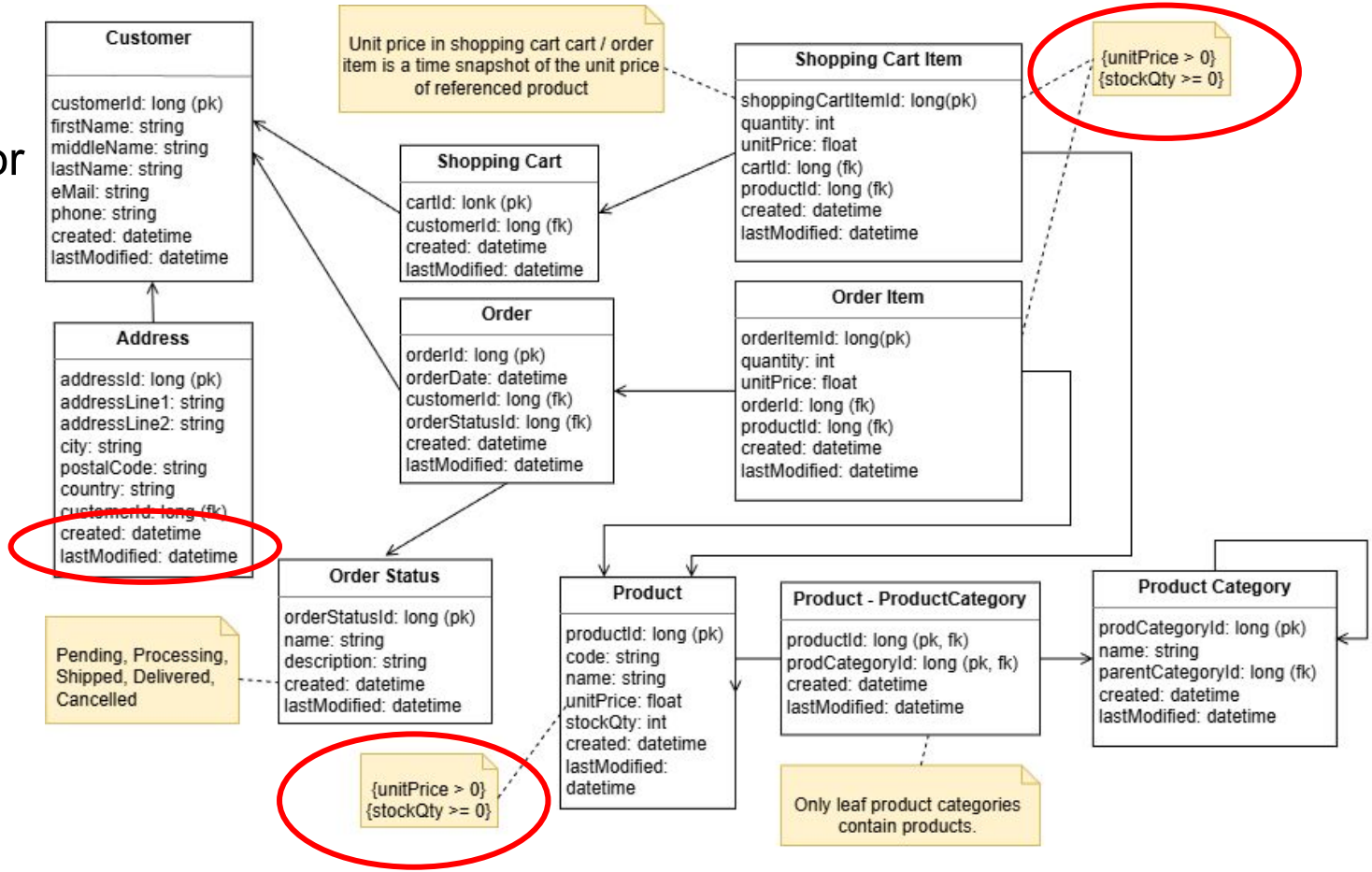


Example

Logical model for relational tables

Additional attributes and integrity checks

A separate check for each "1:* cardinality" is needed (these checks are not shown in this logical model)



Example

Logical model for relational tables

Skipped:

- Derived attributes Shopping Cart Item -> total price, Order Item -> total price
  - Design decision -> computed on-the-fly (not persisted)
- Association names and cardinalities
  - Can be (mostly) derived from referential integrity
  - Additional checks needed for “1..*” cardinality

Physical data model

= A database-specific representation of the data
= Actual representation of the database

This step starts with associating the model with a Database Management System (DBMS).

- Describes tables, columns and referential integrity
- Describes indexes, triggers, constraints, stored procedures, functions, …
- Uses DBMS specific data types
- Uses DBMS compatible table names and column names
- Validation and optimization (e.g. denormalization)

Typically created during **Design & Architecture phase.**

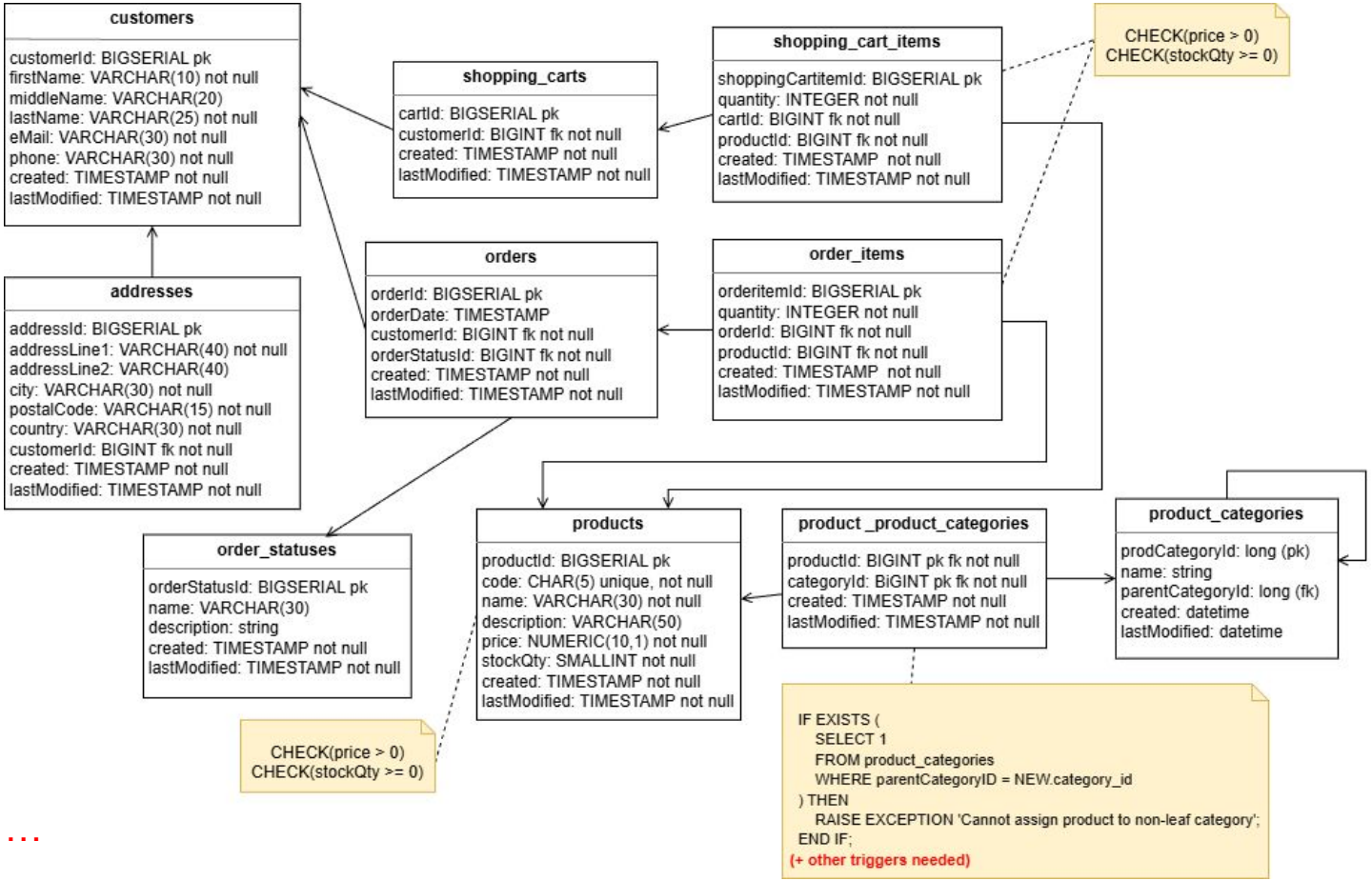
Example

Physical model for PostgreSQL

DB-specific table names, attribute names and data types

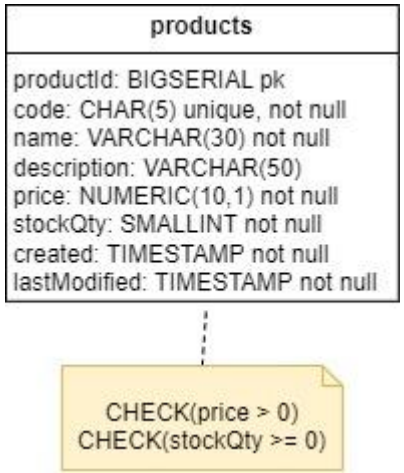
Business rules/ Integrity checks substituted with constraints

Null / not null, unique, …



Example

Physical model for PostgreSQL




```
CREATE TABLE products (
  productId BIGSERIAL,
  code CHAR(5) UNIQUE,
  name VARCHAR(15) NOT NULL,
  description VARCHAR(50),
  price  NUMERIC(10,1) NOT NULL,
  stockQty SMALLINT NOT NULL,
  created TIMESTAMP NOT NULL DEFAULT NOW(),
  lastModified TIMESTAMP NOT NULL DEFAULT NOW(),
  PRIMARY KEY(productId),
  CHECK (price > 0),
  CHECK (stockQty >= 0)
);
```

(Default can be captured also in the graphical representation)

Object-relational mapping - ORM

= a programming technique used to facilitate the interaction between object-oriented code and relational databases

ORM allows developers to work with databases using object-oriented programming (OOP) concepts, such as classes, objects, and methods, rather than writing raw SQL queries.

- Modeling data
- Creating / altering DB according to the model
- CRUD - inserting / querying / updating / deleting data

```
# conn = sqlite database connection
cursor = conn.execute("SELECT * from customers WHERE lastName='Doe'
OR lastName='Newton'")

for row in cursor:
   print("ID = ", row["customerId"])
   print("FIRST_NAME = ", row["firstName"])
   print("FULLNAME = ", row["lastName"], "\n")

conn.close()
```

Traditional DB access

```
# engine = sqlalchemy db engine
# Customer = mapped class, part of the model
session = Session(engine)
stmt = select(Customer).where(Customer.lastName.in_(["Doe", "Newton"]))

for customer in session.scalars(stmt):
   print("ID = ", customer.customerId)
   print("NAME = ", customer.firstName)
   print("FULLNAME = ", customer.lastName, "\n")

session.close()
```

ORM

Object-relational impedance mismatch

There are several fundamental differences between OO design and DB design:

- Classes have instances, inheritance, relationships; relational databases have just tables
- References vs pointers
- Datatype differences
- Database normal forms make little sense in OOP

Object-relational impedance mismatch is a set diculties that encountered if we use relational databases with an OO application.

OODBMSs and document stores

OODBMS - Object-oriented databases

- Eliminate the need for converting data to and from its SQL form, as the data is stored in its original object representation and relationships are directly represented

Documents stores (JSON, XML)

- Storing objects in tree structures is more straightforward than in relational tables

ORM software - examples

- Java: Hibernate
- Python: SQLAlchemy (see Example), Django ORM
- C# / .NET: Entity Framework
- JavaScript / TypeScript / Node.js: Sequelize, TypeORM, Prisma
- …

ORM - pros and cons

(+) **Abstraction of database details** - ORM abstracts the low-level DB details and allows developers to work with high-level, object-oriented code, making it easier to understand and maintain
(+) **OO approach** - ORM aligns well with object-oriented programming (OOP) principles. It allows you to model your data as objects, which can make your code more intuitive and maintainable
(+) **Reduced amount of code** - the amount of repetitive SQL code and database-related logic is reduced
(+) **Security** (prevents from SQL injection)

(-) **Abstraction of database details** - ORM abstracts DB details, which can limit the control over certain database-specific features and optimizations.
(-) **Possible performance overhead** - queries are created and executed dynamically and cannot be hand-optimized
(-) **Steeper learning curve**
(-) **Complex queries** - it may be difficult to express more complex queries

Non-parameterized query (security risk)

```
# conn = sqlite database connection
cursor = conn.execute("SELECT * from customers WHERE
lastName='{lastName1}' OR lastName='{lastName2}'")

…
```

Parameterized query

```
# conn = sqlite database connection
cursor = conn.execute("SELECT * from customers WHERE lastName=? OR
lastName=?",(lastName1, lastName2))

…
```