Tune the App, Not the SQL

DBA Sherlock’s Adventures in Hibernate/jOOQ Land
It’s the App, not the SQL
Tuning Application Performance - DBA Sherlock’s View

- Perform tracing
- Rank SQL by elapsed time
- Recommend / perform
  - Rewrite of SQL
  - Change of access structures
  - Parameter changes
  - Using hints

*Hope that overall performance will improve :-)*)
OK but

... What’s Missing?

- We're tuning every statement in isolation ...

- What if application logic matters? Retrieving
  - WHAT
  - WHEN, and
  - HOW MUCH of it

- Credits to Stéphane Faroult for emphasizing this aspect
Sherlock: “What’s that application doing”

- SQL trace shows statements in order of execution
- But … all application processing remains unknown
- In a real-world application, it may be completely impossible to reconstruct the logic

- DBAs and developers need to work together :-)

Tune the App, not the SQL
Sample Use Case
Application Tuning Example – Data

- Sakila schema (https://dev.mysql.com/doc/sakila/en/)
- Ported to Oracle by Lukas Eder
  https://github.com/jOOQ/jOOQ/tree/master/jOOQ-examples/Sakila/oracle-sakila-db
- Used for its higher complexity compared to e.g. Oracle Order Entry
Application Tuning Example – Sakila Schema
Application Tuning Example – Use Case

- When a customer browses the film rental database, we want to display films that
  - have been watched by “similar” customers and
  - are in the customer’s preferred category

- Similarity for our purpose being defined as “living in the same country“

- Tailored to structure of Sakila example schema

- Designed to require several multi-table relationships
Sample Frameworks
Frameworks: jOOQ

- Lightweight, Active Record - style JDBC wrapper with advanced code generation capabilities
- Typesafe writing SQL using a fluent API

```java
private List<BigDecimal> getFilmIds(Connection conn, List<BigDecimal> similarCustomerIds) {
    List<BigDecimal> filmIds = DSL.using(conn, SQLDialect.ORACLE)
        .select(INVENTORY.FILM_ID)
        .from(INVENTORY)
        .join(RENTAL)
        .on(INVENTORY.INVENTORY_ID.equal(RENTAL.INVENTORY_ID))
        .where(RENTAL.CUSTOMER_ID.in(similarCustomerIds))
        .fetchInto(BigDecimal.class);
    return filmIds;
}
```
Frameworks: Hibernate

- Object-Relational Mapping Framework providing advanced features (e.g., multi-level caching, custom configuration of fetch plans and strategies)
- Keyword: “object-relational impedance mismatch“
- Entity relationships configured in the Java objects

```java
@OneToMany(cascade = CascadeType.ALL, mappedBy = "customerId")
private Set<Payment> paymentSet;

@JoinColumn(name = "ADDRESS_ID", referencedColumnName = "ADDRESS_ID")
@ManyToOne(optional = false)
private Address address;

@JoinColumn(name = "STORE_ID", referencedColumnName = "STORE_ID")
@ManyToOne(optional = false)
private Store storeId;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "customerId")
private Set<Rental> rentalSet;

public Customer() { }
```
Tuning jOOQ
jOOQ: Basic Implementation

Step 1: Load the target customer

```java
CustomerRecord customer = getCustomer(conn);

private CustomerRecord getCustomer(Connection conn) {
    return DSL using(conn, SQLDialect.ORACLE)
        .selectFrom(CUSTOMER)
        .where(CUSTOMER.CUSTOME...R_ID).equal(
            BigDecimal.valueOf(new Random().nextInt(CUSTOMER_MAX_ID) + 1))
        .fetchOne();
}
```
Step 2: Find similar customers

- 2a: get target customer’s country

```java
CountryRecord customerCountry = getCountry(conn, customer.getAddressId());

private CountryRecord getCountry(Connection conn, BigDecimal addressId) {
    return DSL.using(conn, SQLDialect.ORACLE)
        .select(COUNTRY.COUNTRY_ID, COUNTRY.COUNTRY_, COUNTRY.LAST_UPDATE)
        .from(ADDRESS)
        .join(CITY)
        .on(CITY.CITY_ID.equal(ADDRESS.CITY_ID))
        .join(COUNTRY)
        .on(COUNTRY.COUNTRY_ID.equal(CITY.COUNTRY_ID))
        .where(ADDRESS.ADDRESS_ID.equal(addressId))
        .fetchOneInto(CountryRecord.class);
}
```
Step 2: Find similar customers

- 2b: find customers living in same country

```java
List<BigDecimal> similarCustomerIds = getSimilarCustomers(conn, customerCountry.getCountryId());
```

```java
private List<BigDecimal> getSimilarCustomers(Connection conn, BigDecimal countryId)
{
    return DSL.using(conn, SQLDialect.ORACLE)
        .select(CUSTOMER.CUSTOMER_ID)
        .from(CUSTOMER)
        .join(ADDRESS)
        .on(CUSTOMER.ADDRESS_ID.equal(ADDRESS.ADDRESS_ID))
        .join(CITY)
        .on(CITY.CITY_ID.equal(ADDRESS.CITY_ID))
        .join(COUNTRY)
        .on(COUNTRY.COUNTRY_ID.equal(CITY.COUNTRY_ID))
        .where(COUNTRY.COUNTRY_ID.equal(countryId))
        .fetchInto(BigDecimal.class);
}
```
Step 3: find customer’s preferred category

```java
BigDecimal categoryId = getPreferredCategory(conn, customer);

private BigDecimal getPreferredCategory(Connection conn, CustomerRecord customer) {
    return DSL.using(conn, SQLDialect.ORACLE)
        .select(CATEGORY.CATEGORY_ID)
        .from(CATEGORY)
        .where(CATEGORY.CATEGORY_ID.equal(BigDecimal.valueOf(
            new Random().nextInt(CATEGORY_MAX_ID) + 1)))
        .fetchOneInto(BigDecimal.class);
}
```
Step 4: Identify films that have been watched by similar customers

List<BigDecimal> filmIds = getFilmIds(conn, similarCustomerIds);

private List<BigDecimal> getFilmIds(Connection conn, List<BigDecimal> similarCustomerIds) {
    List<BigDecimal> filmIds = DSL.using(conn, SQLDialect.ORACLE)
        .select(INVENTORY.FILM_ID)
        .from(INVENTORY)
        .join(RENTAL)
        .on(INVENTORY.INVENTORY_ID.equal(RENTAL.INVENTORY_ID))
        .where(RENTAL.CUSTOMER_ID.in(similarCustomerIds))
        .fetchInto(BigDecimal.class);
    return filmIds;
}
jOOQ: Basic Implementation

Step 5: Display films that match both criteria

List<FilmRecord> films = getFilmInfo(conn, filmIds, categoryId);

private List<FilmRecord> getFilmInfo(Connection conn, List<BigDecimal> filmIds, BigDecimal categoryId) {
    List<FilmRecord> films = DSL.using(conn, SQLDialect.ORACLE)
        .selectDistinct(FILM.FILM_ID, FILM.DESCRIPTION, FILM.LANGUAGE_ID, FILM.LAST_UPDATE, FILM.LENGTH, 
                        FILM.ORIGINAL_LANGUAGE_ID, FILM.RATING, FILM.RELEASE_YEAR, FILM.RENTAL_DURATION, 
                        FILM.RENTAL_RATE, FILM.REPLACEMENT_COST, FILM.SPECIAL_FEATURES, FILM.TITLE)
        .from(FILM)
        .join(FILMCATEGORY)
        .on(FILMCATEGORY.FILM_ID.equal(FILM.FILM_ID))
        .join(CATEGORY)
        .on(CATEGORY.CATEGORY_ID.equal(FILMCATEGORY.CATEGORY_ID))
        .where(FILM.FILM_ID.in(filmIds.stream().limit(1000).collect(Collectors.toList())))
        .and(CATEGORY.CATEGORY_ID.equal(categoryId))
        .fetchInto(FilmRecord.class);
    return films;
}
jOOQ: Basic Implementation – Performance (10000 executions)

Client execution time: 1,009 s

And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>60000</td>
<td>0.85</td>
<td>1.73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>60000</td>
<td>2.55</td>
<td>3.55</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>699761</td>
<td>67.39</td>
<td>98.27</td>
<td>0</td>
<td>3166345</td>
<td>0</td>
<td>6552953</td>
</tr>
<tr>
<td>total</td>
<td>819761</td>
<td>70.80</td>
<td>103.56</td>
<td>0</td>
<td>3166345</td>
<td>0</td>
<td>6552953</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times Waited</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
<td>699764</td>
<td>0.50</td>
<td>866.76</td>
</tr>
</tbody>
</table>
How long do the single steps take?

<table>
<thead>
<tr>
<th>Step</th>
<th>Elapsed Time (s)</th>
<th>No. Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get films from similar customers</td>
<td>69.597</td>
<td>10000</td>
</tr>
<tr>
<td>get customers in same country</td>
<td>16.368</td>
<td>10000</td>
</tr>
<tr>
<td>get films to display</td>
<td>15.639</td>
<td>10000</td>
</tr>
<tr>
<td>get customer country</td>
<td>0.737</td>
<td>10000</td>
</tr>
<tr>
<td>get target customer</td>
<td>0.716</td>
<td>10000</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>0.558</td>
<td>10000</td>
</tr>
</tbody>
</table>
jOOQ: Basic Implementation - Performance

- SQL statement execution takes MUCH less time than data transfer over network
- Any attempt at “SQL Tuning” will therefore not have overwhelming effects …
- Sherlock: “Could you do that with fewer data?”
- “Perhaps even with fewer statements?”
- “Sure let’s have a look …”
  - “We could integrate retrieving the customer’s country into the main get similar customers query …”
  - “We should be able to retrieve the data needed to display the films in one step, too”
List<BigDecimal> similarCustomerIds = getSimilarCustomers1Step(conn, customer.getAddressId());

private List<BigDecimal> getSimilarCustomers1Step(Connection conn, BigDecimal addressId) {
    Customer c1 = CUSTOMER.as("c1");
    Address a1 = ADDRESS.as("a1");
    City c11 = CITY.as("c11");
    Country c01 = COUNTRY.as("c01");
    Address a2 = ADDRESS.as("a2");
    City c12 = CITY.as("c12");
    Country c02 = COUNTRY.as("c02");
    return DSL.using(conn, SQLDialect.ORACLE)
        .select(c1.CUSTOMER_ID).from(c1)
        .join(a1).on(c1.ADDRESS_ID.equal(a1.ADDRESS_ID))
        .join(c11).on(c11.CITY_ID.equal(a1.CITY_ID))
        .join(c01).on(c01.COUNTRY_ID.equal(c11.COUNTRY_ID))
        .join(a2).on(c12.CITY_ID.equal(a2.CITY_ID))
        .where(a2.ADDRESS_ID.equal(addressId))
        .fetchInto(BigDecimal.class);
jOOQ: Refactoring 1 - Performance

- Client execution time: 970s
- And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>50000</td>
<td>0.77</td>
<td>1.54</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>50000</td>
<td>2.58</td>
<td>3.39</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>673321</td>
<td>64.90</td>
<td>94.96</td>
<td>0</td>
<td>3128232</td>
<td>0</td>
<td>6379218</td>
</tr>
<tr>
<td>total</td>
<td>773321</td>
<td>68.26</td>
<td>99.90</td>
<td>0</td>
<td>3128232</td>
<td>0</td>
<td>6379218</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times Waited</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
<td>673324</td>
<td>0.49</td>
<td>819.78</td>
</tr>
</tbody>
</table>
How long do the single steps take?

<table>
<thead>
<tr>
<th>Step</th>
<th>Elapsed Time (s)</th>
<th>No. Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get films from similar customers</td>
<td>67.894</td>
<td>10000</td>
</tr>
<tr>
<td>get similar customers, 1 step</td>
<td>15.447</td>
<td>10000</td>
</tr>
<tr>
<td>get films to display</td>
<td>15.322</td>
<td>10000</td>
</tr>
<tr>
<td>get target customer</td>
<td>0.713</td>
<td>10000</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>0.578</td>
<td>10000</td>
</tr>
</tbody>
</table>
**jOOQ: Refactoring 1 - Performance**

- Refactored version is faster by 1,685 ms
- The new version is even faster than part 2 of the original version…!
- Sherlock: “How is that even possible?”
- “Wildly differing execution plans?” -> No!
- But: In addition to expected measurement errors, there is variance due to random data selection
- Still, we didn’t gain much – no wonder: Room for improvement was small since part 1 of original version already was very fast (737 ms)
- Sherlock: “Room for improvement should be bigger with the second refactoring, then”
Step 4 new: Get films to display (matching all criteria) in 1 step

```java
List<FilmRecord> films = getFilmInfo1Step(conn, similarCustomerIds, categoryId);

private List<FilmRecord> getFilmInfo1Step(Connection conn, List<BigDecimal> similarCustomerIds, BigDecimal categoryId) {
    List<FilmRecord> films = DSL.using(conn, SQLDialect.ORACLE)
        .selectDistinct(FILM.FILM_ID, FILM.DESCRIPTION, FILM.LANGUAGE_ID, FILM.LAST_UPDATE, FILM.LENGTH, FILM.ORIGINAL_LANGUAGE_ID, FILM.RATING, FILM.RELEASE_YEAR, FILM.RENTAL_DURATION,
                       FILM.RENTAL_RATE, FILM.REPLACEMENT_COST, FILM.SPECIAL_FEATURES, FILM.TITLE)
        .from(FILM)
        .join(FILM_CATEGORY).on(FILM_CATEGORY.FILM_ID.equal(FILM.FILM_ID))
        .join(CATEGORY).on(CATEGORY.CATEGORY_ID.equal(FILM_CATEGORY.CATEGORY_ID))
        .join(INVENTORY).on(INVENTORY.FILM_ID.equal(FILM.FILM_ID))
        .join(RENTAL).on(RENTAL.INVENTORY_ID.equal(INVENTORY.INVENTORY_ID))
        .where(CATEGORY.CATEGORY_ID.equal(categoryId))
        .and(RENTAL.CUSTOMER_ID.in(similarCustomerIds))
        .fetchInto(FilmRecord.class);
    return films;
}
```
jOOQ: Refactoring 2 - Performance

- Client execution time: 206s
- And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>40000</td>
<td>0.43</td>
<td>1.10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>40000</td>
<td>0.94</td>
<td>1.65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>78621</td>
<td>70.99</td>
<td>78.17</td>
<td>0</td>
<td>2607753</td>
<td>0</td>
<td>482374</td>
</tr>
</tbody>
</table>

| total    | 158621| 72.36| 80.93   | 0    | 2607753 | 0       | 482374 |

Event waited on

<table>
<thead>
<tr>
<th>SQL*Net message from client</th>
<th>Times Waited</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>78624</td>
<td>0.48</td>
<td>106.78</td>
<td></td>
</tr>
</tbody>
</table>
jOOQ: Refactoring 2 - Performance

How long do the single steps take?

<table>
<thead>
<tr>
<th>Step</th>
<th>Elapsed Time (s)</th>
<th>No. Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get similar customers, 1 step</td>
<td>15.620</td>
<td>10000</td>
</tr>
<tr>
<td>get films to display, 1 step</td>
<td>63.993</td>
<td>10000</td>
</tr>
<tr>
<td>get target customer</td>
<td>0.788</td>
<td>10000</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>0.568</td>
<td>10000</td>
</tr>
</tbody>
</table>
jOOQ: Refactoring 2 - Performance

- This time, combining two application steps results in a speedup of 400%
- Where does the difference come from?

<table>
<thead>
<tr>
<th></th>
<th>Refactoring 1</th>
<th>Refactoring 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL elapsed time (s)</td>
<td>100</td>
<td>81</td>
</tr>
<tr>
<td>SQL*Net from client (s)</td>
<td>820</td>
<td>107</td>
</tr>
</tbody>
</table>
Tuning jOOQ: Summary
Tuning jOOQ: Summary

- Without even paying attention to details, original performance *was improved by 400% just changing the query logic in the application*.
- At the same time, refactoring 1 demonstrated that combining/condensing queries does not necessarily result in speedup.
- Instead of relying on intuition, the *actual performance should be checked on application and database sides*.
Tuning Hibernate
Step 1: Load the target customer

```java
Customer customer = getCustomer(em);

private Customer getCustomer( EntityManager em) {
    return em.find(Customer.class,
                    BigDecimal.valueOf(new Random().nextInt(CUSTOMER_MAX_ID) + 1));
}
```
Hibernate: Basic Implementation

- Step 2: Find similar customers
  - 2a: get target customer’s country

```java
Country customerCountry = getCountry(em, customer.getAddress().getAddressId());

private Country getCountry(EntityManager em, BigDecimal addressId) {
    return (Country) em.createQuery("select c from Country c join c.citySet ci join ci.addressSet a where a.addressId = " + addressId).getSingleResult();
}
```
Hibernate: Basic Implementation

- Step 2: Find similar customers
  - 2b: find customers living in same country

```java
List<Customer> similarCustomers = getSimilarCustomers(em, customerCountry);

private List<Customer> getSimilarCustomers(EntityManager em, Country country) {
    List<Customer> c = em.createQuery(
        "select c from Customer c join c.address a join a.city ci join ci.country co"
        + " where co.countryId =" + country.getCountryId()).getResultList();
    return c;
}
```
Hibernate: Basic Implementation

- Step 3: Find customer’s preferred category

```java
BigDecimal categoryId = getPreferredCategory(conn, customer);

private Category getPreferredCategory(EntityManager em, Customer customer) {
    return em.find(Category.class,
        BigDecimal.valueOf(new Random().nextInt(CATEGORY_MAX_ID) + 1));
}
```
Step 4: Identify films that have been watched by similar customers

```java
List<Inventory> inventories = getInventories(em, similarCustomers);

private List<Inventory> getInventories(EntityManager em, List<Customer> similarCustomers) {
    Query query = em.createQuery(
        "select i from Inventory i join i.rentalSet r where r.customerId in :ids"
    );
    query.setParameter("ids", similarCustomers);
    List<Inventory> inventories = query.getResultList();
    return inventories;
}
```
Hibernate: Basic Implementation

Step 5: Display films that match both criteria

```java
List<Film> films = getFilmInfo(em, inventories, category);

private List<Film> getFilmInfo(EntityManager em, List<Inventory> inventories, Category category) {
    Query query = em.createNativeQuery("select distinct f.* from film f join inventory i on (f.film_id=i.film_id) "+" join film_category fc on (f.film_id = fc.film_id) join category c on (c.category_id=fc.category_id)"+
                                      " where i.film_id in :f and c.name = :c",
                                      Film.class);
    query.setParameter("f", inventories.stream().map(
        i -> i.getFilm().getFilmId()).limit(1000).collect(Collectors.toList()));
    query.setParameter("c", category.getName());
    List<Film> films = query.getResultList();
    return films;
}
```
Client execution time: 1,616s

And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>41577</td>
<td>2.54</td>
<td>3.34</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>41577</td>
<td>5.49</td>
<td>6.29</td>
<td>0</td>
<td>280</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>667636</td>
<td>88.84</td>
<td>121.36</td>
<td>0</td>
<td>3825253</td>
<td>0</td>
<td>6393616</td>
</tr>
<tr>
<td>total</td>
<td>750190</td>
<td>96.88</td>
<td>131.00</td>
<td>0</td>
<td>3825559</td>
<td>0</td>
<td>6393616</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times Waited</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
<td>667151</td>
<td>0.76</td>
<td>1432.10</td>
</tr>
</tbody>
</table>
# Hibernate: Basic Implementation - Performance

<table>
<thead>
<tr>
<th>Step</th>
<th>Distinct SQL IDs</th>
<th>Elapsed Time (s)</th>
<th>No. Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get films from similar customers</td>
<td>21</td>
<td>69.837</td>
<td>10000</td>
</tr>
<tr>
<td>get films to display</td>
<td>69</td>
<td>43.644</td>
<td>10000</td>
</tr>
<tr>
<td>get customers in same country</td>
<td>108</td>
<td>15.661</td>
<td>10000</td>
</tr>
<tr>
<td>get customer country</td>
<td>599</td>
<td>2.318</td>
<td>10000</td>
</tr>
<tr>
<td>select film0_.FILM_ID as FILM_ID1_7_0_, film0_.DESCRIPTION</td>
<td>1</td>
<td>0.095</td>
<td>958</td>
</tr>
<tr>
<td>get target customer</td>
<td>1</td>
<td>0.078</td>
<td>108</td>
</tr>
<tr>
<td>select address0_.ADDRESS_ID as ADDRESS_ID1_2_0_, address0_.A</td>
<td>1</td>
<td>0.042</td>
<td>491</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>1</td>
<td>0.007</td>
<td>15</td>
</tr>
<tr>
<td>select store0_.STORE_ID as STORE_ID1_16_0_, store0_.ADDRESS</td>
<td>1</td>
<td>0.005</td>
<td>1</td>
</tr>
<tr>
<td>select city0_.CITY_ID as CITY_ID1_4_0_, city0_.CITY as CITY2</td>
<td>1</td>
<td>0.001</td>
<td>1</td>
</tr>
</tbody>
</table>
OK Sherlock … What Is Going On?

- SQL trace shows
  - order of execution (raw trace file)
  - execution counts (processed)

- "What can we deduce?"
Time for some detective work!
And DBA Sherlock Investigates… [Mystery SQLs, 1/4]

```
SELECT city0_.CITY_ID AS CITY_ID1_4_0_, city0_.CITY AS CITY2_4_0_,
       city0_.COUNTRY_ID AS COUNTRY_ID4_4_0_, city0_.LAST_UPDATE AS
       LAST_UPDATE3_4_0_, country1_.COUNTRY_ID AS COUNTRY_ID1_5_1_,
       country1_.COUNTRY AS COUNTRY2_5_1_, country1_.LAST_UPDATE AS
       LAST_UPDATE3_5_1_
FROM CITY city0_.inner join COUNTRY country1_ on city0_.COUNTRY_ID=
   country1_.COUNTRY_ID where city0_.CITY_ID=1
```

- **Occurs:** exactly one time, on the first iteration, directly before the `get country` query
- **WHY?** -> <no solution>
- **Problem?** -> No: query executed just once
And DBA Sherlock Investigates… [Mystery SQLs, 2/4]

```sql
select store0_.STORE_ID as STORE_ID1_16_0_, store0_.ADDRESS_ID as ADDRESS_ID3_16_0_, store0_.LAST_UPDATE as LAST_UPDATE2_16_0_, 
store0_.MANAGER_STAFF_ID as MANAGER_STAFF_ID4_16_0_, address1_.ADDRESS_ID as ADDRESS_ID1_2_1_, address1_.ADDRESS as ADDRESS2_2_1_, address1_.ADDRESS2
...
from
store0_.inner join ADDRESS address1_ on store0_.ADDRESS_ID= 
address1_.ADDRESS_ID inner join CITY city2_ on address1_.CITY_ID= 
city2_.CITY_ID inner join COUNTRY country3_ on city2_.COUNTRY_ID= 
country3_.COUNTRY_ID inner join STAFF staff4_ on store0_.MANAGER_STAFF_ID= 
staff4_.STAFF_ID inner join ADDRESS address5_ on staff4_.ADDRESS_ID= 
address5_.ADDRESS_ID inner join STORE store6_ on staff4_.STORE_ID= 
store6_.STORE_ID where store0_.STORE_ID=1
```

- occurs exactly one time, on the first iteration, in between several instances of the *get films from similar customers* query
- WHY?  -> <no solution>
- Problem?  -> No: query executed just once
And DBA Sherlock Investigates… [Mystery SQLs, 3/4]

```sql
select address0_.ADDRESS_ID as ADDRESS_ID1_2_0_, address0_.ADDRESS as ADDRESS2_2_0_, address0_.ADDRESS2 as ADDRESS3_2_0_, address0_.CITY_ID as CITY_ID8_2_0, address0_.DISTRICT as DISTRICT4_2_0_, address0_.LAST_UPDATE as LAST_UPDATE5_2_0_, address0_.PHONE as PHONE6_2_0_, address0_.POSTAL_CODE as POSTAL_CODE7_2_0_, city1_.CITY_ID as CITY_ID4_4_1_, city1_.CITY as CITY2_4_1_, city1_.COUNTRY_ID as COUNTRY_ID3_4_1_, city1_.LAST_UPDATE as LAST_UPDATE3_4_1_, country2_.COUNTRY_ID as COUNTRY_ID5_5_2_, country2_.COUNTRY as COUNTRY2_5_2_, country2_.LAST_UPDATE as LAST_UPDATE3_5_2_
from
ADDRESS address0_ inner join CITY city1_ on address0_.CITY_ID=city1_.CITY_ID inner join COUNTRY country2_ on city1_.COUNTRY_ID=country2_.COUNTRY_ID
where address0_.ADDRESS_ID=:1
```

- **occurs 491 times, directly before the** *get preferred category* **query**
- **WHY?**  -> <no solution>
- **Problem?**  -> No: takes just 42 ms overall
select film0_.FILM_ID as FILM_ID1_7_0 , film0_.DESCRIPTION as DESCRIPTION2_7_0 , film0_.LANGUAGE_ID as LANGUAGE_ID12_7_0 ,
(...) 
TITLE1_7_0 , language1_.LANGUAGE_ID as LANGUAGE_ID1_0_1 ,
language1_.LAST_UPDATE as LAST_UPDATE2_0_1 , language1_.NAME as NAME3_0_1 ,
language2_.LANGUAGE_ID as LANGUAGE_ID1_0_2 , language2_.LAST_UPDATE as LAST_UPDATE2_0_2 , language2_.NAME as NAME3_0_2
from FILM film0_ inner join "LANGUAGE" language1_ on film0_.LANGUAGE_ID=
language1_.LANGUAGE_ID left outer join "LANGUAGE" language2_ on
film0_.ORIGINAL_LANGUAGE_ID=language2_.LANGUAGE_ID where film0_.FILM_ID=1

occurs 958 times, after the get films from similar customers query

WHY? ... wait ...
And DBA Sherlock Investigates… [Mystery SQLs, 4/4]

“show me that code again”

```java
Query query = em.createNativeQuery(
    "select distinct f.* from film f join inventory i on (f.film_id=i.film_id) "+
    "join film_category fc on (f.film_id = fc.film_id) join category c on (c.category_id=fc.category_id)"
    + " where i.film_id in :f and c.name= :c",
    Film.class);
query.setParameter("f", inventories.stream().map(
    i -> i.getFilm().getFilmId()).limit(1000).collect(Collectors.toList()));
```

Yaaay …!

OK but … what exactly is Hibernate doing here?
And DBA Sherlock Investigates… [Mystery SQLs, 4/4]

- Getting films …
- … inner joining LANGUAGE b/c of NOT NULL foreign key constraint from FILM.LANGUAGE_ID
- … outer joining LANGUAGE b/c of NULLABLE foreign key constraint from FILM.ORIGINAL_LANGUAGE_ID
- “This should disappear automagically with the planned refactorings …”
And DBA Sherlock Investigates… [Execution Counts]

- Two of the “regular” / expected queries are not executed 10000, but
  - 108 (get target customer) resp.
  - 15 times (get customer preferred category)
- Which is once per distinct category / customer …
- “There must be some Hibernate caching going on”
“Wait ... and what is THIS?”

This is the get target customer query ... 

```sql
select customer0_.CUSTOMER_ID as CUSTOMER_ID1_6_0_, customer0_.ACTIVE as ACTIVE2_6_6_, customer0_.ADDRESS_ID as ADDRESS_ID8_6_0_,
(...)
from
CUSTOMER customer0_ inner join ADDRESS address1_ on customer0_.ADDRESS_ID=address1_.ADDRESS_ID inner join CITY city2_ on address1_.CITY_ID=city2_.CITY_ID inner join COUNTRY country3_ on city2_.COUNTRY_ID=country3_.COUNTRY_ID inner join STORE store4_ on customer0_.STORE_ID=store4_.STORE_ID inner join ADDRESS address5_ on store4_.ADDRESS_ID=address5_.ADDRESS_ID inner join STAFF staff6_ on store4_.MANAGER_STAFF_ID=staff6_.STAFF_ID inner join ADDRESS address7_ on staff6_.ADDRESS_ID=address7_.ADDRESS_ID inner join STORE store8_ on staff6_.STORE_ID=store8_.STORE_ID where customer0_.CUSTOMER_ID=1
```

“but we just want (and need!) the customer ...?”
“Wait ... and what is THIS?”

All joins are due to many-to-one relationships ...

```
select customer0_.CUSTOMER_ID as CUSTOMER_ID1_6_0, customer0_.ACTIVE as ACTIVE2_6_0, customer0_.ADDRESS_ID as ADDRESS_ID8_6_0,  
(...)
from CUSTOMER customer0_  
inner join ADDRESS address1_ on customer0_.ADDRESS_ID=address1_.ADDRESS_ID  
inner join CITY city2_ on address1_.CITY_ID=city2_.CITY_ID  
inner join COUNTRY country3_ on city2_.COUNTRY_ID=country3_.COUNTRY_ID  
inner join STORE store4_ on customer0_.STORE_ID=store4_.STORE_ID  
inner join ADDRESS address5_ on store4_.ADDRESS_ID=address5_.ADDRESS_ID  
inner join STAFF staff6_ on store4_.MANAGER_STAFF_ID=staff6_.STAFF_ID  
inner join ADDRESS address7_ on staff6_.ADDRESS_ID=address7_.ADDRESS_ID  
inner join STORE store8_ on staff6_.STORE_ID=store8_.STORE_ID  
where customer0_.CUSTOMER_ID=1
```

“This looks like a quick win”
Step 1 new: Load the target customer – and just the target customer

```java
Customer customer = getCustomer(em);

private Customer getCustomer(EntityManager em) {
    return (Customer) em.createQuery("select c from Customer c where c.customerId = "+ BigDecimal.valueOf(new Random().nextInt(CUSTOMER_MAX_ID) + 1))
    .singleResult();
}
```
Hibernate: Refactoring 1 - Performance

- Client execution time: 1,755s
- And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>51577</td>
<td>3.08</td>
<td>4.33</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>51577</td>
<td>5.52</td>
<td>6.35</td>
<td>0</td>
<td>280</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>684620</td>
<td>87.80</td>
<td>126.18</td>
<td>0</td>
<td>3861110</td>
<td>0</td>
<td>6478751</td>
</tr>
<tr>
<td>total</td>
<td>787774</td>
<td>96.42</td>
<td>130.88</td>
<td>0</td>
<td>3961416</td>
<td>0</td>
<td>6478751</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
<td>684620</td>
<td>0.76</td>
<td>1570.22</td>
</tr>
</tbody>
</table>
## Hibernate: Refactoring 1 - Performance

<table>
<thead>
<tr>
<th>Step</th>
<th>Distinct SQL_IDs</th>
<th>Elapsed Time (s)</th>
<th>No. Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get films from similar customers</td>
<td>21</td>
<td>69.242</td>
<td>10000</td>
</tr>
<tr>
<td>get films to display</td>
<td>69</td>
<td>43.510</td>
<td>10000</td>
</tr>
<tr>
<td>get customers in same country</td>
<td>108</td>
<td>15.208</td>
<td>10000</td>
</tr>
<tr>
<td>get customer country</td>
<td>599</td>
<td>2.296</td>
<td>10000</td>
</tr>
<tr>
<td><strong>get target customer NEW</strong></td>
<td><strong>599 (was: 1)</strong></td>
<td><strong>1.450 (was: 78)</strong></td>
<td><strong>10000 (was: 108)</strong></td>
</tr>
<tr>
<td>select film0_.FILM_ID as FILM_ID1_7_0, film0_.DESCRIPTION</td>
<td>1</td>
<td>0.087</td>
<td>958</td>
</tr>
<tr>
<td>select address0_.ADDRESS_ID as ADDRESS_ID1_2_0, address0_.A</td>
<td>1</td>
<td>0.070</td>
<td>599</td>
</tr>
<tr>
<td>select store0_.STORE_ID as STORE_ID1_16_0, store0_.ADDRESS_</td>
<td>1</td>
<td>0.025</td>
<td>1</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>1</td>
<td>0.006</td>
<td>15</td>
</tr>
<tr>
<td>select city0_.CITY_ID as CITY_ID1_4_0, city0_.CITY as CITY2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hibernate: Refactoring 1 - Summary

- Reduced response time per execution gets counteracted by much higher execution count – no net wins
- But DBA Sherlock spotted something else now …
- “Why does the number of distinct SQL IDs increase?”
- “Let’s make sure we’re using bind variables everywhere”
Steps 1/2 new: Use bind variables instead of literals

```java
private Customer getCustomer( EntityManager em ) {
    return ( Customer ) em.createQuery( "select c from Customer c"
        + " where c.customerId = :cstid"
    ).setParameter( "cstid", // BigDecimal.valueOf( new Random().nextInt( CUSTOMER_MAX_ID ) + 1 )
        .getSingleResult();
}

private Country getCountry( EntityManager em ) {
    return ( Country ) em.createQuery( "select c from Country c join c.citySet"
        + " ci join ci.addressSet a where a.addressId = :adr_id"
    ).setParameter( "adr_id", addressId ).getSingleResult();
}

private List<Customer> getSimilarCustomers( EntityManager em, Country country ) {
    List<Customer> c = em.createQuery( "select c from Customer c join c.address a "
        + " join a.city ci join ci.country co where co.countryId = :ctr_id"
    ).setParameter( "ctr_id", country.getCountryId() )
        .getResultList();
    return c;
}
```
Hibernate: Refactoring 2 - Performance

- Client execution time: 1,770s
- And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>51577</td>
<td>1.15</td>
<td>1.96</td>
<td>6</td>
<td>26</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>51577</td>
<td>5.44</td>
<td>6.48</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>580318</td>
<td>90.39</td>
<td>127.22</td>
<td>0</td>
<td>386677</td>
<td>0</td>
<td>6434677</td>
</tr>
<tr>
<td>total</td>
<td>783472</td>
<td>96.98</td>
<td>135.67</td>
<td>0</td>
<td>3866803</td>
<td>0</td>
<td>6434677</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times Waited</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
<td>686325</td>
<td>0.78</td>
<td>1576.57</td>
</tr>
</tbody>
</table>
### Hibernate: Refactoring 2 - Performance

<table>
<thead>
<tr>
<th>Step</th>
<th>Distinct SQL_IDS</th>
<th>Elapsed Time (s)</th>
<th>No. Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get films from similar customers</td>
<td>21</td>
<td>70.213</td>
<td>10000</td>
</tr>
<tr>
<td>get films to display</td>
<td>69</td>
<td>42.786</td>
<td>10000</td>
</tr>
<tr>
<td>get customers in same country bind var</td>
<td>1</td>
<td>16.165 (was: 15.208)</td>
<td>10000</td>
</tr>
<tr>
<td>get customer country bind var</td>
<td>1</td>
<td>0.946 (was: 2.296)</td>
<td>10000</td>
</tr>
<tr>
<td>get target customer NEW bind var</td>
<td>1</td>
<td>0.771 (was: 1.450)</td>
<td>10000</td>
</tr>
<tr>
<td>select film0_.FILM_ID as FILM_ID1_7_0_, film0_.DESCRIPTION</td>
<td>1</td>
<td>0.089</td>
<td>958</td>
</tr>
<tr>
<td>select address0_.ADDRESS_ID as ADDRESS_ID1_2_0_, address0_.A</td>
<td>1</td>
<td>0.066</td>
<td>599</td>
</tr>
<tr>
<td>select store0_.STORE_ID as STORE_ID1_16_0_, store0_.ADDRESS_</td>
<td>1</td>
<td>0.028</td>
<td>2</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>1</td>
<td>0.006</td>
<td>15</td>
</tr>
</tbody>
</table>
Hibernate: Refactoring 2 - Summary

- Reduced parse time noticeable (only) for statements that don’t retrieve much data
- “After this cleanup, let’s do the main refactorings”
Hibernate: Refactoring 3

Step 2 new: Find similar customers in 1 step

```java
List<Customer> similarCustomers = getSimilarCustomers1Step(em, customer.getAddress().getAddressId());

private List<Customer> getSimilarCustomers1Step(EntityManager em, BigDecimal addressId) {
    Query query = em.createNativeQuery("select c1.customer_id, c1.store_id, c1.first_name, c1.last_name," + "c1.email, c1.address_id, c1.active, c1.create_date, c1.last_update " + "from customer c1" + "  join address a1 on c1.address_id=a1.address_id" + "  join city cil on cil.city_id = a1.city_id" + "  join country col on col.country_id=cil.country_id" + "  join country co2 on col.country_id = co2.country_id" + "  join city ci2 on co2.country_id=ci2.country_id" + "  join address a2 on ci2.city_id=a2.city_id" + "  WHERE a2.address_id = :a_id", Customer.class);
    query.setParameter("a_id", addressId);
    List<Customer> c = query.getResultList();
    return c;
}
```
Client execution time: 1,749s

And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>41577</td>
<td>1.06</td>
<td>1.74</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>41577</td>
<td>5.82</td>
<td>6.55</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>685422</td>
<td>85.19</td>
<td>117.90</td>
<td>0</td>
<td>3896141</td>
<td>0</td>
<td>6576643</td>
</tr>
</tbody>
</table>

Total: 768576 | 92.08 | 126.20 | 0 | 3896167 | 0 | 6576643

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times Waited</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
<td>685430</td>
<td>0.72</td>
<td>1576.53</td>
</tr>
</tbody>
</table>
## Hibernate: Refactoring 3 - Performance

<table>
<thead>
<tr>
<th>Step</th>
<th>Distinct SQL_IDs</th>
<th>Elapsed Time (s)</th>
<th>No. Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get films from similar customers</td>
<td>21</td>
<td>71.339</td>
<td>10000</td>
</tr>
<tr>
<td>get films to display</td>
<td>69</td>
<td>44.003</td>
<td>10000</td>
</tr>
<tr>
<td><strong>get similar customers 1 step</strong></td>
<td>1</td>
<td>9.933 (was: 16.165 + 946)</td>
<td>10000</td>
</tr>
<tr>
<td>get target customer NEW bind var</td>
<td>1</td>
<td>0.803</td>
<td>10000</td>
</tr>
<tr>
<td>select film0_.FILM_ID as FILM_ID1_7_0_, film0_.DESCRIPTION</td>
<td>1</td>
<td>0.090</td>
<td>958</td>
</tr>
<tr>
<td>select address0_.ADDRESS_ID as ADDRESS_ID1_2_0_, address0_.A</td>
<td>1</td>
<td>0.068</td>
<td>599</td>
</tr>
<tr>
<td>select store0_.STORE_ID as STORE_ID1_16_0_, store0_.ADDRESS_</td>
<td>1</td>
<td>0.026</td>
<td>2</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>1</td>
<td>0.007</td>
<td>15</td>
</tr>
</tbody>
</table>
Hibernate: Refactoring 3 - Summary

- Same as with jOOQ, the refactored query is faster than both original queries taken together.
- Again same as with jOOQ, there is no overall win as overall response time is dominated by the display films queries …
- “Let’s approach the decisive step …”
Hibernate: Refactoring 4: Combining Steps 4 & 5

Step 4 new: Get films to display (matching all criteria) in 1 step

List<Film> films = getFilmInfolStep(em, similarCustomers, category);

private List<Film> getFilmInfolStep(EntityManager em, List<Customer> similarCustomers, Category category) {

    Query query = em.createNativeQuery("select distinct f.* from film f 
    + "join film_category fc on (f.film_id = fc.film_id) join category c on (c.category_id=fc.category_id)"
    + " join inventory i on (f.film_id = i.film_id) join rental r on r.inventory_id = i.inventory_id"
    + " where r.customer_id in :s and c.name= :c", Film.class);
    query.setParameter("s", similarCustomers);
    query.setParameter("c", category.getName());
    List<Film> films = query.getResultList();
    return films;
Hibernate: Refactoring 4 - Performance

Client execution time: 487s

And the database says...

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>30620</td>
<td>0.56</td>
<td>1.12</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>30620</td>
<td>1.46</td>
<td>2.10</td>
<td>0</td>
<td>135</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>68511</td>
<td>91.83</td>
<td>99.82</td>
<td>0</td>
<td>2153157</td>
<td>0</td>
<td>466238</td>
</tr>
<tr>
<td>total</td>
<td>129751</td>
<td>93.86</td>
<td>103.06</td>
<td>0</td>
<td>2153318</td>
<td>0</td>
<td>466238</td>
</tr>
</tbody>
</table>

Elapsed times include waiting on following events:

<table>
<thead>
<tr>
<th>Event waited on</th>
<th>Times Waited</th>
<th>Max. Wait</th>
<th>Total Waited</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
<td>68519</td>
<td>0.72</td>
<td>365.45</td>
</tr>
<tr>
<td>Step</td>
<td>Distinct SQL_IDs</td>
<td>Elapsed Time (s)</td>
<td>No. Executions</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------</td>
<td>----------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>get films to display 1 step</td>
<td>21</td>
<td>86.326 (was: 71.339 + 44.003)</td>
<td>10000</td>
</tr>
<tr>
<td>get similar customers 1 step</td>
<td>1</td>
<td>15.894</td>
<td>10000</td>
</tr>
<tr>
<td>get target customer NEW bind var</td>
<td>1</td>
<td>0.792</td>
<td>10000</td>
</tr>
<tr>
<td>select address0_.ADDRESS_ID as ADDRESS_ID1_2_0_. address0_.A</td>
<td>1</td>
<td>0.084</td>
<td>599</td>
</tr>
<tr>
<td>select store0_.STORE_ID as STORE_ID1_16_0_. store0_.ADDRESS_</td>
<td>1</td>
<td>0.025</td>
<td>2</td>
</tr>
<tr>
<td>get customer preferred category</td>
<td>1</td>
<td>0.006</td>
<td>15</td>
</tr>
</tbody>
</table>
Hibernate: Refactoring 4 - Summary

- With Hibernate, the achieved speedup is at 230%.
- With Hibernate, speedup is even more due to reduced network traffic than with jOOQ:

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Refactoring 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL elapsed time (s)</td>
<td>131</td>
<td>103</td>
</tr>
<tr>
<td>SQL*Net from client (s)</td>
<td>1432</td>
<td>365</td>
</tr>
</tbody>
</table>
Case over - time for lessons learned ...
Lessons learned …

- Don’t just tune SQL, tune the application
- Don’t assume that reduced number of steps / increased complexity of SQL will *always* lead to speedup
- “*Sherlock* wait … *informant’s* calling …”
Lessons learned ...

- "For Hibernate, did you try read-only queries? No need for dirty checking when you don’t do modifications …"
- "Watson, we have work to do …"
Watch out for upcoming season 2 on www.trivadis.com
Thank you!

Sigrid Keydana

Tech Event, February 28 2016