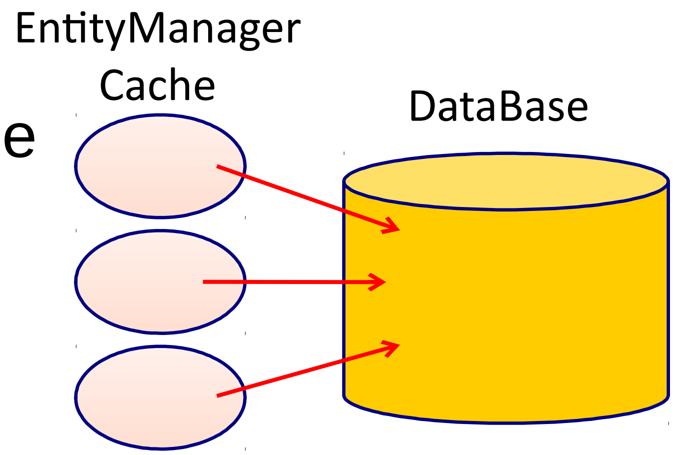
CS544 EA

Hibernate

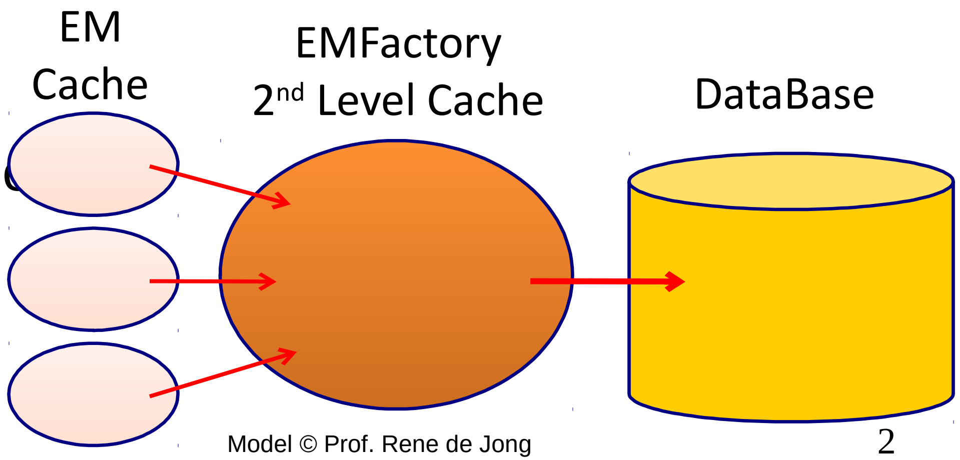
Optimization: 2nd Level Cache

2nd Level Caching

- By default JPA only uses EntityManager cache
  - Very short term cache

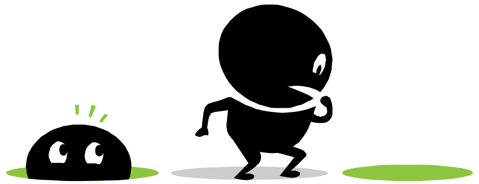


- To reduce hits on the DB
  - Objects can also be cached for longe
  - Managed by EntityManagerFactory
  - Shared by all EntityManagers



Caching VS Optimization

- Caching can be seen as a **form of scaling**
  - Doesn’t solve bad queries
  - But can alleviate pressure on the DB
- Caching is a large and interesting field
  - We will look at some basics
  - Be aware that **improper configuration** can create situations that are **hard to debug** (cached versions != DB versions)



What to cache?

- Good candidates for caching:
  - **Do not change**, or change rarely
  - Are modified only by your app
  - Are non-critical to the app

- Typically: **Reference data**

4 Caching Strategies

Stricter and therefore Slower $\downarrow$

- **Read Only**: very fast strategy, but can only be used for data that never changes
- **Non-Strict Read-Write**: data may be stale for a while, but gets refreshed at a timeout
- **Read-Write**: prevents stale data, but at a cost. Use for read-mostly data in a non-clustered setup
- **Transactional**: Can prevent stale data in a clustered environment. Can be used for read-mostly data

Cache Providers

- Hibernate can have **only one** provider per EMF

Provider	Read Only	Non Strict Read Write	Read Write	Transactional
EHCache	✓	✓	✓	
OSCache	✓	✓	✓	
SwarmCache	✓	✓		
JBoss Cache 1.x	✓			✓
JBoss Cache 2.x	✓			✓

Annotate Classes with Strategy

- Using Hibernate’s **@Cache** annotation

```
@Entity
@org.hibernate.annotations.Cache(usage=
    CacheConcurrencyStrategy.NONSTRICT_READ_WRITE
)
public class SalesRep {
  @Id
  @GeneratedValue
  private int id;
  private String name;

  @OneToMany(mappedBy="salesRep", cascade=CascadeType.PERSIST)
  private Set<Customer> customers = new HashSet<Customer>();

  ...
```

Setup Cache Provider

- Inside **persistence.xml**

```xml
<properties>
    <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/cs544?useSSL=false"/>
    <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
    <property name="javax.persistence.jdbc.user" value="root"/>
    <property name="javax.persistence.jdbc.password" value="root"/>
    <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />

    <!-- 2nd Level Caching -->
    <property name="hibernate.cache.provider_class" value="org.hibernate.cache.EhCacheProvider"/>
    <!-- To analyze cache performance -->
    <property name="hibernate.generate_statistics" value="true" />

    <property name="hibernate.show_sql" value="true" />
    <property name="hibernate.format_sql" value="true" />
    <property name="hibernate.id.new_generator_mappings" value="false" />
    <property name="hibernate.hbm2ddl.import_files" value="test.sql" />
    <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
</properties>
```

Configure Cache Provider

```
<ehcache>
  <diskStore path="java.io.tmpdir"/>
  <defaultCache
    maxElementsInMemory="10000"
    eternal="false"
    timeToIdleSeconds="120"
    timeToLiveSeconds="120"
    overflowToDisk="true" />

  <cache name="cacheDemo.Category"
    maxElementsInMemory="50"
    eternal="true"
    timeToIdleSeconds="0"
    timeToLiveSeconds="0"
    overflowToDisk="false" />

  <cache name="cacheDemo.Category.customers"
    maxElementsInMemory="50"
    eternal="false"
    timeToIdleSeconds="3600"
    timeToLiveSeconds="7200"
    overflowToDisk="false" />

  <cache name="cacheDemo.SalesRep"
    maxElementsInMemory="500"
    eternal="false"
    timeToIdleSeconds="1800"
    timeToLiveSeconds="10800"
    overflowToDisk="false" />
</ehcache>
```

General Config

Config for an Entity

Config for a Collection

Statistics

```
SessionFactory sessionFactory = emf.unwrap(SessionFactory.class);

Statistics stats = sessionFactory.getStatistics();
long hits   = stats.getSecondLevelCacheHitCount();
long misses = stats.getSecondLevelCacheMissCount();
long puts   = stats.getSecondLevelCachePutCount();
System.out.printf("\nGeneral 2nd Level Cache Stats\n");
System.out.printf("Hit: %d Miss: %d Put: %d\n", hits, misses, puts);

SecondLevelCacheStatistics salesRepStats =
    stats.getSecondLevelCacheStatistics("cacheDemo.SalesRep");
long srCurrent = salesRepStats.getElementCountInMemory();
long srMemsize = salesRepStats.getSizeInMemory();
long srHits    = salesRepStats.getHitCount();
long srMisses  = salesRepStats.getMissCount();
long srPuts    = salesRepStats.getPutCount();
System.out.printf("\nSalesRep Cache Region - Size: %d Holds: %d\n", srMemsize, srCurrent);
System.out.printf("Hit: %d Miss: %d Put: %d\n", srHits, srMisses, srPuts);
```

General 2nd level cache statistics

Statistics for a specific cache region

```
SessionFactory sessionFactory = emf.unwrap(SessionFactory.class);
Statistics stats = sessionFactory.getStatistics();
Stats.clear();
stats.setStatisticsEnabled(true);
...
stats.setStatisticsEnabled(false);
```

You can also programmatically turn stats on and off for more targeted measuring