# Mastering Hibernate

## The Ultimate Guide to Java ORM

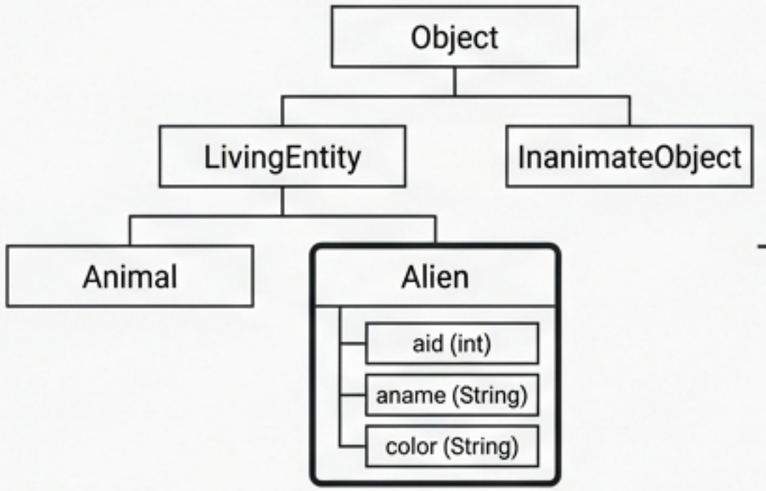
---

Based on the Telusko Hibernate Tutorial

# The Impedance Mismatch

## Bridging Objects and Relations

### Java (OOP)




- Inheritance & Polymorphism
- Transient Data (In Memory)
- Example: Alien Object (aid, aname, color)

```
Alien Object {
  aid: 101, aname: "Zorg",
  color: "Green"
}
```


Hibernate automates the translation, allowing developers to persist data without writing complex JDBC.

### Database (RDBMS)

**Alien Table**

| aid | aname | color | planet_id (FK) |
|---|---|---|---|
| 101 | "Zorg" | "Green" | 5 |
| 102 | "Xylar" | "Purple" | 7 |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |

**Planet Table**

| planet_id | planet_name |
|---|---|
| ... | ... |

- Tables & Foreign Keys
- Persistent Data (On Disk)
- Example: Alien Table (columns: aid, aname, color)

```
CREATE TABLE Alien
  (aid INT PRIMARY KEY,
  aname VARCHAR(50),
  color VARCHAR(20));
```

# The Foundation: Maven & Configuration

## Dependency Management (pom.xml)

```xml
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.x</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.x</version>
</dependency>
```

## The Control Center (hibernate.cfg.xml)

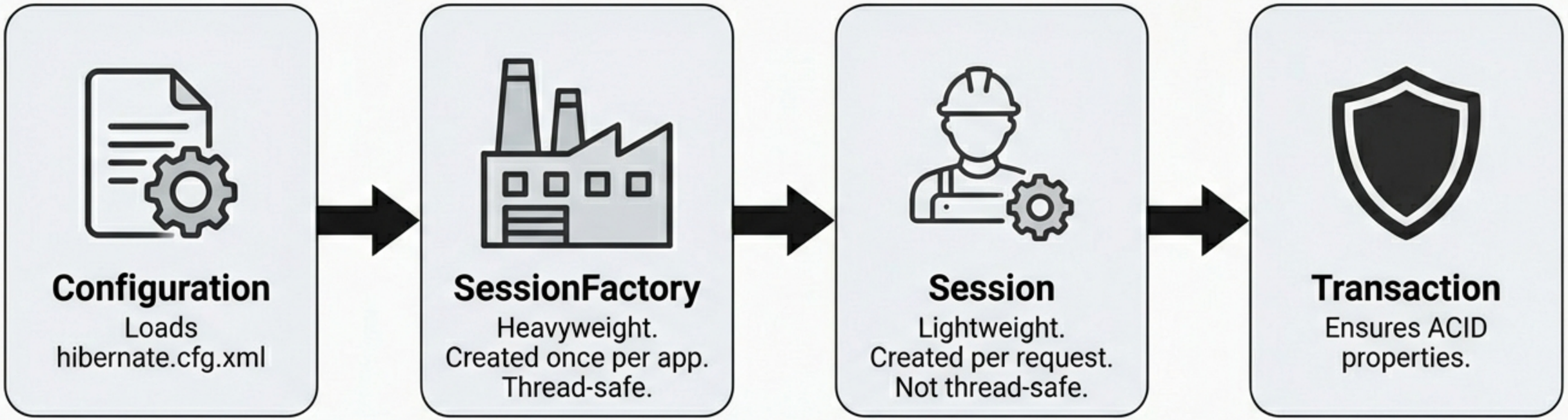
```xml
<hibernate-configuration>
    <session-factory>
        <!-- Database Connection -->
        <property name="connection.url">jdbc:mysql://localhost:3306/neon</property>
        <property name="connection.username">root</property>

        <!-- The Translator -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Schema Management -->
        <property name="hbm2ddl.auto">update</property>
    </session-factory>
</hibernate-configuration>
```

`update` = alters schema safely; `create` = drops and recreates (dataloss risk).

# Core Architecture: The Factory Pattern




```
Configuration con = new Configuration().configure().addAnnotatedClass(Alien.class);
SessionFactory sf = con.buildSessionFactory();
Session session = sf.openSession();

Transaction tx = session.beginTransaction();
// ... Database Operations ...
tx.commit();
```

# Mapping an Entity: The Alien Class

## The Java Definition

```java
@Entity
public class Alien {
    @Id
    private int aid;

    @Column(name="alien_name")
    private String aname;

    @Transient // Not stored in DB
    private String nickname;

    private String color;
}
```

Defines class as a database table.

Primary Key.

Ignored by Hibernate.

## The Persistence Operation

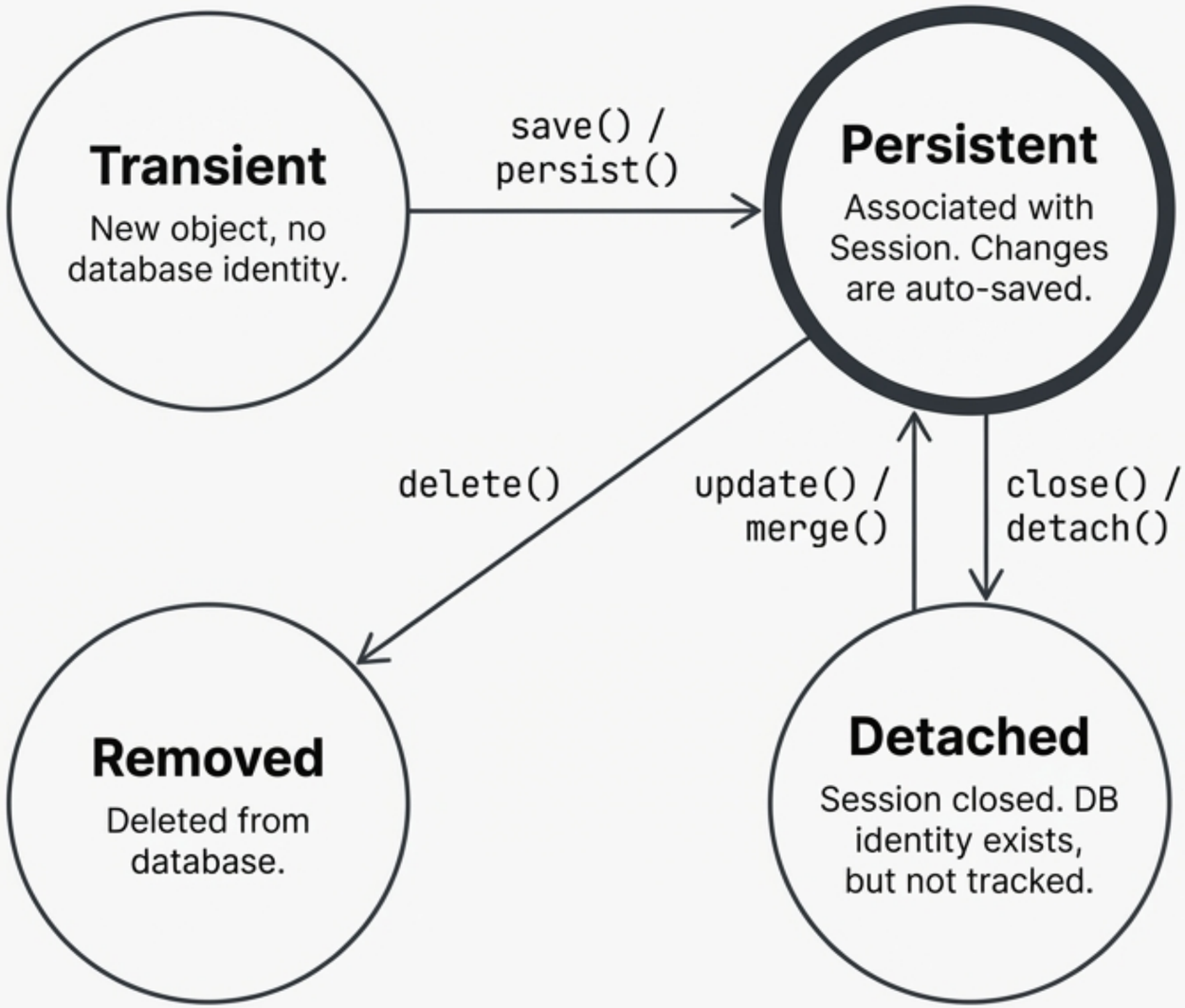
```java
Alien telusko = new Alien();
telusko.setAid(101);
telusko.setAname("Naveen");
telusko.setColor("Green");

session.save(telusko);
```

### Result:

```
INSERT INTO Alien (aid, alien_name,
color) VALUES (101, 'Naveen', 'Green');
```

# The Persistence Lifecycle: Object States



# Fetching Strategies: get() vs load()

```
session.get(Alien.class, 101)
```

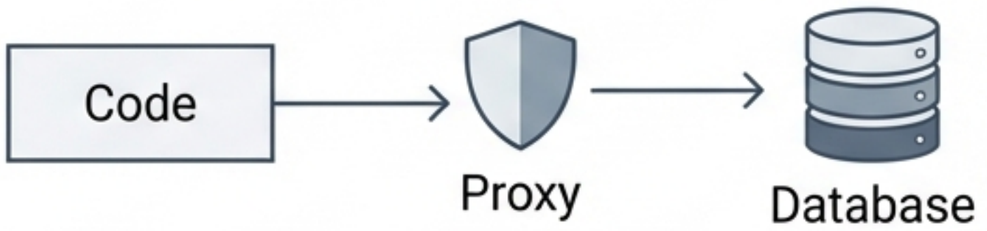
**Execution:** Eager. Hits the database immediately.


**Result:** Returns the real Object or `null`.

**Use Case:** When you aren't sure if the object exists.

```
session.load(Alien.class, 101)
```

**Execution:** Lazy. Hits database only when data is accessed.




**Result:** Returns a Proxy (placeholder). Throws `ObjectNotFoundException` if missing.

**Use Case:** Performance optimization. When you only need the reference.

# Composition: Embeddable Objects

Mapping complex objects to a single table.

**Alien**

```
int aid;
String color;
@Embedded AlienName aname;
```

**AlienName**

```
@Embeddable
class AlienName {
   String fName;
   String mName;
   String lName;
}
```

Hibernate Mapping

**Alien Table**

| aid | color | fName | mName | lName |
|---|---|---|---|---|
| 101 | Green | John | William | Doe |

`AlienName` did not become its own table; its fields were flattened into the `Alien` table.

# Mapping Relationships: 1-to-1 & 1-to-Many

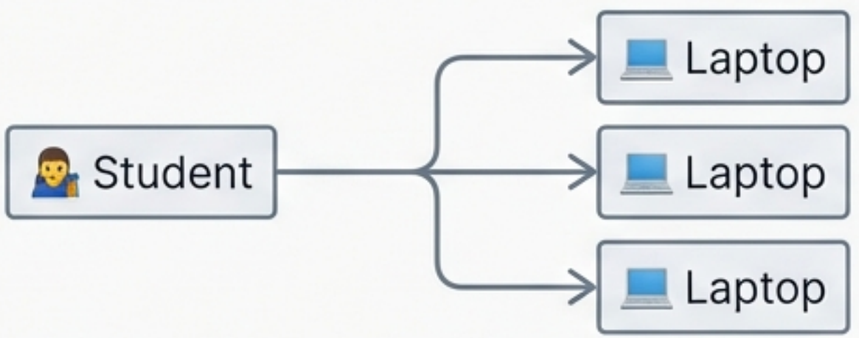
## One-to-One (@OneToOne)

A Student has one Laptop.


```
@OneToOne
private Laptop laptop;
```

## One-to-Many (@OneToMany)

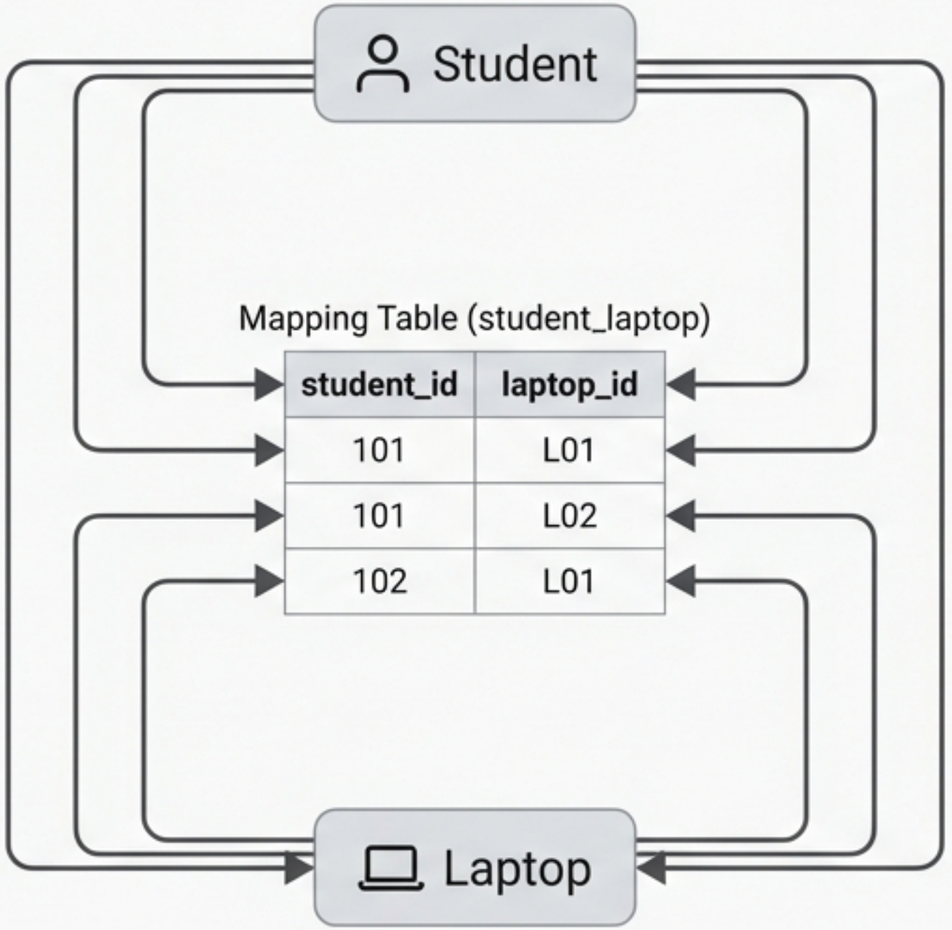
A Student has a list of Laptops.




```
@OneToMany(mappedBy="student")
private List<Laptop> laptops = new ArrayList<>();
```

# Advanced Relationships: Many-to-Many

## Handling complex associations with @ManyToMany




### MappedBy

⚠️ Without configuration, Hibernate creates two mapping tables.

✅ Use ‘mappedBy’ to designate the owner.

```
@ManyToMany(mappedBy="student")
private List<Student> students = new ArrayList<>();
```

Tells Hibernate:‘The other side handles the link. Don’t create a duplicate table.’

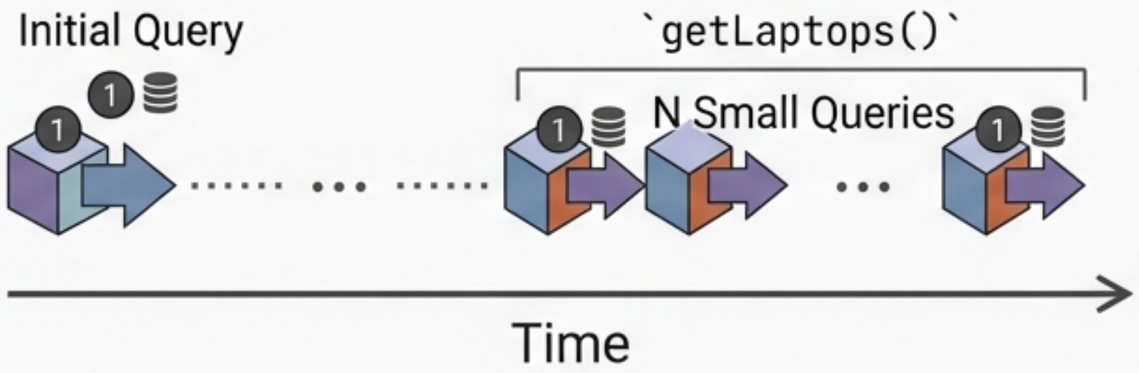
# Performance Tuning: Fetch Types

The "Boss" Analogy.

## Lazy Fetching (Default)

Efficient. Waits to be asked.

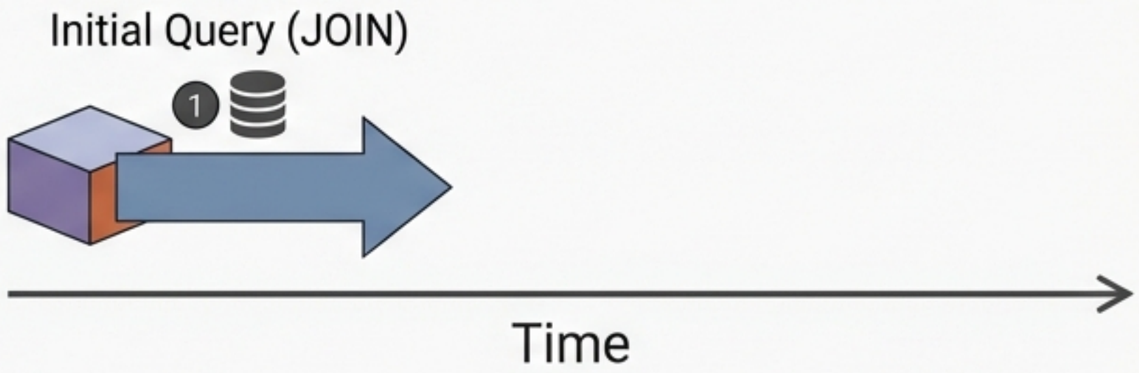
```
fetch = FetchType.LAZY
```




## Eager Fetching

Heavy. Loads everything upfront.

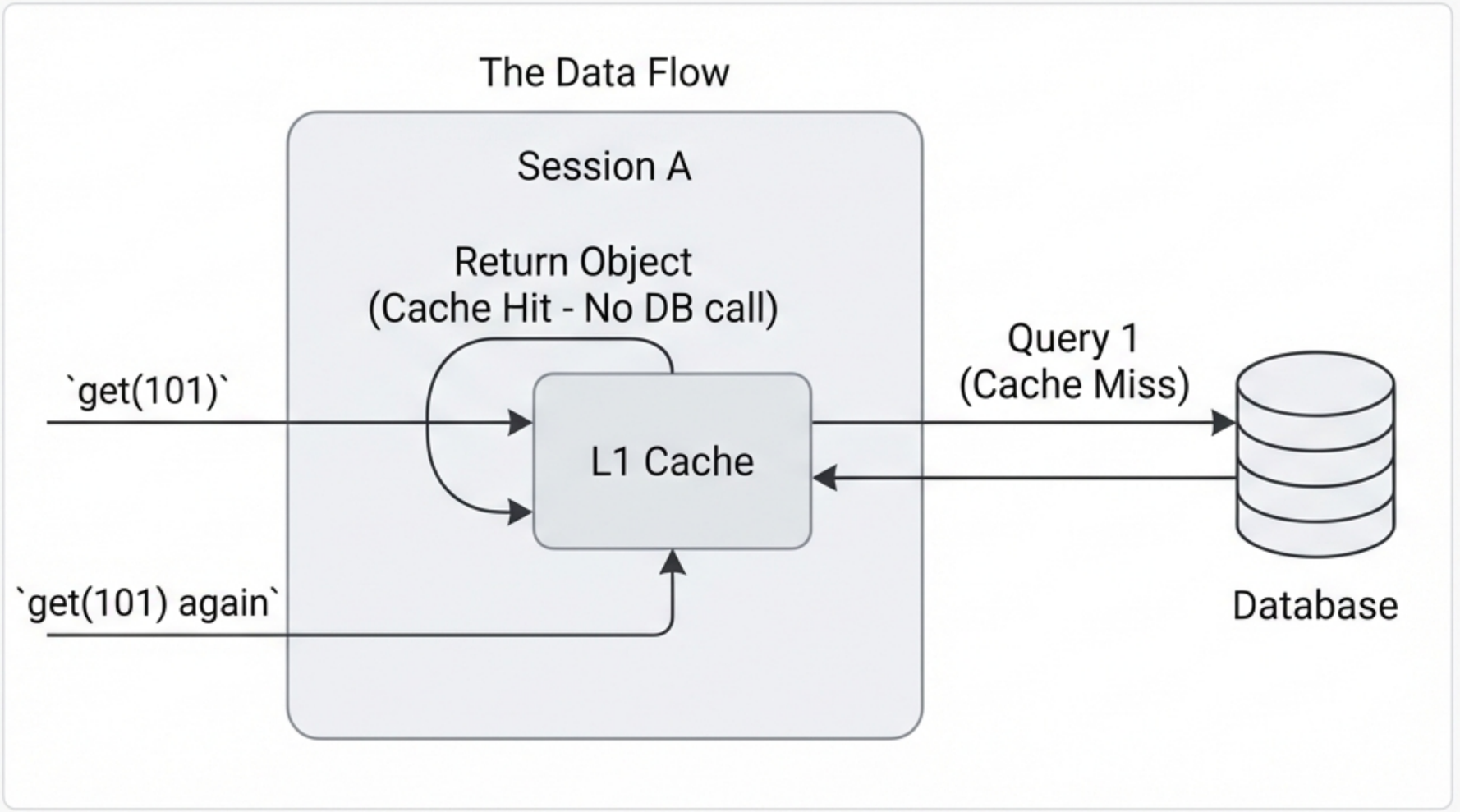
```
fetch = FetchType.EAGER
```




Lazy avoids fetching data you don't need. Eager avoids the N+1 problem if you know you need everything.

# Caching Level 1: The Session Cache

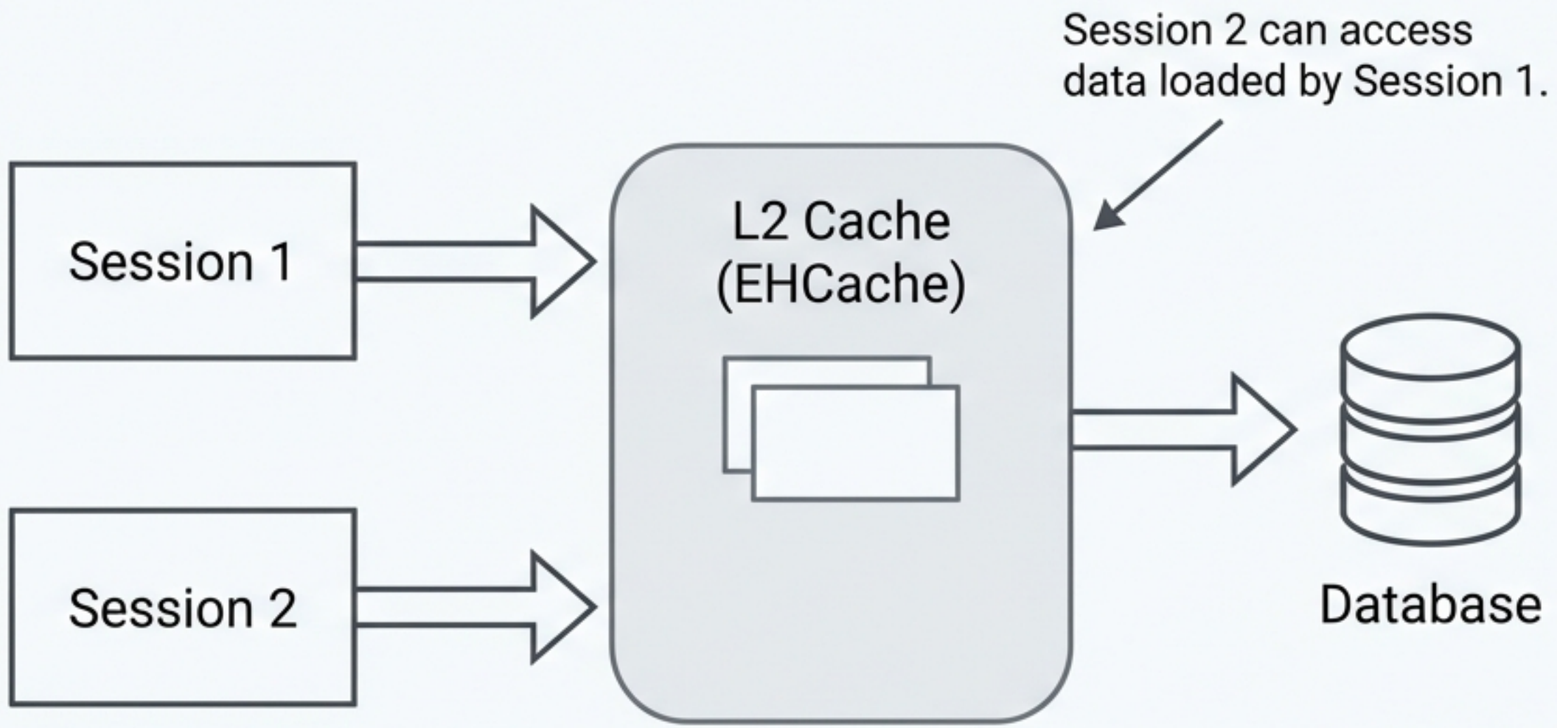
Enabled by default. Scoped to the transaction.




Limitation: Cache dies when the Session closes.

# Caching Level 2: Global Caching (EHCache)

Sharing data across multiple sessions.




## Configuration Checklist

1. Dependency:

```
hibernate-ehcache
```

2. Config:

```
hibernate.cache.use_second_
level_cache = true
```

3. Factory:

```
EhCacheRegionFactory
```

4. Annotation:

```
@Cacheable &
@Cache(usage=READ_ONLY)
```

# HQL: Hibernate Query Language

Think in Objects, not Tables.

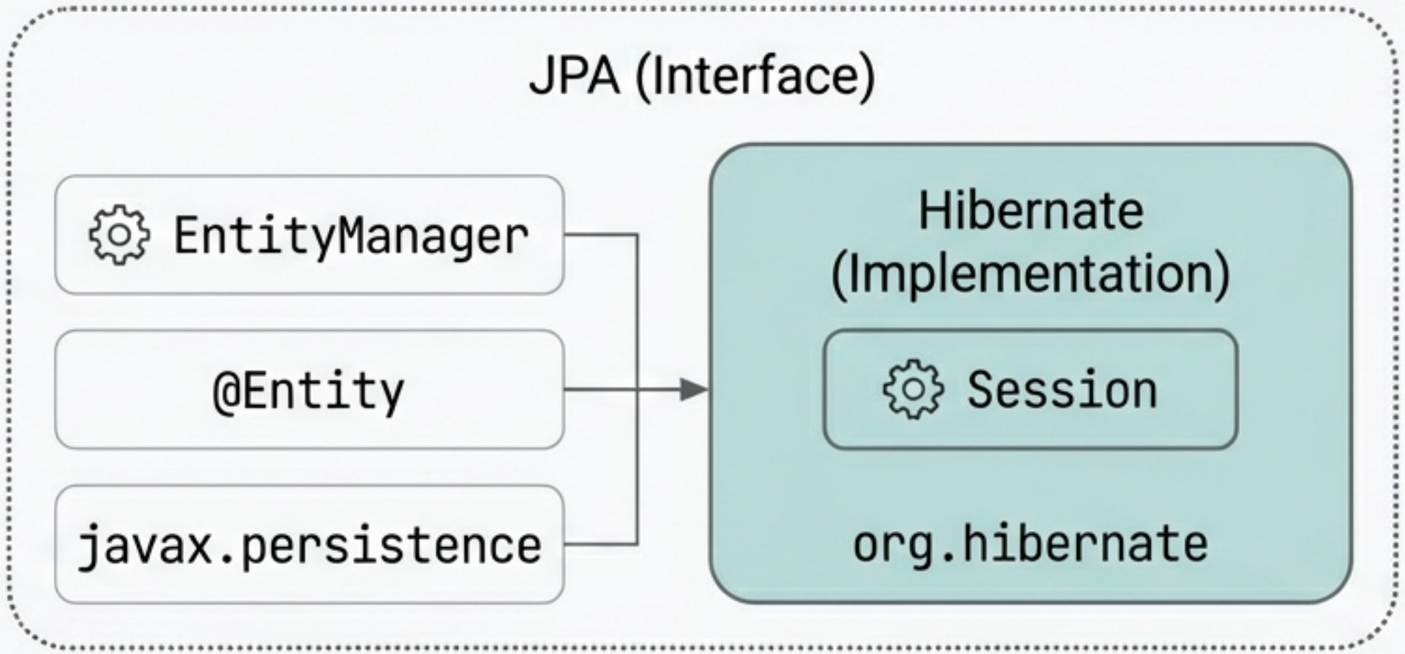
| Concept | SQL | HQL |
| --- | --- | --- |
| Targets Tables & Columns. | Targets Tables & Columns. | Targets Classes & Properties. |
| Syntax Example | `SELECT * FROM alien_table WHERE alien_points > 50` | `FROM Alien WHERE aids > 50` |
| Return Type | ResultSet (needs manual parsing). | `List<Alien>` (Ready-to-use objects). |

```
Query q = session.createQuery("from Alien where aids > 50");
List<Alien> aliens = q.list();
```

# The Future: Hibernate vs. JPA

Standardizing Persistence.

### Specification vs Implementation




### Migration Key Points

- Portability: Write for JPA, switch providers anytime.
- The Shift:

`SessionFactory` → `EntityManagerFactory`

`Session` → `EntityManager`

`hibernate.cfg.xml` → `persistence.xml`

“Mastering Hibernate makes mastering JPA trivial.”