jOOQ: Data Abstractions Without Distraction

Jason Lee

Principal Software Engineer, Red Hat

April 2022

Who Am I?

- Software developer since 1997
- Principal Software Engineer at Red Hat
    - WildFly/EAP
    - Undertow
- President Oklahoma City Java Users Group
- Book Author - Java 9 Programming Blueprints
- Blogger - https://jasondl.ee

What is jOOQ?

- “[O]riginally…created as a library for complete abstraction of JDBC and all database interaction”
- Type-safe SQL building
- SQL dialect abstraction
- Improved query execution and data retrieval API
- Active Records
- So much more…

jOOQ Alternatives

Lots of options in the Java ecosystem:

- JDBC
- Spring JdbcTemplate/Data
- Jdbi
- Hibernate/JPA
- MyBatis
- ...

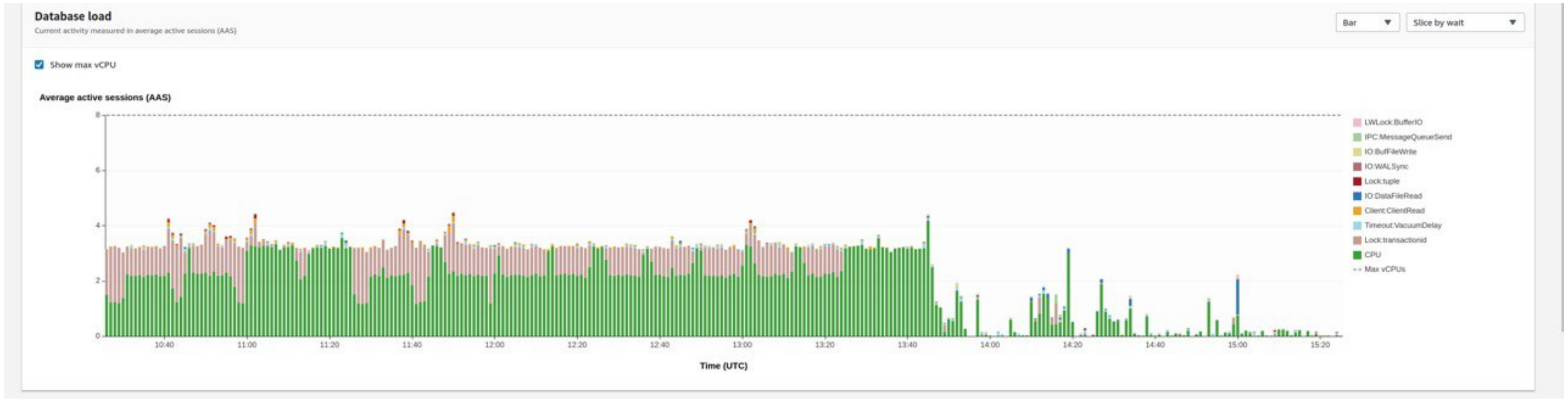
If any of these work for you, carry on!

However...

jOOQ Performance

Average active sessions before/after migrating from Hibernate to jOOQ.

"Past performance is no guarantee of future results," of course.



source

Supported Databases

What databases are supported?

All of them!

Supported Databases (cont)

- Open Source and Commercial licenses
- Open Source license
  - PostgreSQL
  - MySQL
  - MariaDB
  - Firebird
  - Derby
  - H2
  - SQLite
- Commercial (three tiers: Express, Professional, and Enterprise)
  - MS Access
  - Oracle
  - SQL Server
  - Redshift
  - Aurora
  - CockroachDB
  - Informix
  - Teradata

Show me the code!!

Setting up the build - Maven

```
<dependency>
    <groupId>org.jooq</groupId>
    <artifactId>jooq</artifactId>
    <version>3.16.5</version>
</dependency>
<dependency>
    <groupId>org.jooq</groupId>
    <artifactId>jooq-meta</artifactId>
    <version>3.16.5</version>
</dependency>
```

Setting up the build - Gradle

You're on your own

Just kidding!

```
dependencies {
    compile 'org.jooq:jooq:3.16.5'
    compile 'org.jooq:jooq-meta:3.16.5'
}
```

First step - DSLContext

- Primary API is the `DSLContext`
- Can use either `Connection` or a `DataSource`
  - DSLContext context = DSL.using(conn, SQLDialect.MYSQL);
  - DSLContext context = DSL.using(dataSource, SQLDialect.MYSQL);

First step - DSLContext (cont)

- For finer-grained control, use a `Configuration` object:

```
Configuration configuration = new DefaultConfiguration()
    .set(conn)
    .set(SQLDialect.POSTGRES)
    .set(new Settings()
        .withRenderQuotedNames(RenderQuotedNames.NEVER)
        .withRenderNameCase(RenderNameCase.LOWER_IF_UNQUOTED)
    );
DSLContext context = DSL.using(configuration);
```

Writing queries - non-codegen

```
dsl.select(
    DSL.field("actor_id"),
    DSL.field("last_name"),
    DSL.field("first_name")
)
.from(DSL.table("actor"))
```

Fetching data - non-codegen

```
dsl.select(
    DSL.field("actor_id"),
    DSL.field("last_name"),
    DSL.field("first_name")
)
.from(DSL.table("actor"))
.fetch()
    .map(r -> Author.fromRecord(r))
    .collect(Collectors.toList())
```

Fetching data - non-codegen (cont)

Fetching:

```
public class Actor {
    public static Actor fromRecord(Record r) {
        return new Actor()
                .setId(r.getValue("actor_id", Long.class))
                .setFirstName(r.getValue("first_name", String.class))
                .setLastName(r.getValue("last_name", String.class));
    }
}
```

Code Generation - jOOQ Done right

You can use jOOQ just for SQL authoring. But you probably shouldn't. :)

Code generation produces lots of helpful artifacts

- Tables
- POJOs
- User-defined types
- Records
- DAOs
- and many more
- Sequences
- Stored procedures

Generating code

Options:

- XML (commandline and Maven)
- Gradle

Generating Code - XML

```
<plugin>
  <groupId>org.jooq</groupId>
  <artifactId>jooq-codegen-maven</artifactId>
  <version>${version.jooq}</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Generating Code - XML (cont)

```
<configuration>
    <jdbc>
        <url>${jdbc.url}</url>
        <user>${jdbc.user}</user>
        <password>${jdbc.password}</password>
        <schema>public</schema>
    </jdbc>
    <generator>
        <database>
            <name>org.jooq.meta.postgres.PostgresDatabase</name>
            <includes>.*</includes>
            <inputSchema>public</inputSchema>
            <outputSchema>public</outputSchema>
        </database>
        <target>
            <packageName>com.steeplesoft.jooq_demo.generated</packageName>
            <directory>${jooq.outputdir}</directory>
        </target>
    </generator>
</configuration>
```

Generating Code - XML (cont)

```
$ mvn generate-sources
...
[INFO] Generating catalog       : DefaultCatalog.java
[INFO] ==========================================================
[INFO] Generating schemata      : Total: 1
[INFO] No schema version is applied for schema public. Regenerating.
[INFO] Generating schema        : Public.java
[INFO] ----------------------------------------------------------
[INFO] Tables fetched           : 31 (31 included, 0 excluded)
[INFO] Enums fetched            : 1 (1 included, 0 excluded)
[INFO] UDTs fetched             : 0 (0 included, 0 excluded)
[INFO] Sequences fetched        : 0 (0 included, 0 excluded)
[INFO] Generating tables
[INFO] Embeddables fetched      : 0 (0 included, 0 excluded)
[INFO] Generating table         : Actor.java [input=actor, output=actor, pk=actor_pkey]
```

Generating Code - XML + CLI

Maven `configuration` saved in external file (e.g., jooq.xml)

File pre-amble:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <configuration xmlns="http://www.jooq.org/xsd/jooq-codegen-3.16.5.xsd">
```

Command-line:

```
$ java -classpath jooq-3.16.5.jar:\
    jooq-meta-3.16.5.jar:jooq-codegen-3.16.5.jar:\
    reactive-streams-1.0.3.jar:r2dbc-spi-0.9.0.RELEASE.jar:\
    jakarta.xml.bind-api-3.0.0.jar:mysql-connector-java.jar:. \
    org.jooq.codegen.GenerationTool jooq.xml
```

Generating Code - Gradle

- Recommended to use the Gradle plugin by Etienne Studer (from Gradle Inc.).
- Examples at https://github.com/etiennestuder/gradle-jooq-plugin/tree/master/example
- Programmatic:

```
GenerationTool.generate(new Configuration()
    .withJdbc(new Jdbc()
        .withDriver('org.h2.Driver')
        .withUrl('jdbc:h2:~/test-gradle')
        .withUser('sa')
        .withPassword(''))
    .withGenerator(new Generator()) // ...
```

- Full example

Get on with it!!

Writing queries - Codegen

Old:

```
dsl.select(
        DSL.field("actor_id"),
        DSL.field("last_name"),
        DSL.field("first_name")
)
.from(DSL.table("actor"))
```

Code gen:

```
dsl.fetch(ACTOR)
```

Filtering Data

- Simple

```
List films = dsl.select()
    .from(FILM)
    .where(FILM.TITLE.like("%THE%"))
    .fetch().map(r -> FilmModel.fromRecord(r));
```

- SQL:

```
select
  film.film_id,
  film.title,
  ...
  film.fulltext
from film
where film.title like '%THE%'
```

Filtering Data (cont)

- More complex

```
List customers = dsl.select()
    .from(CUSTOMER)
    .where(CUSTOMER.FIRST_NAME.eq("MARION"))
    .and(CUSTOMER.LAST_NAME.eq("SNYDER"))
    .fetch().map(r -> CustomerModel.fromRecord(r));
```

- SQL:

```
select
  customer.customer_id,
  ...
  customer.active
from customer
where (
  customer.first_name = 'MARION'
  and customer.last_name = 'SNYDER'
)
```

Filter Data (cont)

- One more:

```
dsl.select().from(CUSTOMER)
    .where(CUSTOMER.FIRST_NAME.eq("MARION"))
        .and(CUSTOMER.LAST_NAME.eq("SNYDER"))
    .or(CUSTOMER.FIRST_NAME.eq("TERRY"))
        .and(CUSTOMER.LAST_NAME.eq("GRISSOM"))
```

- SQL:

```
select ... from customer
where (
  (
    (
      customer.first_name = 'MARION' and customer.last_name = 'SNYDER'
    )
    or customer.first_name = 'TERRY'
  )
  and customer.last_name = 'GRISSOM')
```

Filter Data (cont)

- I lied. Let's fix it. Old:

```
dsl.select().from(CUSTOMER)
    .where(CUSTOMER.FIRST_NAME.eq("MARION"))
        .and(CUSTOMER.LAST_NAME.eq("SNYDER"))
    .or(CUSTOMER.FIRST_NAME.eq("TERRY"))
        .and(CUSTOMER.LAST_NAME.eq("GRISSOM"))
```

- SQL:

```
select ... from customer
where (
  (
    (
      customer.first_name = 'MARION'
      and customer.last_name = 'SNYDER'
    )
    or customer.first_name = 'TERRY'
  )
  and customer.last_name = 'GRISSOM')
```

- New:

```
dsl.select().from(CUSTOMER)
    .where(CUSTOMER.FIRST_NAME.eq("MARION")
        .and(CUSTOMER.LAST_NAME.eq("SNYDER")))
    .or(CUSTOMER.FIRST_NAME.eq("TERRY")
        .and(CUSTOMER.LAST_NAME.eq("GRISSOM")))
```

- SQL:

```
where (
  (
    customer.first_name = 'MARION'
    and customer.last_name = 'SNYDER'
  ) or (
    customer.first_name = 'TERRY'
    and customer.last_name = 'GRISSOM'
  )
)
```

Joins

- Join types supported:
  - Cross - Cartesian product
  - Inner - Values in both tables
  - Outer - Value from one table, null from the other (LEFT, RIGHT, and FULL)
  - Semi - "existence of rows from one table in another table (using EXISTS or IN)"

Joins - Cross

No `WHERE` class allowed.

```
dsl.select()
    .from(STORE)
    .crossJoin(STAFF)
    .fetch();
```

Joins - Inner

```
dsl.select()
    .from(STORE)
        .join(STAFF)
            .on(STORE.STORE_ID.eq(STAFF.STORE_ID))
    .fetch();
```

Joins - Outer

- Left outer join:

```
dsl.select()
    .from(AUTHORS)
    .leftOuterJoin(BOOKS).on(AUTHORS.ID.eq(BOOKS.AUTHOR_ID))
```

- Right outer join:

```
dsl.select()
    .from(AUTHORS)
    .rightOuterJoin(BOOKS).on(AUTHORS.ID.eq(BOOKS.AUTHOR_ID))
```

- Full outer join:

```
dsl.select()
    .from(AUTHORS)
    .fullOuterJoin(BOOKS).on(AUTHORS.ID.eq(BOOKS.AUTHOR_ID))
```

Joins - Implicit

"Normal" way

```
dsl.select(
            STAFF.STAFF_ID,
            STAFF.LAST_NAME,
            STAFF.FIRST_NAME,
            STAFF.STORE_ID,
            ADDRESS.ADDRESS_,
            CITY.CITY_
    ).from(STAFF)
    .join(STORE).on(STAFF.STORE_ID.eq(STORE.STORE_ID))
    .join(ADDRESS).on(STORE.ADDRESS_ID.eq(ADDRESS.ADDRESS_ID))
    .join(CITY).on(ADDRESS.CITY_ID.eq(CITY.CITY_ID))
    .fetch();
```

Joins - Implicit (cont)

Implicit join

```
dsl.select(
            STAFF.STAFF_ID,
            STAFF.LAST_NAME,
            STAFF.FIRST_NAME,
            STAFF.store().STORE_ID,
            STAFF.store().address().ADDRESS_,
            STAFF.store().address().city().CITY_
    ).from(STAFF)
    .fetch();
```

Inserting Data

Multiple ways:

- INSERT .. VALUES
- INSERT .. RETURNING
- INSERT .. SET
- INSERT .. DEFAULT VALUES
- INSERT .. SELECT
- INSERT .. ON DUPLICATE KEY

Inserting Data: INSERT .. VALUES

- dsl.insertInto(<Table>, [field1, field2, …, fieldN])
- Example:

```
dsl.insertInto(CUSTOMER,
    CUSTOMER.CUSTOMER_ID, CUSTOMER.STORE_ID, CUSTOMER.FIRST_NAME, CUSTOMER.LAST_NAME,
        CUSTOMER.ADDRESS_ID, CUSTOMER.ACTIVE)
    .values(1000, 1, "Dummy", "User", 1, 1)
    .execute();
```

Inserting Data: INSERT .. RETURNING

```
Record1<Integer> record = dsl.insertInto(CUSTOMER,
            CUSTOMER.STORE_ID, CUSTOMER.FIRST_NAME, CUSTOMER.LAST_NAME,
            CUSTOMER.ADDRESS_ID, CUSTOMER.ACTIVE)
    .values(1, "Dummy", "User", 1, 1)
    .returningResult(CUSTOMER.CUSTOMER_ID)
    .fetchOne();
Integer key = record.get(CUSTOMER.CUSTOMER_ID);
```

Inserting Data: INSERT .. SET

```
Record1<Integer> record = dsl.insertInto(CUSTOMER)
        .set(CUSTOMER.STORE_ID, 1)
        .set(CUSTOMER.FIRST_NAME, "Dummy")
        .set(CUSTOMER.LAST_NAME, "User")
        .set(CUSTOMER.ADDRESS_ID, 1)
        .set(CUSTOMER.ACTIVE, 1)
        .returningResult(CUSTOMER.CUSTOMER_ID)
        .fetchOne();
Integer key = record.get(CUSTOMER.CUSTOMER_ID);
```

Updating Data

- Basic:

```
int count = dsl.update(CUSTOMER)
    .set(CUSTOMER.ACTIVEBOOL, false)
    .set(CUSTOMER.ACTIVE, 0)
    .where(CUSTOMER.CUSTOMER_ID.eq(1))
    .execute();
```

- Row value expression:

```
int count = dsl.update(CUSTOMER)
        .set(
           row(CUSTOMER.ACTIVEBOOL, CUSTOMER.ACTIVE),
           row(true, 1)
        )
    .where(CUSTOMER.CUSTOMER_ID.eq(1))
    .execute();
```

Deleting Data

```
int count = dsl.delete(CUSTOMER)
    .where(CUSTOMER.CUSTOMER_ID.eq(1000))
    .execute();
```

Aggregate Functions

- AVG
- SUM
- COUNT
- MAX
- MEDIAN
- MIN
- BOOL_AND
- BOOL_OR
- COLLECT
- CUME_DIST
- DENSE_RANK
- EVERY
- GROUP_CONCAT
- JSON_ARRAYAGG
- JSON_OBJECTAGG
- LISTAGG
- MODE
- MULTISET_AGG
- PERCENT_RANK
- PERCENTILE_CONT
- PERCENTILE_DISC
- PRODUCT
- RANK
- XMLAGG

Data Definition Language(DDL)

You can alter the database schema itself as well:

- ALTER DATABASE
- ALTER SCHEMA
- ALTER TYPE
- DROP DOMAIN
- DROP PROCEDURE
- DROP TABLE
- DROP VIEW
- CREATE FUNCTION
- ALTER DOMAIN
- ALTER SEQUENCE
- ALTER VIEW
- DROP FUNCTION
- DROP SCHEMA
- DROP TRIGGER
- CREATE DATABASE
- CREATE OR REPLACE FUNCTION
- ALTER INDEX
- ALTER TABLE
- DROP DATABASE
- DROP INDEX
- DROP SEQUENCE
- DROP TYPE
- CREATE DOMAIN

A CRUDdy Tour

- Create

```
ActorRecord record = dsl.newRecord(ACTOR);
record.setFirstName("Test");
record.setLastName("Actor");
record.store();
int storeId = record.getActorId();
```

- Update

```
ActorRecord actor = dsl.fetchOne(ACTOR, ACTOR.ACTOR_ID.eq(10));
actor.setLastName("Updated");
actor.update();
```

- Delete

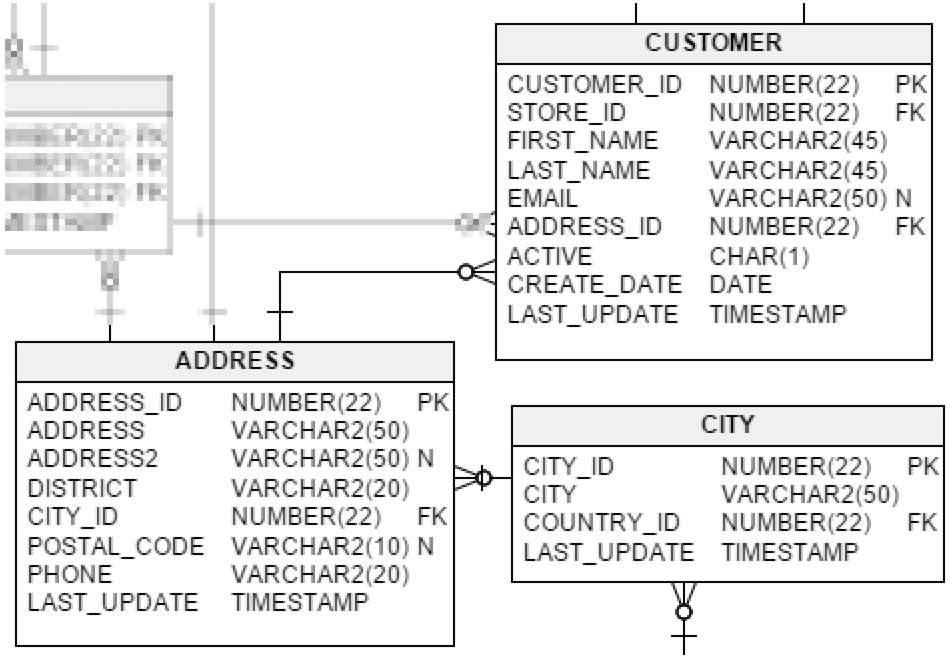
```
ActorRecord record = dsl.fetchOne(ACTOR, ACTOR.ACTOR_ID.eq(201));
record.delete()
```

Advanced Wizardry

- Nested Objects
- Multisets

Advanced Wizardry - Nested Objects

Scenario: 1:1 relationships, such as Customer -> Address



CUSTOMER		
CUSTOMER_ID	NUMBER(22)	PK
STORE_ID	NUMBER(22)	FK
FIRST_NAME	VARCHAR2(45)	
LAST_NAME	VARCHAR2(45)	
EMAIL	VARCHAR2(50) N	
ADDRESS_ID	NUMBER(22)	FK
ACTIVE	CHAR(1)	
CREATE_DATE	DATE	
LAST_UPDATE	TIMESTAMP	

ADDRESS		
ADDRESS_ID	NUMBER(22)	PK
ADDRESS	VARCHAR2(50)	
ADDRESS2	VARCHAR2(50) N	
DISTRICT	VARCHAR2(20)	
CITY_ID	NUMBER(22)	FK
POSTAL_CODE	VARCHAR2(10) N	
PHONE	VARCHAR2(20)	
LAST_UPDATE	TIMESTAMP	

CITY		
CITY_ID	NUMBER(22)	PK
CITY	VARCHAR2(50)	
COUNTRY_ID	NUMBER(22)	FK
LAST_UPDATE	TIMESTAMP	

Advanced Wizardry - Nested Objects (cont)

The magic sauce: `DSL.row()`

```
row(
    ADDRESS.ADDRESS_,
    ADDRESS.ADDRESS2,
    ADDRESS.CITY_ID,
    ADDRESS.POSTAL_CODE,
    ADDRESS.PHONE
).mapping(
    (address, address2, cityId, postalCode, phone) -> new AddressModel()
        .setAddress(address)
        .setAddress2(address2)
        .setPostalCode(postalCode)
        .setPhone(phone)
).as("address")
```

Advanced Wizardry - Nested Objects (cont)

You can nest the nesting:

```
row(
        ADDRESS.ADDRESS_,
        ADDRESS.ADDRESS2,
        row(
                CITY.CITY_ID,
                CITY.CITY_,
                CITY.LAST_UPDATE
        ).mapping((id, city, lastUpdate) -> new CityModel()
                .setCityId(id)
                .setCity(city)
                .setLastUpdate(lastUpdate)
        ).as("city"),
        ADDRESS.POSTAL_CODE,
        ADDRESS.PHONE
)
```

Advanced Wizardry - Nested Objects (cont)

You still need to make the joins

```
dsl.select(
    CUSTOMER.FIRST_NAME,
    //...
)
.from(CUSTOMER)
    .join(ADDRESS)
        .on(CUSTOMER.ADDRESS_ID.eq(ADDRESS.ADDRESS_ID))
    .join(CITY)
        .on(ADDRESS.CITY_ID.eq(CITY.CITY_ID))
```

Advanced Wizardry - Nested Objects (cont)

However, implicit joins work

```
dsl.select(
        CUSTOMER.FIRST_NAME,
        ...
        row(
                CUSTOMER.address().ADDRESS_,
                CUSTOMER.address().ADDRESS2,
                row(
                        CUSTOMER.address().city().CITY_ID,
                        CUSTOMER.address().city().CITY_,
                        CUSTOMER.address().city().LAST_UPDATE
                ).mapping(...),
                ...
        ).mapping(..)
)
.from(CUSTOMER)
.where(CUSTOMER.CUSTOMER_ID.eq(1))
```

Advanced Wizardry - Nested Objects (cont)

Function references and methods can make the code more concise and reusable:

```
dsl.select(
        CUSTOMER.FIRST_NAME,
        CUSTOMER.LAST_NAME,
        CUSTOMER.EMAIL,
        addressRow()
)
.from(CUSTOMER)
.where(CUSTOMER.CUSTOMER_ID.eq(1))
.fetchOne(mapping(this::mapCustomer));
```

Advanced Wizardry - Nested Objects (cont)

Extracted address `row`:

```
private SelectField<AddressModel> addressRow() {
    return row(
            CUSTOMER.address().ADDRESS_,
            CUSTOMER.address().ADDRESS2,
            cityRow(),
            CUSTOMER.address().POSTAL_CODE,
            CUSTOMER.address().PHONE
    ).mapping(this::mapAddress).as("address");
}
```

Advanced Wizardry - Nested Objects (cont)

Extracted record mapping:

```
private AddressModel mapAddress(String address,
        String address2,
        CityModel city,
        String postalCode,
        String phone) {
    return new AddressModel()
            .setAddress(address)
            .setAddress2(address2)
            .setCity(city)
            .setPostalCode(postalCode)
            .setPhone(phone);
}
```

Advanced Wizardry - Multisets

- Great for 1:M relationships
- Get the "base" record and 0 or more related records as a `List`
- Query is built to return related objects as JSON and reconstituted in the application

Advanced Wizardry - Multisets (cont)

```
dsl.select(
    STORE.STORE_ID,
    addressRow(STORE.address()),
    multiset(
        dsl.select(
            STAFF.STAFF_ID,
            STAFF.FIRST_NAME,
            STAFF.LAST_NAME,
            addressRow(STAFF.address()),
            STAFF.EMAIL
        )
        .from(STAFF)
        .where(STAFF.STORE_ID.eq(STORE.STORE_ID))
    ).as("staff").convertFrom(r -> r.map(mapping(this::mapStaff)))
)
.from(STORE)
.fetchInto(StoreModel.class)
```

Advanced Wizardry - Multisets (cont)

```
private StaffModel mapStaff(Integer id,
        String firstName,
        String lastName,
        AddressModel address,
        String email) {
    return new StaffModel()
            .setStaffId(id)
            .setFirstName(firstName)
            .setLastName(lastName)
            .setAddress(address)
            .setEmail(email);
}
```

Advanced Wizardry - Multisets (cont)

```
private SelectField<AddressModel> addressRow(Address address) {
    return row(
            address.ADDRESS_,
            address.ADDRESS2,
            cityRow(address),
            address.POSTAL_CODE,
            address.PHONE
    ).mapping(this::mapAddress).as("address");
}
```

So very much more

- Conditional conditions
- Lazy fetching (with streams)
- Reactive fetching
- Batch operations
- Stored Procedures and functions
- Importing from / exporting to XML, CSV, JSON, …

More resources

- jOOQ Docs: https://www.jooq.org/doc/latest/manual
- Stackoverflow: https://stackoverflow.com/questions/tagged/jooq

Thanks for coming!

- Twitter: @jasondlee
- Blog: https://jasondl.ee
- LinkedIn: https://linkedin.com/in/jasondlee
- Presentation source: https://github.com/jasondlee/jooq-presentation