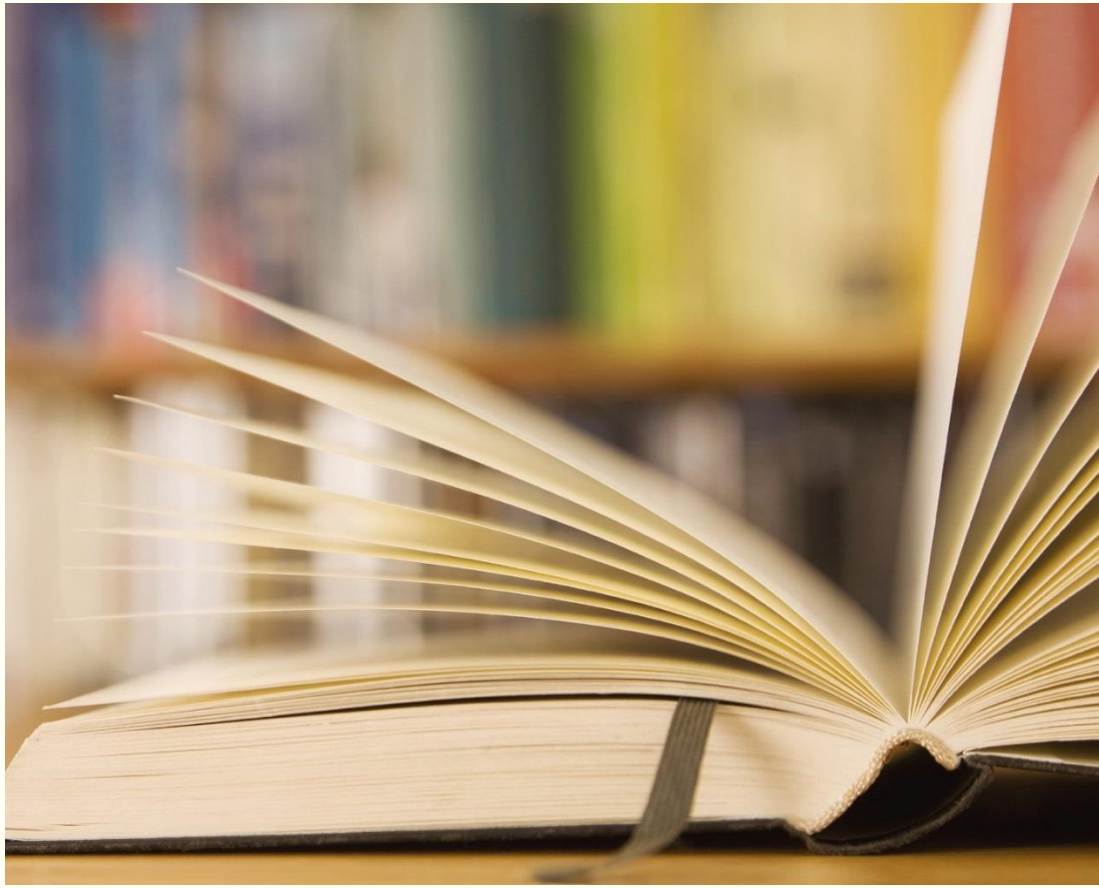
# ENTERPRISE ARCHITECTURE

Najeeb Najeeb, PhD





# INHERITANCE

# Inheritance

- Inheritance is a critical for OO to achieve polymorphism.
- The problem
  - RDBMS does not have inheritance as a default property.
  - We need a solution to enable us to achieve inheritance in an RDBMS.
  - Consider polymorphism when building RDBMS inheritance solutions.
- Do we need more than one solution?
- Is an RDBMS always properly designed?
- Is it possible to simulate inheritance in an RDBMS?

# STRATEGIES

# Single-Table Strategy

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="P_TYPE")
public abstract class Person {
    @Id@GeneratedValue(...)
    private int id;
    private String name;
    private int age;
}

@Entity
@DiscriminatorValue("Std")
Public class Student extends Person{
    private float gpa;
    private int courses;
}

@Entity
@DiscriminatorValue("Emp")
Public class Employee extends Person {
    private float salary;
    private int office;
}
```

```
TABLE : Person
id          INT             PK
name    VARCHAR(20)
age         INT
gpa         FLOAT
courses   INT
salary    FLOAT
office    INT
P_TYPE   VARCHAR(10)
```

# Single-Table Strategy TABLE

| id | name | age | gpa | courses | salary | room | P-TYPE |
|---|---|---|---|---|---|---|---|
| 1 | Jack | 25 | 3.3 | 3 | | | Std |
| 2 | John | 23 | 3.4 | 7 | | | Std |
| 3 | Jill | 26 | | | 1200 | 2 | Emp |
| 4 | James | 29 | | | 1000 | 16 | Emp |
| 5 | Jim | 28 | 2.9 | 2 | | | Std |
| 6 | July | 24 | 3.8 | 5 | | | Std |
| 7 | Jessica | 31 | | | 1100 | 10 | Emp |
| 8 | Jeremy | 22 | | | | | Person |

# Joined Strategy

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="P_TYPE")
public abstract class Person {
    @Id@GeneratedValue(...)
    private int id;
    private String name;
    private int age;
}

@Entity
@DiscriminatorValue("Std")
Public class Student extends Person{
    private float gpa;
    private int courses;
}

@Entity
@DiscriminatorValue("Emp")
Public class Employee extends Person {
    private float salary;
    private int office;
}
```

```
TABLE : Person
id          INT             PK
name    VARCHAR(20)
age         INT
P_TYPE   VARCHAR(10)

TABLE : Student
id          INT             PK/FK
gpa       FLOAT
courses   INT

TABLE : Employee
id          INT             PK/FK
salary    FLOAT
office    INT
```

# Joined Strategy TABLES

Person

| id | name | age | P-TYPE |
|---|---|---|---|
| 1 | Jack | 25 | Std |
| 2 | John | 23 | Std |
| 3 | Jill | 26 | Emp |
| 4 | James | 29 | Emp |
| 5 | Jim | 28 | Std |
| 6 | July | 24 | Std |
| 7 | Jessica | 31 | Emp |

Student

| id | gpa | courses |
|---|---|---|
| 1 | 3.3 | 3 |
| 2 | 3.4 | 7 |
| 5 | 2.9 | 2 |
| 6 | 3.8 | 5 |

Employee

| id | salary | room |
|---|---|---|
| 3 | 1200 | 2 |
| 4 | 1000 | 16 |
| 7 | 1100 | 10 |

# Table-per-Concrete-Class* Strategy

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Person {
    @Id@GeneratedValue(...)
    private int id;
    private String name;
    private int age;
}

@Entity
Public class Student extends Person{
    private float gpa;
    private int courses;
}

@Entity
Public class Employee extends Person {
    private float salary;
    private int office;
}
```

```
TABLE : Student
id          INT             PK
name        VARCHAR(20)
age         INT
gpa         FLOAT
courses     INT

TABLE : Employee
id          INT             PK
name        VARCHAR(20)
age         INT
salary      FLOAT
office      INT
```

# Table-per-Concrete-Class Strategy TABLE

Student

| id | name | age | gpa | courses |
|---|---|---|---|---|
| 1 | Jack | 25 | 3.3 | 3 |
| 2 | John | 23 | 3.4 | 7 |
| 5 | Jim | 28 | 2.9 | 2 |
| 6 | July | 24 | 3.8 | 5 |

Employee

| id | name | age | salary | room |
|---|---|---|---|---|
| 3 | Jill | 26 | 1200 | 2 |
| 4 | James | 29 | 1000 | 16 |
| 7 | Jessica | 31 | 1100 | 10 |

# Pros & Cons

- @MappedSuperclass
- Mixed Inheritance
- Single Table
  - Peak performance for polymorphic queries
  - Peak performance for writes
  - No Joins
- Joined Table
  - Data normalized
- Table-per-concrete class
  - Single entity lookup is cheap.
  - No discriminator column.
- Single Table
  - Normalization (not normalized)
  - Storage waste
- Joined Table
  - Single entity lookup is expensive
  - Insert is also expensive
  - PK of table is limited to type of parent's PK
- Table-per-concrete class
  - Polymorphic query very expensive
  - Need to issue union or multiple queries on each table

# Main Point

- While RDBMS does not natively support inheritance JPA is able to simulate inheritance in the DB. There are different inheritance strategies to address the deferent needs when implementing inheritance, both from the DB or OO side.
- All the laws of nature are expressions of the Unified Field. The Unified Field is the ultimate inheritance strategy.