CS544 EA

Hibernate

Common Problems

Problems

- The essence of ORM related problems are that the database is being **overloaded**
  - Doesn’t work properly anymore
  - Cannot handle the load

- Solution avenues are:
  - Lower the load by **using better techniques** (this chapter)
  - Spread the load by caching and scaling (we discuss caching)

Bad Queries

- The most common type of bad query is a **Cartesian Product**
  - **Caused by joining** 2 (or more) **collections**
  - Creates an ‘exploded’ resultset (takes the DB a long time)

- Hibernate will never generate such a query
  - Throws exception if 2 collections are set as eager on one Entity
  - But a (unaware) programmer can easily write such a query!

Code

Cust1 has 3 books and 3 movies
Cust2 1 book
Cust3 1 movie

```
Customer cust1 = new Customer("Frank", "Brown");
Customer cust2 = new Customer("Jane", "Terrien");
Customer cust3 = new Customer("John", "Doe");
cust1.addBook(new Book("Harry Potter and the Deathly Hallows"));
cust1.addBook(new Book("Unseen Academicals (Discworld)"));
cust1.addBook(new Book("The Color of Magic (Discworld)"));
cust1.addMovie(new Movie("Shrek"));
cust1.addMovie(new Movie("WALL-E"));
cust1.addMovie(new Movie("Howls Moving Castle"));
cust2.addBook(new Book("Twilight (The Twilight Saga, Book1)"));
cust3.addMovie(new Movie("Forgetting Sarah Marshall"));
em.persist(cust1);
em.persist(cust2);
em.persist(cust3);

em.getTransaction().commit();
em.clear();

em.getTransaction().begin();
TypedQuery<Customer> query = em.createQuery(
     "select c from Customer c join c.movies join c.books",
     Customer.class);
List<Customer> customers = query.getResultList();

em.getTransaction().commit();
```

Joining 2 collections

```
select
    customer0_.id as id1_1_,
    customer0_.firstName as firstNam2_1_,
    customer0_.lastName as lastName3_1_,
    customer0_.salesRep_id as salesRep4_1_
from
    Customer customer0_
inner join
    Movie movies1_
       on customer0_.id=movies1_.movies_id
inner join
    Book books2_
        on customer0_.id=books2_.books_id
```

Resultset

- Joining 2 collections **creates R x N x M rows**
  - R normal rows, N size of cllct. 1, M size of cllct. 2

FIRSTNAME0_0_	LASTNAME0_0_	TITLE1_1_	TITLE2_2_
Frank	Brown	Unseen Academicals (Discworld)	WALL-E
Frank	Brown	Unseen Academicals (Discworld)	Shrek
Frank	Brown	Unseen Academicals (Discworld)	Howls Moving Castle
Frank	Brown	The Color of Magic (Discworld)	WALL-E
Frank	Brown	The Color of Magic (Discworld)	Shrek
Frank	Brown	The Color of Magic (Discworld)	Howls Moving Castle
Frank	Brown	Harry Potter and the Deathly Hallows	WALL-E
Frank	Brown	Harry Potter and the Deathly Hallows	Shrek
Frank	Brown	Harry Potter and the Deathly Hallows	Howls Moving Castle
Jane	Terrien	Twilight (The Twilight Saga, Book1)	[null]
John	Doe	[null]	Forgetting Sarah Marshall

Redundancy

Very Inefficient!

27 cells to give
7 pieces of data

Frank Brown	Discworld	Pixar
Frank Brown	Discworld	Dream Works
Frank Brown	Discworld	Studio Ghibli
Frank Brown	Harry Potter	Pixar
Frank Brown	Harry Potter	Dream Works
Frank Brown	Harry Potter	Studio Ghibli
Frank Brown	Twilight	Pixar
Frank Brown	Twilight	Dream Works
Frank Brown	Twilight	Studio Ghibli

N + 1 Problem

- The N+1 problem is where Hibernate executes **many small selects** to load related data
  - This data could have been loaded in **one big select**

Much faster!

- People sometimes associate it with lazy loading
  - But happens with eager loading too!
  - It's Just Hibernate not knowing **how** to best load data

Lazy Collections N+1

- By default Hibernate lazily loads collections
  - A good default, they can contain a lot of data

- If we create **a query** for all SalesReps
  - Then **use a loop** to get the customers of those reps
  - 1 select for the salesreps (say there are 10)
  - 10 selects, one for each collection of customers

Code

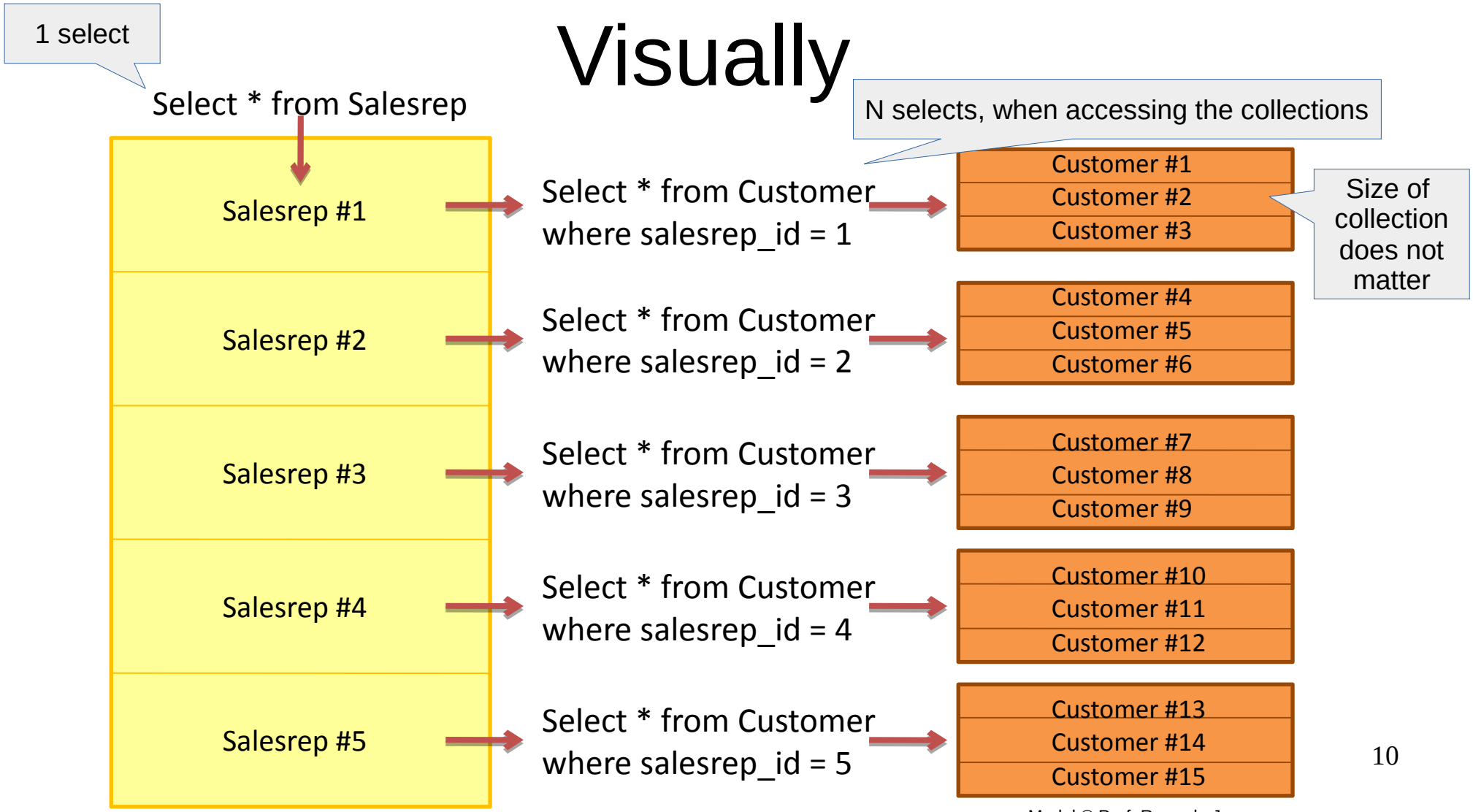
1 select for the salesreps
N selects for each
list of customers

```java
em.getTransaction().begin();

SalesRep sr1 = new SalesRep("John Willis");
SalesRep sr2 = new SalesRep("Mary Long");

sr1.addCustomer(new Customer("Frank", "Brown"));
sr1.addCustomer(new Customer("Jane", "Terrien"));
sr2.addCustomer(new Customer("John", "Doe"));
sr2.addCustomer(new Customer("Carol", "Reno"));

em.persist(sr1);
em.persist(sr2);
em.getTransaction().commit();
```

```java
TypedQuery<SalesRep> query =
          em.createQuery("from SalesRep", SalesRep.class);
List<SalesRep> salesReps = query.getResultList();
for(SalesRep s : salesReps) {
  System.out.println(s.getCustomers().get(0));
}
```

```
Hibernate:
    select
        salesrep0_.id as id1_1_,
        salesrep0_.name as name2_1_
    from
        SalesRep salesrep0_
Hibernate:
    select
        customers0_.salesRep_id as salesRep4_0_0_,
        customers0_.id as id1_0_0_,
        customers0_.id as id1_0_1_,
        customers0_.firstName as firstNam2_0_1_,
        customers0_.lastName as lastName3_0_1_,
        customers0_.salesRep_id as salesRep4_0_1_
    from
        Customer customers0_
    where
        customers0_.salesRep_id=?
Hibernate:
    select
        customers0_.salesRep_id as salesRep4_0_0_,
        customers0_.id as id1_0_0_,
        customers0_.id as id1_0_1_,
        customers0_.firstName as firstNam2_0_1_,
        customers0_.lastName as lastName3_0_1_,
        customers0_.salesRep_id as salesRep4_0_1_
    from
        Customer customers0_
    where
        customers0_.salesRep_id=?
```

Visually



Eager References N+1

- By default Hibernate uses eager loading for
  - @OneToOne and @ManyToOne
  - If eager associations are not yet fulfilled
    - Hibernate will execute select statements to fix it

Good policy.
Cost of joining a single row is low, and generally reduces selects

- If you execute **1 query** for all customers
  - Without Join Fetch-ing the @ManyToOne SalesRep
  - Hibernate will ‘fix’ this right away with **N extra selects**

References are Eager by default

Doesn’t even need a loop

Code

Each customer has its own Rep

```
em.getTransaction().begin();
Customer cust1 = new Customer("Frank", "Brown");
Customer cust2 = new Customer("Jane", "Terrien");
Customer cust3 = new Customer("John", "Doe");
Customer cust4 = new Customer("Carol", "Reno");
cust1.setSalesRep(new SalesRep("John Willis"));
cust2.setSalesRep(new SalesRep("Mary Long"));
cust3.setSalesRep(new SalesRep("Ted Walker"));
cust4.setSalesRep(new SalesRep("Keith Rogers"));

em.persist(cust1);
em.persist(cust2);
em.persist(cust3);
em.persist(cust4);
em.getTransaction().commit();
```

```
List<Customer> customers = em.createQuery(
     "from Customer").getResultList();
```

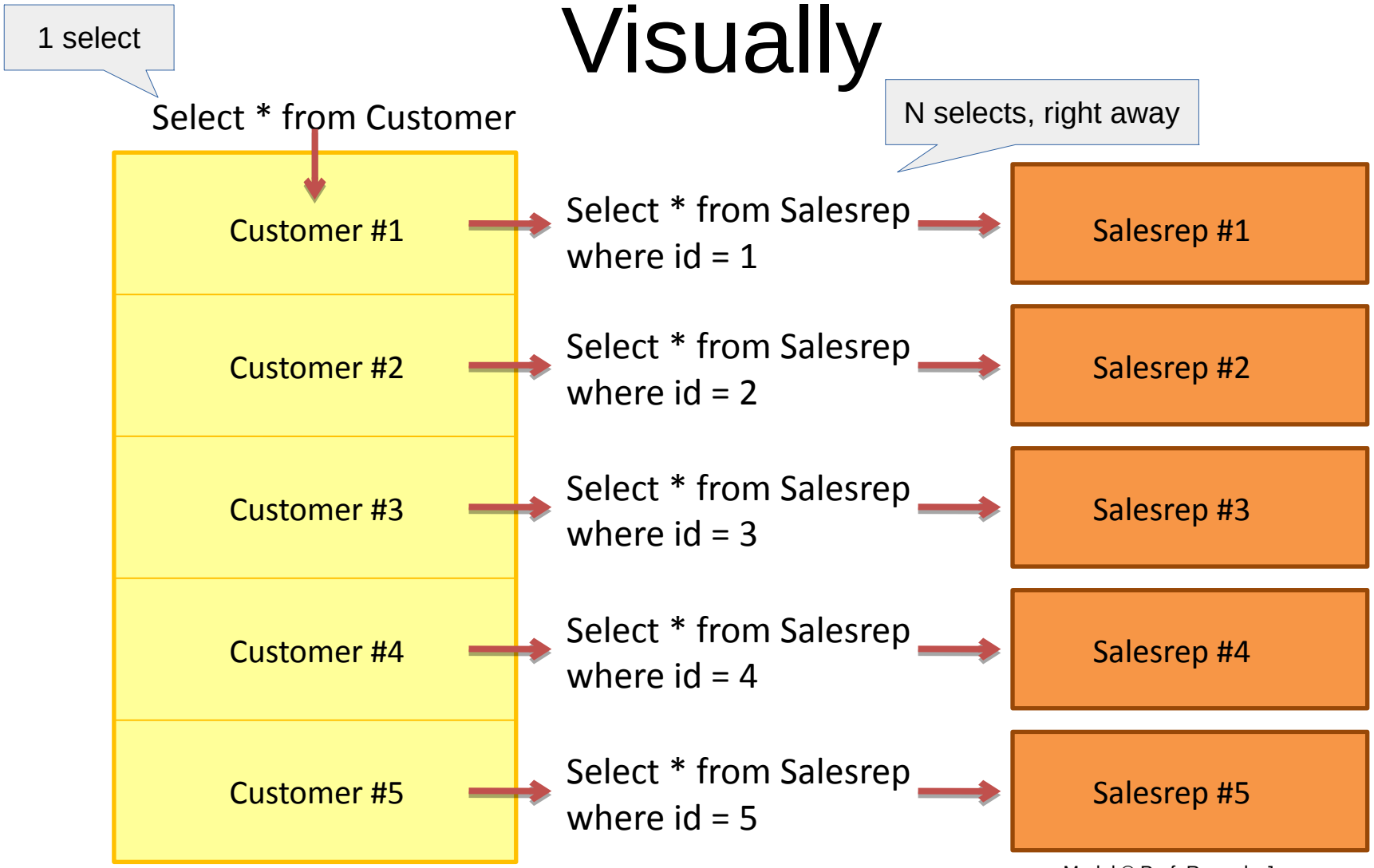
No loop or anything.

Hibernate executes the selects
to fix the missing eager references

1 select for customers,
N selects for the reps

```
Hibernate:
    select
        customer0_.id as id1_0_,
        customer0_.firstName as firstNam2_0_,
        customer0_.lastName as lastName3_0_,
        customer0_.salesRep_id as salesRep4_0_
    from
        Customer customer0_
Hibernate:
    select
        salesrep0_.id as id1_1_0_,
        salesrep0_.name as name2_1_0_
    from
        SalesRep salesrep0_
    where
        salesrep0_.id=?
Hibernate:
    select
        salesrep0_.id as id1_1_0_,
        salesrep0_.name as name2_1_0_
    from
        SalesRep salesrep0_
    where
        salesrep0_.id=?
Hibernate:
    select
        salesrep0_.id as id1_1_0_,
        salesrep0_.name as name2_1_0_
    from
        SalesRep salesrep0_
    where
```

Visually



Changing Doesn't Help

- Changing the references to LAZY
  - Just makes it so that Hibernate doesn't load the entities until you access them (with a loop)

```
@Entity
public class Customer {
    @Id
    @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;
    @ManyToOne(fetch=FetchType.LAZY)
    private SalesRep salesRep;
```

LAZY

- Similarly, changing the collection to EAGER
  - Makes the N selects happen right away
  - The **problem is not in WHEN, but HOW**

Solutions

- The solution for the **Cartesian product** is simple:
  - **Don’t join 2 or more** collections in one query
  - Join max 1, use separate queries for the others
  - Similarly other bad queries can be analyzed and fixed

- The solution for **N+1 is not that easy**
  - We’ll look at potential strategies coming up