# Object-relational mapping

# Pros and cons of JDBC (SQL-based technologies)

Pros

- Clean and simple SQL processing
- Good performance with large data
- Good for small applications
- Simple syntax, easy to learn

Cons

- Complex when used in large projects
- Large programming overhead
- Impedance mismatch – the object model of OOP and the relational model of RDBMS do not work very well togeather.
- Query is RDBMS specific

# Impedance mismatch

| Mismatch | Explained |
| --- | --- |
| Granularity | Sometimes, an object model can have more classes than the number of corresponding tables in the database. An example is an “Address” class. |
| Subtypes (inheritance) | Natural paradigm in OOP languages; RDBMS does not define anything similar. |
| Identity | A RDBMS defines one notion of "sameness", the primary key. In Java however there is object identity (a==b) and object equality (a.equals(b)). |
| Associations | Object-oriented languages represent associations using object references where as a RDBMS represents an association as a foreign key column. A bidirectional relationship in Java requires that the association is defined twice, from both sides. |
| Navigation | The ways you access objects in Java and in a RDBMS are fundamentally different. In Java you navigate from one association to another by walking the object network. This is not efficient in RDBMS where you want to minimize the number of SQL queries and thus you load several entities via joins before walking the object network. |

# What is ORM?

- Stands for Object-Relational Mapping.
- Programming technique for converting data between relational databases and object-oriented programming languages such as Java.
- Advantages:
  - Allows business code to access objects rather than DB tables.
  - Hides details of SQL queries from OO logic.
  - No need to deal with the RDBMS implementation.
  - Entities based on business concepts rather than database structure.
  - Transaction management and automatic key generation out of the box.
  - Fast development.