

Querydsl - ##### ##

**Timo Westkämper
Samppa Saarela
Vesa Marttila
Lassi Immonen
Ruben Dijkstra**

Querydsl - #####

Timo Westkämper, Samppa Saarela, Vesa Marttila, Lassi Immonen, ### Ruben Dijkstra

3.5.1

© 2007-2014 Original authors

Legal Notice

© Mysema Ltd, 2007-2013. # ##### [Apache License, Version 2.0](#) # ## ## ##, ##, ##, ### #####.

##

##	vi
1. Introduction	1
1.1. Background	1
1.2. ##	1
2. #####	2
2.1. JPA ##	2
### ##	2
Ant ##	3
Roo## Querydsl JPA #####	3
hbm.xml ##### ## ## #####	4
## ## #####	4
##	5
##	6
## ##	7
##	7
###	7
DeleteClause	8
UpdateClause	8
#####	8
### JPA Query ###	9
JPA ##### ##### SQL #####	9
2.2. JDO ##	11
### ##	11
Ant ##	12
## ## #####	12
##	13
## ##	14
##	15
###	15
DeleteClause	15
#####	15
##### SQL #####	16
2.3. SQL ##	17
### ##	18
##### ## ## ##	18
ANT# ## ## ##	21
## ## #####	21
##	22
##	23
## ##	23

##	24
##	24
###	24
####	25
### ##	25
## ## ##	25
### ##	26
## SQL ###	26
DML ## ####	26
##	26
##	28
##	28
DMLClause# ## ##	28
# ### ##	29
SQL ### ### ####	30
### ##	30
Query# Clause ###	31
2.4. ## ##	31
Maven integration	31
## ## ##	32
##	32
## ##	32
##	33
## ## ##	33
###	33
##(fuzzy) ##	34
## ### ### ####	34
2.5. Hibernate Search ##	34
Querydsl ## ## ##	34
##	34
## ##	35
2.6. Mongoddb ##	35
### ##	35
##	36
## ##	36
##	36
## ## ##	37
###	37
##(Geospatial) ##	37
## ### ####	37
2.7. ### ##	37
### ## ## ## ####	38

### ## ### ## #####	39
### ##	39
Ant ##	40
Hamcrest matchers	40
2.8. Scala## #####	41
Scala# ## DSL ##	41
### #####	42
SQL# ### ##	42
### ##	43
## ##	43
## ##### ## ##	44
3. ## ###	46
3.1. ## ##	46
## ##(complex predicates)	46
## ###	46
## ##	47
Case ###	47
Casting ###	48
### ##	48
3.2. ## ##	49
## ## ##	49
# ##(population)	49
### ##	50
## ##(aggregation)	51
3.3. ## ##	51
## ###	51
#####	52
### ## ##	54
## ##(Delegate methods)	54
##### ## ##	56
##### ## ## ##	57
### ###	57
Scala ##	59
3.4. ## ###	60
4. #####	62
4.1. ##### ## ##	62
4.2. ##### ##### Querydsl Q### ###	62
4.3. JDK5 ##	63

##

Querydsl ## #### SQL# ## #### # ## # ## #####. #### XML ### #### ##, Querydsl
#(Fluent) API# #### ## # ##.

Fluent API# ## ## ## ##.

1. IDE# ## ## ## ##
2. ##### ## ## ## ##
3. #### ## #### ## ## # ##
4. #### ## #### # # # #

1#. Introduction

1.1. Background

Querydsl# #### HQL #### ## #### #####. HQL #### ##### String #### ##, ## ####
#####. String# ##### ## #### ##### ## #### ## # # ##, ## String
HQL #### # ##### # ## ##.

##. ##### #### ##### ## ####, ## ## ##### ## ##### ## #
##.

Querydsl# ## ## ## ## Hibernate# HQL####, ## JPA, JDO, JDBC, Lucene, Hibernate Search,
MongoDB, ## #### RDFBean# #####.

1.2.

Querydsl# ## #### ## ####(Type safety)##. #### ##### ##### ## #### ##### ## ##### ##. ##, #### #
##/#### ## #####.

####(consistency)##. ## #### ##### ## #### ##### ## ####, Query ##### ## ## ##### ##.

#####. ## ## ## #### ##### ##### ## #####.

Javadoc## com.mysema.query.Query, com.mysema.query.Projectable ###
com.mysema.query.types.Expression# #### ## Querydsl #### ## #### ##### ## # ##.

```
#### ## ## ## ## Querydsl# #### ## ### ## ## ## ## ##.
```

Querydsl# ## ### ## ## # ## ##### ## ## ### ##### ##. JDO# JPA# Querydsl# ##### ## #####. # ##
JPA# ## Querydsl# ##### ## ##.

##

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-apt</artifactId>
  <version>${querydsl.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-jpa</artifactId>
  <version>${querydsl.version}</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.6.1</version>
</dependency>
```

```
<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.maven</groupId>
        <artifactId>apt-maven-plugin</artifactId>
        <version>1.0.9</version>
        <executions>
          <execution>
            <goals>
              <goal>process</goal>
            </goals>
```



```

    <configuration>
      <outputDirectory>target/generated-sources/java</outputDirectory>
      <processor>com.mysema.query.apt.jpa.JPAAnnotationProcessor</processor>
    </configuration>
  </execution>
</executions>
</plugin>
...
</plugins>
</build>
</project>

```

JPAAnnotationProcessor# javax.persistence.Entity ##### ## ### ### ## ## ## ## ##.

Hibernate ##### ##, APT #####
com.mysema.query.apt.hibernate.HibernateAnnotationProcessor# ##### #.

mvn clean install # #####, target/generated-sources/java ##### Query ### #####.

#, mvn eclipse:eclipse # ##### target/generated-sources/java ##### ## ### #####.

Query ### ##### JPA ## ##### ## ### ## ##### ## # ##.

Ant

full-deps# ### jar ##### ##, ## ##### ##### Querydsl ### #####.

```

<!-- APT based code generation -->
<javac srcdir="${src}" classpathref="cp">
  <compilerarg value="-proc:only"/>
  <compilerarg value="-processor"/>
  <compilerarg value="com.mysema.query.apt.jpa.JPAAnnotationProcessor"/>
  <compilerarg value="-s"/>
  <compilerarg value="${generated}"/>
</javac>

<!-- compilation -->
<javac classpathref="cp" destdir="${build}">
  <src path="${src}"/>
  <src path="${generated}"/>
</javac>

```

src# ## ## ### #####, generated# ### ### ## ### #####, build# ### ## ### #####.

Roo## Querydsl JPA

Roo## Querydsl JPA# #####, com.mysema.query.apt.jpa.JPAAnnotationProcessor
com.mysema.query.apt.roo.RooAnnotationProcessor# ### # ##.
RooAnnotationProcessor# @Entity# ### ### ## @RooJpaEntity# @RooJpaActiveRecord ###
#####.

APT ### ## ## ### AspectJ IDT## # ##### ##.

hbm.xml ##### ## ##

XML ## ## ## ## ##, Querydsl ### ##### ## XML ##### ## # ##.

com.mysema.query.jpa.codegen.HibernateDomainExporter# # ## ##.

```
HibernateDomainExporter exporter = new HibernateDomainExporter(
    "Q", // name prefix
    new File("target/gen3"), // target folder
    configuration); // instance of org.hibernate.cfg.Configuration

exporter.export();
```

HibernateDomainExporter# ##### ##### ##### ##### ## ## ## ##, HibernateDomainExporter# ##### ##
##.

JPA ##### ##, @QueryInit## @QueryType# ## Querydsl ##### ##.

#####

Querydsl# ##### ## ## ##, ## Query ##### ## ## ## ## ##.

##.

```
@Entity
public class Customer {
    private String firstName;
    private String lastName;

    public String getFirstName(){
        return firstName;
    }

    public String getLastName(){
        return lastName;
    }

    public void setFirstName(String fn){
        firstName = fn;
    }

    public void setLastName(String ln){
        lastName = ln;
    }
}
```

Querydsl# Customer# ### ##### QCustomer## ## ## ## ## ##. Querydsl ##### Customer ## ## ##
QCustomer# #####.

QCustomer# ## ##### ## ## ##, ## ## ## ## ## # ##.

```
QCustomer customer = QCustomer.customer;
```

Customer ### ## ## ## ##.

```
QCustomer customer = new QCustomer("myCustomer");
```

##

Querdsl JPA ### JPA# Hibernate API# ## #####.

JPA API# ##### ## ## JPAQuery ##### ## ##.

```
// where entityManager is a JPA EntityManager
JPAQuery query = new JPAQuery(entityManager);
```

Hibernate# #####, HibernateQuery# ##### ##.

```
// where session is a Hibernate session
HibernateQuery query = new HibernateQuery(session);
```

JPAQuery# HibernateQuery# # # JPQLQuery ##### ## ##.

firstName ##### Bob# Customer# ##### ## ## ## ## ## ##.

```
QCustomer customer = QCustomer.customer;
JPAQuery query = new JPAQuery(entityManager);
Customer bob = query.from(customer)
    .where(customer.firstName.eq("Bob"))
    .uniqueResult(customer);
```

from ##### ## ##(##)# #####, where ### ## ## ##, uniqueResult# ##### ## ##, 1# ## ## ## ## ##.

##.

```
QCustomer customer = QCustomer.customer;
QCompany company = QCompany.company;
query.from(customer, company);
```

##.

```
query.from(customer)
    .where(customer.firstName.eq("Bob"), customer.lastName.eq("Wilson"));
```

##, ### ## ## ##.

```
query.from(customer)
    .where(customer.firstName.eq("Bob").and(customer.lastName.eq("Wilson")));
```

JPQL ### ##### ## ##.

```
from Customer as customer
    where customer.firstName = "Bob" and customer.lastName = "Wilson"
```

or# ##### ## ## ##.

```
query.from(customer)
    .where(customer.firstName.eq("Bob").or(customer.lastName.eq("Wilson")));
```

##

Querydsl# JPQL# ## ##, ##, ### ##, #####. ## ## ## ## ## ##.

```
QCat cat = QCat.cat;
QCat mate = new QCat("mate");
QCat kitten = new QCat("kitten");
query.from(cat)
    .innerJoin(cat.mate, mate)
    .leftJoin(cat.kittens, kitten)
    .list(cat);
```

JPQL# ##### ## ##.

```
from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
```

##.

```
query.from(cat)
    .leftJoin(cat.kittens, kitten)
    .on(kitten.bodyWeight.gt(10.0))
    .list(cat);
```

JPQL ### ## ##.

```
from Cat as cat
    left join cat.kittens as kitten
    on kitten.bodyWeight > 10.0
```

##

JPQLQuery ##### cascading ##### ##.

from: ## ### #####.

innerJoin, join, leftJoin, fullJoin, on: ## ### #####. ## ##### # ## ### ## ####, # ## ### ##(##)##.

where: ## ### #####. ##### and/or ##### ##### ## #####.

groupBy: ##### ## ## ##### ## #####.

having: Predicate ##### "group by" ##### ## #####.

orderBy: ## ##### ##### ## ## #####. ## ##### ##### asc()# desc()# ####, OrderSpecifier# ##### ## ## #
#####.

limit, offset, restrict: ### ##### #####. limit# ## ## ##, offset# ### ## #, restrict# limit# offset# ## #####.

##

##.

```
QCustomer customer = QCustomer.customer;
query.from(customer)
    .orderBy(customer.lastName.asc(), customer.firstName.desc())
    .list(customer);
```

JPQL# #####.

```
from Customer as customer
    order by customer.lastName asc, customer.firstName desc
```

###

#####.

```
query.from(customer)
    .groupBy(customer.lastName)
    .list(customer.lastName);
```

JPQL# ### ##.

```
select customer.lastName
    from Customer as customer
    group by customer.lastName
```

DeleteClause

Querydsl JPA## DeleteClause# ### delete-where-execute ### ###. ### # ## #.

```
QCustomer customer = QCustomer.customer;
// delete all customers
new JPADeleteClause(entityManager, customer).execute();
// delete all customers with a level less than 3
new JPADeleteClause(entityManager, customer).where(customer.level.lt(3)).execute();
```

JPADeleteClause ##### # ## ##### ### ### #####. where# ### ## ### # ###, execute# ##### ### ##### #
#####.

Hibernate ###, HibernateDeleteClause# ##### ##.

JPA# DML ## JPA ### ### ## ### ### ##, 2# ## ##### ##### ##.

UpdateClause

Querydsl JPA# UpdateClause# ### update-set/where-execute ### ###. ### # ## #.

```
QCustomer customer = QCustomer.customer;
// rename customers named Bob to Bobby
new JPAUpdateClause(session, customer).where(customer.name.eq("Bob"))
    .set(customer.name, "Bobby")
    .execute();
```

JPAUpdateClause ##### # ## ##### ### ### #####. set# SQL# update ##### ##### ### #####, execute# ###
#####.

Hibernate ###, HibernateUpdateClause# #####.

JPA## DML ## JPA ### ### ## ### ### ##, 2# ## ##### ##### ##.

#####

JPASubQuery# ##### ##. ##### ### ## from ##### ## ##### #####, unique# list# #####. unique
list# ### ### ## #####. ##### ##### ### ### Querydsl #####.

```
QDepartment department = QDepartment.department;
QDepartment d = new QDepartment("d");
query.from(department)
    .where(department.employees.size().eq(
        new JPASubQuery().from(d).unique(d.employees.size().max())
    )).list(department);
```

##

```

QEmployee employee = QEmployee.employee;
QEmployee e = new QEmployee("e");
query.from(employee)
    .where(employee.weeklyhours.gt(
        new JPASubQuery().from(employee.department.employees, e)
        .where(e.manager.eq(employee.manager))
        .unique(e.weeklyhours.avg())
    )).list(employee);

```

Hibernate# ### ##, HibernateSubQuery# ##### ##.

JPA Query

JPA Query# ### ##, ## ### #####.

```

JPAQuery query = new JPAQuery(entityManager);
Query jpaQuery = query.from(employee).createQuery(employee);
// ...
List results = jpaQuery.getResultList();

```

JPA ##### SQL

JPASQLQuery ##### JPA# ##### SQL# Querydsl## ### # ##.

SQL ##### ## Querydsl ## ### ##### ##. ### ## ## Maven ## ## ##### ##.

```

<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.querydsl</groupId>
        <artifactId>querydsl-maven-plugin</artifactId>
        <version>${querydsl.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>export</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <jdbcDriver>org.apache.derby.jdbc.EmbeddedDriver</jdbcDriver>
          <jdbcUrl>jdbc:derby:target/demoDB;create=true</jdbcUrl>
          <packageName>com.mycompany.mydomain</packageName>
          <targetFolder>${project.basedir}/target/generated-sources/java</targetFolder>
        </configuration>
        <dependencies>
          <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derby</artifactId>
            <version>${derby.version}</version>

```

```

        </dependency>
    </dependencies>
</plugin>
...
</plugins>
</build>
</project>

```

#####, #### # #### # ##.

##:

```
// serialization templates
SQLTemplates templates = new DerbyTemplates();
// query types (S* for SQL, Q* for domain types)
SAnimal cat = new SAnimal("cat");
SAnimal mate = new SAnimal("mate");
QCat catEntity = QCat.cat;

JPASQLQuery query = new JPASQLQuery(entityManager, templates);
List<String> names = query.from(cat).list(cat.name);
```

```
# ##### (#, QCat)# ##### (#, SAnimal)# ## ##### ## ## ##, ## ##### ##### ## ##. SAnimal.animal# "animal"#
# ##### ##### ## ## ## (new SAnimal("cat"))# ## #####.
```

##.

```
QCat catEntity = QCat.cat;  
SAnimal cat = new SAnimal(catEntity.getMetadata().getName());
```

##:

```
query = new JPASQLQuery(entityManager, templates);
List<Object[]> rows = query.from(cat).list(cat.id, cat.name);
```

##:

```
List<Object[]> rows = query.from(cat).list(cat.all());
```

SQL# ### ##, ### ##### ###:

```
query = new JPASQLQuery(entityManager, templates);  
List<Cat> cats = query.from(cat).orderBy(cat.name.asc()).list(catEntity);
```

##:


```
query = new JPASQLQuery(entityManager, templates);
cats = query.from(cat)
    .innerJoin(mate).on(cat.mateId.eq(mate.id))
    .where(cat.dtype.eq("Cat"), mate.dtype.eq("Cat"))
    .list(catEntity);
```

DTO# ###:

```
query = new JPASQLQuery(entityManager, templates);
List<CatDTO> catDTOS = query.from(cat)
    .orderBy(cat.name.asc())
    .list(ConstructorExpression.create(CatDTO.class, cat.id, cat.name));
```

JPA API ## ##### API# #####, HibernateSQLQuery# #####.

2.2. JDO

Querydsl# ## ### ## ## # ## ##### ## ## ### ##### ##. JDO# JPA# Querydsl# ##### ## #####. # ##
JDO# ## Querydsl# ##### ## #####.

##

##.

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-apt</artifactId>
  <version>${querydsl.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-jdo</artifactId>
  <version>${querydsl.version}</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.6.1</version>
</dependency>
```

Querydsl## ## ### ##### ## ##### ## APT ##### ##.

```
<project>
  <build>
    <plugins>
      ...
    <plugin>
```

```

    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>1.0.9</version>
    <executions>
      <execution>
        <goals>
          <goal>process</goal>
        </goals>
        <configuration>
          <outputDirectory>target/generated-sources/java</outputDirectory>
          <processor>com.mysema.query.apt.jdo.JDOAnnotationProcessor</processor>
        </configuration>
      </execution>
    </executions>
  </plugin>
  ...
</plugins>
</build>
</project>

```

JDOAnnotationProcessor# javax.jdo.annotations.PersistenceCapable ##### ## ## #
##.

`mvn clean install`# ####, target/generated-sources/java ##### Query ### ##.

##, `mvn eclipse:eclipse`# ##### target/generated-sources/java ##### ## ## ##.

Query ### ##### JDO ## ##### ## ## ## ## ## ## ##.

Ant

full-deps# ### jar ##### ## ## ## ## Querydsl ### ##.

```

<!-- APT based code generation -->
<javac srcdir="${src}" classpathref="cp">
  <compilerarg value="-proc:only"/>
  <compilerarg value="-processor"/>
  <compilerarg value="com.mysema.query.apt.jdo.JDOAnnotationProcessor"/>
  <compilerarg value="-s"/>
  <compilerarg value="${generated}"/>
</javac>

<!-- compilation -->
<javac classpathref="cp" destdir="${build}">
  <src path="${src}"/>
  <src path="${generated}"/>
</javac>

```

src# ## ## ## ## ##, generated# ## ## ## ## ## ##, build# ## ## ## ## ##.

##

Querydsl# ##### ## ## ##, ### Query ##### ##. ## ## ## ##.

#####.

```
@PersistenceCapable
public class Customer {
    private String firstName;
    private String lastName;

    public String getFirstName(){
        return firstName;
    }

    public String getLastName(){
        return lastName;
    }

    public void setFirstName(String fn){
        firstName = fn;
    }

    public void setLastName(String ln){
        lastName = ln;
    }
}
```

Querydsl# Customer# ### ##### QCustomer## ### ## ## ### #####. Querydsl ##### Customer ### ## ##
QCustomer# #####.

QCustomer# ## ##### ## ## ##, ### ## ## ## ## ## # ##.

```
QCustomer customer = QCustomer.customer;
```

Customer ### ## ## ## ##.

```
QCustomer customer = new QCustomer("myCustomer");
```

QCustomer# ## Customer ### ## ##### public ### #####. firstName ### ### ## ## ## # ##.

```
customer.firstName;
```

##

JDOQuery# JDO ### ## Query #####, ### ## ##### #####.

```
PersistenceManager pm = ...;
JDOQuery query = new JDOQuery (pm);
```

firstName ##### Bob# Customer# ##### ### ## ## ## ## ##.

```
QCustomer customer = QCustomer.customer;
JDOQuery query = new JDOQuery (pm);
Customer bob = query.from(customer)
    .where(customer.firstName.eq("Bob"))
    .uniqueResult(customer);
query.close();
```

from ##### ## ##(##)#####, where ##### ## ## ## ##, uniqueResult# ##### ##### 1# ##### #####.

##.

```
QCustomer customer = QCustomer.customer;
QCompany company = QCompany.company;
query.from(customer, company);
```

##.

```
query.from(customer)
    .where(customer.firstName.eq("Bob"), customer.lastName.eq("Wilson"));
```

##, ## ## ## ##.

```
query.from(customer)
    .where(customer.firstName.eq("Bob").and(customer.lastName.eq("Wilson")));
```

or# ##### ## ## ## ##.

```
query.from(customer)
    .where(customer.firstName.eq("Bob").or(customer.lastName.eq("Wilson")));
```

##

JDOQuery ##### cascading ##### ## ##.

from: ## #####. # ## ## ## ## ##, ##### ## ##.

where: ## #####. ##### and/or ##### ## ##.

groupBy: ##### ## ## ## ## ##.

having: Predicate ##### "group by" ##### ## ##.

orderBy: ## ##### ## ## ##. ## ## ## asc()# desc()# #####, OrderSpecifier# ##### ## ## #
#####.

limit, offset, restrict: ## #####. limit# ## ## ##, offset# ## ## #, restrict# limit# offset# ## ##.

##

##.

```
QCustomer customer = QCustomer.customer;
query.from(customer)
    .orderBy(customer.lastName.asc(), customer.firstName.desc())
    .list(customer);
```

###

#####.

```
query.from(customer)
    .groupBy(customer.lastName)
    .list(customer.lastName);
```

DeleteClause

Querydsl JDO## DeleteClause# ### delete-where-execute ### ###. ### # ## ##.

```
QCustomer customer = QCustomer.customer;
// delete all customers
new JDODeleteClause(pm, customer).execute();
// delete all customers with a level less than 3
new JDODeleteClause(pm, customer).where(customer.level.lt(3)).execute();
```

JDODeleteClause ##### # ## ##### ### ### #####. where# ### ## ### # ###, execute# ##### ### ##### #
#####.

####

JDOSubQuery# ##### ##. ##### ## ## from ##### ## ##### ##, unique# list# #####. unique
list# ### ## ## ##. ##### ## ## ## Querydsl #####.

```
QDepartment department = QDepartment.department;
QDepartment d = new QDepartment("d");
query.from(department)
    .where(department.employees.size().eq(
        new JDOSubQuery().from(d).unique(AggregationFunctions.max(d.employees.size()))
    )).list(department);
```

JDO ### #####.

```
SELECT this FROM com.mysema.query.jdoql.models.company.Department
WHERE this.employees.size() ==
```

```
(SELECT max(d.employees.size()) FROM com.mysema.query.jdoql.models.company.Department d)
```

##

```
QEmployee employee = QEmployee.employee;
QEmployee e = new QEmployee("e");
query.from(employee)
    .where(employee.weeklyhours.gt(
        new JDOSubQuery().from(employee.department.employees, e)
        .where(e.manager.eq(employee.manager))
        .unique(AggregationFunctions.avg(e.weeklyhours))
    )).list(employee);
```

JDO ### ##.

```
SELECT this FROM com.mysema.query.jdoql.models.company.Employee
WHERE this.weeklyhours >
(SELECT avg(e.weeklyhours) FROM this.department.employees e WHERE e.manager == this.manager)
```

SQL

JDOSQLQuery ##### JDO# ##### SQL# Querydsl## ### # ##.

SQL ##### ## Querydsl ## ### ##### ##. ### ## ## Maven ## ## ##### ##.

```
<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.querydsl</groupId>
        <artifactId>querydsl-maven-plugin</artifactId>
        <version>${querydsl.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>export</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <jdbcDriver>org.apache.derby.jdbc.EmbeddedDriver</jdbcDriver>
          <jdbcUrl>jdbc:derby:target/demoDB;create=true</jdbcUrl>
          <packageName>com.mycompany.mydomain</packageName>
          <targetFolder>${project.basedir}/target/generated-sources/java</targetFolder>
        </configuration>
        <dependencies>
          <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derby</artifactId>
            <version>${derby.version}</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
</project>
```

```

        </dependencies>
    </plugin>

    ...
</plugins>
</build>
</project>

```

##.

##:

```

// serialization templates
SQLTemplates templates = new DerbyTemplates();
// query types (S* for SQL, Q* for domain types)
SAnimal cat = new SAnimal("cat");
SAnimal mate = new SAnimal("mate");

JDOSQLQuery query = new JDOSQLQuery(pm, templates);
List<String> names = query.from(cat).list(cat.name);

```

##:

```

query = new JDOSQLQuery(pm, templates);
List<Object[]> rows = query.from(cat).list(cat.id, cat.name);

```

##:

```

List<Object[]> rows = query.from(cat).list(cat.all());

```

##:

```

query = new JDOSQLQuery(pm, templates);
cats = query.from(cat)
    .innerJoin(mate).on(cat.mateId.eq(mate.id))
    .where(cat.dtype.eq("Cat"), mate.dtype.eq("Cat"))
    .list(catEntity);

```

DTO# ##:

```

query = new JDOSQLQuery(pm, templates);
List<CatDTO> catDTOS = query.from(cat)
    .orderBy(cat.name.asc())
    .list(ConstructorExpression.create(CatDTO.class, cat.id, cat.name));

```

2.3. SQL

SQL ## ## ## ## ## ## ## ## ## ##.

##

##.

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-sql</artifactId>
  <version>${querydsl.version}</version>
</dependency>

<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-sql-codegen</artifactId>
  <version>${querydsl.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.6.1</version>
</dependency>
```

Ant## # ## querydsl-sql-codegen ### ## # ##.

##

##. ### ## ##.

```
<project>
  <build>
    <plugins>
      ...
    <plugin>
      <groupId>com.mysema.querydsl</groupId>
      <artifactId>querydsl-maven-plugin</artifactId>
      <version>${querydsl.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>export</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <jdbcDriver>org.apache.derby.jdbc.EmbeddedDriver</jdbcDriver>
        <jdbcUrl>jdbc:derby:target/demoDB;create=true</jdbcUrl>
        <packageName>com.myproject.domain</packageName>
        <targetFolder>${project.basedir}/target/generated-sources/java</targetFolder>
      </configuration>
      <dependencies>
        <dependency>
          <groupId>org.apache.derby</groupId>
```



```

        <artifactId>derby</artifactId>
        <version>${derby.version}</version>
    </dependency>
</dependencies>
</plugin>
...
</plugins>
</build>
</project>

```

targetFolder# ##### test-export ## ##### ##.

2.1.

##	##
jdbcDriver	JDBC ##### ## ##
jdbcUrl	JDBC URL
jdbcUser	JDBC ###
jdbcPassword	JDBC ##
namePrefix	### ## ##### ## (##: Q)
nameSuffix	### ## ##### ## (##:)
beanPrefix	### #Bean ##### ##
beanSuffix	### # ##### ##
packageName	### ## ## ## ##
beanPackageName	# ### ## ## ## (##: packageName)
beanInterfaces	# ##### ## ##### ## (##: ##)
beanAddToString	true# ##### ## toString() ### ## (##: false)
beanAddFullConstructor	true# ##### ## ## ## ## ## ## (##: false)
beanPrintSupertype	true# ##### ## ## ## ## (##: false)
schemaPattern	### ## ##. ### ##### ##### ## ## ## ## ##. (# #: null)
tableNamePattern	### ## ##. ### ##### ##### ## ## ## ## ##, ## # ##### # # ## ## ## ##. (##: null)
targetFolder	## ## ## ## ##
namingStrategyClass	NamingStrategy# ### ## ## ## (##: DefaultNamingStrategy)

##	##
beanSerializerClass	BeanSerializer# ### ## (##: BeanSerializer)
serializerClass	Serializer# ### ## (##: MetadataSerializer)
exportBeans	true# ##### ## ## ##. 2.14.13 ##. (##: false)
innerClassesForKeys	true# ##### ## ## ## (##: false)
validationAnnotations	true# ##### Validation ##### ## (##: false)
columnAnnotations	true# ##### ## ##### ## (##: false)
createScalaSources	true# ##### ## ## ## Scala ### ## (##: false)
schemaToPackage	true# ##### ## ## ## (##: false)
lowerCase	true# ##### ## ## ## (##: false)
exportTables	true# ##### ## ## (##: true)
exportViews	true# ### ## ## (##: true)
exportPrimaryKeys	true# ##### PK# ## (##: true)
exportForeignKeys	true# ##### ## ## (##: true)
customTypes	### ## ## (##: ##)
typeMappings	####.#### ## ## ## (##: ##)
numericMappings	##/##### ## ## ## (##: ##)
imports	### ## ##### ## ## import ##: ##### ## (. * ##) ### ##(, com.bar), ##### ## ## ## ##(, com.bar.Foo) ##. (##: ##)

customTypes# #####.

```
<customTypes>
  <customType>com.mysema.query.sql.types.InputStreamType</customType>
</customTypes>
```

####.### ## ## ## ## ##### ## # typeMappings# #####.

```
<typeMappings>
  <typeMapping>
    <table>IMAGE</table>
    <column>CONTENTS</column>
```

```
<type>java.io.InputStream</type>
</typeMapping>
</typeMappings>
```

##.

2.2. ##

##	##(Digits)	##
> 18	0	BigInteger
> 9	0	Long
> 4	0	Integer
> 2	0	Short
> 0	0	Byte
> 16	> 0	BigDecimal
> 0	> 0	Double

##/### ## ### ### ## ## ##.

```
<numericMappings>
  <numericMapping>
    <size>1</size>
    <digits>0</digits>
    <javaType>java.lang.Byte</javaType>
  </numericMapping>
</numericMappings>
```

Import# ##### ## ## ## ## ## ## # ##.

APT ## ## ## ## ## # ## ## ## ## # ##. (#, QueryDelegate ##### ##)

ANT# ## ##

Querydsl-sql ##### com.mysema.query.sql.ant.AntMetaDataExporter ANT ##### ANT
####(## ANT ####?)# ## ### #####. ##### ## ##### ## ## ## ##.

##

DB ##### Querydsl# ## ##### ##### ## ## ##.

```
java.sql.Connection conn = ...;
MetaDataExporter exporter = new MetaDataExporter();
```

```
exporter.setPackageName("com.myproject.mydomain");
exporter.setTargetFolder(new File("target/generated-sources/java"));
exporter.export(conn.getMetaData());
```

#####(com.myproject.mydomain #####)# target/generated-sources/
java #####.

#####, ## ## ##### ## ## ## ##.

###, ### ## ## PK# FK# ## ## ##.

##

com.mysema.query.sql.Configuration #####, Configuration ##### Querydsl SQL Dialect
###, H2 DB ### ## ##.

```
SQLTemplates templates = new H2Templates();
Configuration configuration = new Configuration(templates);
```

Querydsl# ## ## RDBMS# ## SQL ##### ## SQL Dialect# #####. ##### Dialect# ## ##.

- CUBRIDTemplates (tested with CUBRID 8.4)
- DerbyTemplates (tested with Derby 10.8.2.2)
- FirebirdTemplates (tested with Firebird 2.5)
- HSQLDBTemplates (tested with HSQLDB 2.2.4)
- H2Templates (tested with H2 1.3.164)
- MySQLTemplates (tested with MySQL 5.5)
- OracleTemplates (test with Oracle 10 and 11)
- PostgresTemplates (tested with PostgreSQL 9.1)
- SQLiteTemplates (tested with xerial JDBC 3.7.2)
- SQLServerTemplates (tested with SQL Server)
- SQLServer2005Templates (for SQL Server 2005)
- SQLServer2008Templates (for SQL Server 2008)
- SQLServer2012Templates (for SQL Server 2012 and later)
- TeradataTemplates (tested with Teradata 14)

SQLTemplate ### ### ##### ### ## ## ### ### # ##.

```
H2Templates.builder()
    .printSchema() // to include the schema in the output
    .quote()       // to quote names
    .newLineToSingleSpace() // to replace new lines with single space in the output
    .escape(ch)    // to set the escape char
    .build();      // to get the customized SQLTemplates instance
```

Configuration ##### setUseLiterals(true)# ## ##### ## ### ##, ##### ### ###, ### ### ### # ##. ###
javadoc# Configuration# #####.

##

Querydsl SQL# ##### ### ##### ##.

```
QCustomer customer = new QCustomer("c");

SQLQuery query = new SQLQuery(connection, configuration);
List<String> lastNames = query.from(customer)
    .where(customer.firstName.eq("Bob"))
    .list(customer.lastName);
```

SQL# #####. (### ### customer, ## ### first_name, last_name### ##)

```
SELECT c.last_name
FROM customer c
WHERE c.first_name = 'Bob'
```

##

SQLQuery ##### cascading ##### ## ##.

from: ## ### #####.

innerJoin, join, leftJoin, fullJoin, on: ## ### #####. ## ##### # ## ### ## #####, # ## ### ##(##)##.

where: ## ### #####. ##### and/or ##### ##### ## #####.

groupBy: ##### ## ## ##### ## ##.

having: Predicate ##### "group by" ##### ## ##.

orderBy: ## ##### ## ## ##. ## ##### asc()# desc()# #####, OrderSpecifier# ##### ## ## #
#####.

limit, offset, restrict: #####. limit# ## ## ##, offset# ## ## #, restrict# limit# offset# ## ##.

##

##.

```
QCustomer customer = QCustomer.customer;
QCompany company = QCompany.company;
query.from(customer)
    .innerJoin(customer.company, company)
    .list(customer.firstName, customer.lastName, company.name);
```

##.

```
query.from(customer)
    .leftJoin(customer.company, company)
    .list(customer.firstName, customer.lastName, company.name);
```

##.

```
query.from(customer)
    .leftJoin(company).on(customer.company.eq(company.id))
    .list(customer.firstName, customer.lastName, company.name);
```

##

#####.

```
query.from(customer)
    .orderBy(customer.lastName.asc(), customer.firstName.asc())
    .list(customer.firstName, customer.lastName);
```

SQL ### #####.

```
SELECT c.first_name, c.last_name
FROM customer c
ORDER BY c.last_name ASC, c.first_name ASC
```

###

##.

```
query.from(customer)
    .groupBy(customer.lastName)
    .list(customer.lastName);
```

SQL ###.

```
SELECT c.last_name
FROM customer c
GROUP BY c.last_name
```

####

SQLSubQuery# ##### ##. ##### ## from ##### ## ##### ##, unique# list# #####. unique
list# ## ## ## ##. ##### ## ## ## Querydsl #####.

```
QCustomer customer = QCustomer.customer;
QCustomer customer2 = new QCustomer("customer2");
query.from(customer).where(
    customer.status.eq(new SQLSubQuery().from(customer2).unique(customer2.status.max()))
    .list(customer.all())
```

##

```
QStatus status = QStatus.status;
query.from(customer).where(
    customer.status.in(new SQLSubQuery().from(status).where(status.level.lt(3)).list(status.id))
    .list(customer.all())
```

##

##, ## ## constant ##### ##.

```
query.list(Expressions.constant(1),
    Expressions.constant("abc"));
```

com.mysema.query.support.Expressions #####, #####, ## ## ## ## ## ## ##.

##

##. ### ## ## AbstractSQLQuery# ##### ## # ##. ### MySQLQuery
##.

```
public class MySQLQuery extends AbstractSQLQuery<MySQLQuery> {

    public MySQLQuery(Connection conn) {
        this(conn, new MySQLTemplates(), new DefaultQueryMetadata());
    }

    public MySQLQuery(Connection conn, SQLTemplates templates) {
        this(conn, templates, new DefaultQueryMetadata());
    }
}
```

```

protected MySQLQuery(Connection conn, SQLTemplates templates, QueryMetadata metadata) {
    super(conn, new Configuration(templates), metadata);
}

public MySQLQuery bigResult(){
    return addFlag(Position.AFTER_SELECT, "SQL_BIG_RESULT ");
}

public MySQLQuery bufferResult(){
    return addFlag(Position.AFTER_SELECT, "SQL_BUFFER_RESULT ");
}

// ...
}

```

SQL ## ##. com.mysema.query.QueryFlag.Position #
##.

##

Querydsl# SQLExpressions #### ## ## ## ## ##.

##.

```

query.from(employee)
    .list(SQLExpressions.rowNumber()
        .over()
        .partitionBy(employee.name)
        .orderBy(employee.id));

```

SQL

SQLExpressions #### ## ## ## ## ## SQL #### ## ##.

DML ## ####

Querydsl SQL ## ## DMLClause #### Connection, ## ## SQLTemplate, DMLClause# ## ## ## ## #
##.

##

##

```

QSurvey survey = QSurvey.survey;

new SQLInsertClause(conn, configuration, survey)
    .columns(survey.id, survey.name)
    .values(3, "Hello").execute();

```


##

```
new SQLInsertClause(conn, configuration, survey)
    .values(4, "Hello").execute();
```

##

```
new SQLInsertClause(conn, configuration, survey)
    .columns(survey.id, survey.name)
    .select(new SQLSubQuery().from(survey2).list(survey2.id.add(1), survey2.name))
    .execute();
```

##, ##

```
new SQLInsertClause(conn, configuration, survey)
    .select(new SQLSubQuery().from(survey2).list(survey2.id.add(10), survey2.name))
    .execute();
```

columns/values# #### ##, set #### ##

```
QSurvey survey = QSurvey.survey;

new SQLInsertClause(conn, configuration, survey)
    .set(survey.id, 3)
    .set(survey.name, "Hello").execute();
```

#####. set #### #### # columns/values# ####.

##.

```
columns(...).select(...)
```

executeWithKey/s #### ##.

```
set(...)
```

#####. ## ## ## ## null# ####.

clause #### # ## ## ##.

```
new SQLInsertClause(conn, configuration, survey)
    .populate(surveyBean).execute();
```

null# ####. null# ##### # ## ## ##.

```
new SQLInsertClause(conn, configuration, survey)
    .populate(surveyBean, DefaultMapper.WITH_NULL_BINDINGS).execute();
```

##

where # ##

```
QSurvey survey = QSurvey.survey;

new SQLUpdateClause(conn, configuration, survey)
    .where(survey.name.eq("XXX"))
    .set(survey.name, "S")
    .execute();
```

where # ##

```
new SQLUpdateClause(conn, configuration, survey)
    .set(survey.name, "S")
    .execute();
```

##

```
new SQLUpdateClause(conn, configuration, survey)
    .populate(surveyBean)
    .execute();
```

##

where # ##

```
QSurvey survey = QSurvey.survey;

new SQLDeleteClause(conn, configuration, survey)
    .where(survey.name.eq("XXX"))
    .execute();
```

where ##

```
new SQLDeleteClause(conn, configuration, survey)
    .execute();
```

DMLClause# ##

Querydsl SQL# DML API# ### JDBC ## ##### #####. ## ### ## DML# ##### ### ##, addBatch() ##### #
DMLClause# ## # ##. UPDATE, DELETE, INSERT# ## ### ##### ### #####.

##:

```
QSurvey survey = QSurvey.survey;

insert(survey).values(2, "A").execute();
insert(survey).values(3, "B").execute();

SQLUpdateClause update = update(survey);
update.set(survey.name, "AA").where(survey.name.eq("A")).addBatch();
update.set(survey.name, "BB").where(survey.name.eq("B")).addBatch();
```

##:

```
insert(survey).values(2, "A").execute();
insert(survey).values(3, "B").execute();

SQLDeleteClause delete = delete(survey);
delete.where(survey.name.eq("A")).addBatch();
delete.where(survey.name.eq("B")).addBatch();
assertEquals(2, delete.execute());
```

##:

```
SQLInsertClause insert = insert(survey);
insert.set(survey.id, 5).set(survey.name, "5").addBatch();
insert.set(survey.id, 6).set(survey.name, "6").addBatch();
assertEquals(2, insert.execute());
```

##

MetaDataExporter# ##### ## ### DTO ### #####.

```
java.sql.Connection conn = ...;
MetaDataExporter exporter = new MetaDataExporter();
exporter.setPackageName("com.myproject.mydomain");
exporter.setTargetFolder(new File("src/main/java"));
exporter.setBeanSerializer(new BeanSerializer());
exporter.export(conn.getMetaData());
```

DMLClause# populate ##### ## # ### ## # ###, ##### # ### ## ### # ##. ### JUnit## ### ## ##.

```
QEmployee e = new QEmployee("e");

// Insert
Employee employee = new Employee();
employee.setFirstname("John");
Integer id = insert(e).populate(employee).executeWithKey(e.id);
employee.setId(id);
```

```
// Update
employee.setLastname("Smith");
assertEquals(11, update(e).populate(employee).where(e.id.eq(employee.getId())).execute());

// Query
Employee smith = query().from(e).where(e.lastname.eq("Smith")).uniqueResult(e);
assertEquals("John", smith.getFirstname());

// Delete
assertEquals(11, delete(e).where(e.id.eq(employee.getId())).execute());
```

##.

```
protected SQLUpdateClause update(RelationalPath<?> e){
    return new SQLUpdateClause(Connections.getConnection(), templates, e);
}

protected SQLInsertClause insert(RelationalPath<?> e){
    return new SQLInsertClause(Connections.getConnection(), templates, e);
}

protected SQLDeleteClause delete(RelationalPath<?> e){
    return new SQLDeleteClause(Connections.getConnection(), templates, e);
}

protected SQLMergeClause merge(RelationalPath<?> e){
    return new SQLMergeClause(Connections.getConnection(), templates, e);
}

protected SQLQuery query() {
    return new SQLQuery(Connections.getConnection(), templates);
}
```

SQL ### ###

getSQL ##### SQL ### ### ## ## # ##.

```
SQLBindings bindings = query.getSQL(customer.id, customer.firstname, customer.lastname);
System.out.println(bindings.getSQL());
```

SQL ##### ## ## #####, setUseLiterals(true)# ##### ## ## ##### ##.

##

Querydsl SQL# ResultSet/Statement## ### ## ### #####. com.mysema.query.sql.Configuration ### #####
#####. Configuration ### ## ### ## ## ##.

```
Configuration configuration = new Configuration(new H2Templates());
// overrides the mapping for Types.DATE
```

```
configuration.register(new UtilDateType());
```

```
## ### ## ## ## ## ##
```

```
Configuration configuration = new Configuration(new H2Templates());
// declares a mapping for the gender column in the person table
configuration.register("person", "gender", new EnumByNameType<Gender>(Gender.class));
```

```
### ## ## ## ## ## ## registerNumeric #####
```

```
configuration.registerNumeric(5,2,Float.class);
```

```
## Float ## NUMERIC(5,2) #####
```

Query# Clause

```
SQLListener# ### DMLClause# ### # # ##### ### #####. Configuration## Query, Clause# addListener
##### SQLListener ### ## # #.
```

```
##### ## ## ## ## ##, ##, ##, ## ##.
```

2.4. ##

```
# ##### ## ## ## ## ##.
```

Maven integration

```
Querydsl ### ##### ## 3# querydsl-lucene3 ### ## ## 4# querydsl-lucene4 ### #####.
```

```
## 3:
```

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-lucene3</artifactId>
  <version>${querydsl.version}</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.6.1</version>
</dependency>
```

```
## 4:
```

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
```

```

<artifactId>querydsl-lucene4</artifactId>
<version>${querydsl.version}</version>
</dependency>

<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>1.6.1</version>
</dependency>

```

##

year# title ### ## ## #### ## #### # ##.

```

public class QDocument extends EntityPathBase<Document>{
    private static final long serialVersionUID = -4872833626508344081L;

    public QDocument(String var) {
        super(Document.class, PathMetadataFactory.forVariable(var));
    }

    public final StringPath year = createString("year");

    public final StringPath title = createString("title");
}

```

QDocument# year# title ### ## ## Document# #####.

##.

##

Querydsl ##### ## #####.

```

QDocument doc = new QDocument("doc");

IndexSearcher searcher = new IndexSearcher(index);
LuceneQuery query = new LuceneQuery(true, searcher);
List<Document> documents = query
    .where(doc.year.between("1800", "2000").and(doc.title.startsWith("Huckle")))
    .list();

```

##.

```
+year:[1800 TO 2000] +title:huckle*
```

##

LuceneQuery ##### cascading ##### ## ##.

where: ## ### ####, ##### and/or ##### #### #. PStrings# ##### #### (matches, indexOf, charAt# #). ## in# ##### #, ## ### ####.

orderBy: ## ##### ## ### ####, ### ##### asc()# desc()# ####, OrderSpecifier# ##### # # # # ##### ####.

limit, offset, restrict: ### #####. limit# ## # #, offset# ### # #, restrict# limit# offset# ## ####.

##

#.

```
query
    .where(doc.title.like("*"))
    .orderBy(doc.title.asc(), doc.year.desc())
    .list();
```

####.

```
title:*
```

title# year# ##### ## ####.

sort ##### Sort ##### ##### ## ## # #.

```
Sort sort = ...;
query
    .where(doc.title.like("*"))
    .sort(sort)
    .list();
```

##

#.

```
query
    .where(doc.title.like("*"))
    .limit(10)
    .list();
```

###

####.

```
query
```

```
.where(doc.title.like("*"))
.offset(3)
.list();
```

##(fuzzy)

com.mysema.query.lucene.LuceneExpressions ##### fuzzyLike ##### ## ## # #
##.

```
query
    .where(LuceneExpressions.fuzzyLike(doc.title, "Hello"))
    .list();
```

#####

##.

```
query
    .where(doc.title.like("*"))
    .filter(filter)
    .list();
```

distinct ##### ## distinct(Path) ##### #####.

```
query
    .where(doc.title.like("*"))
    .distinct(doc.title)
    .list();
```

2.5. Hibernate Search

Hibernate Search ### ## ### #####.

Querydsl ## ##

[JPA ##](#) ##### #####.

##

Querydsl Hibernate Search# ### ### ## ## #####.

```
QUser user = QUser.user;
SearchQuery<User> query = new SearchQuery<User>(session, user);
List<User> list = query
    .where(user.firstName.eq("Bob"))
    .list();
```


##

[Querying Lucene](#) ## ### #####.

Querydsl Lucene module## ### ##### ### ##### ##.
org.hibernate.search.annotations.Field ##### ##, ## ### ## ## ##### name
#####.

2.6. Mongoddb

Mongoddb ### ## ### #####.

##

#####.

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-apt</artifactId>
  <version>${querydsl.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-mongoddb</artifactId>
  <version>${querydsl.version}</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.6.1</version>
</dependency>
```

Querydsl# ##### ## ### ##### ## ### APT ##### #####.

```
<project>
  <build>
    <plugins>
      ...
    <plugin>
      <groupId>com.mysema.maven</groupId>
      <artifactId>apt-maven-plugin</artifactId>
      <version>1.0.9</version>
      <executions>
        <execution>
          <goals>
            <goal>process</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </build>
</project>
```

```
<outputDirectory>target/generated-sources/java</outputDirectory>
<processor>com.mysema.query.apt.morphia.MorphiaAnnotationProcessor</processor>
</configuration>
</execution>
</executions>
</plugin>
...
</plugins>
</build>
</project>
```

```
MorphiaAnnotationProcessor# com.google.code.morphia.annotations.Entity ##### ##
# ### ### ## Querydsl ## ### #####.
```

```
mvn clean install # ##### target/generated-sources/java ### ## ## ### #####.
```

```
##### ##, mvn eclipse:eclipse # ##### target/generated-sources/java# ##### ## ## ##.
```

```
## ## #### #### ##### MongoDB# #### # ##.
```

##

Querydsl MongoDB# ##### ## ##### ## # ##.

```
Morphia morphia;
Datastore datastore;
// ...
QUser user = new QUser("user");
MorphiaQuery<User> query = new MorphiaQuery<User>(morphia, datastore, user);
List<User> list = query
    .where(user.firstName.eq("Bob"))
    .list();
```

##

MongodbQuery ##### cascading ##### ##.

where: ## ### #####. ##### and/or ##### ##### ### #####. PStrings# ##### #####. (matches, indexOf, charAt# ##). ## in# ##### ##, ## ### #####.

```
orderBy: ## ##### ##### ## ### #####. ### ##### ##### asc()# desc()# #####, OrderSpecifier# ##### ## ## #
# ##### #####.
```

limit, offset, restrict: ### #####. limit# ## ## #, offset# ### ## #, restrict# limit# offset# ## #####.

##

##.

```
query
    .where(doc.title.like("*"))
    .orderBy(doc.title.asc(), doc.year.desc())
    .list();
```

title# year# ##### ## ##.

##

##.

```
query
    .where(doc.title.like("*"))
    .limit(10)
    .list();
```

###

##.

```
query
    .where(doc.title.like("*"))
    .offset(3)
    .list();
```

##(Geospatial) ##

near(Douyble[]) ##### ## ## # # ##.

```
query
    .where(gEOEntity.location.near(50.0, 50.0))
    .list();
```

##

##, ## ## ## ## list, iterate, uniqueResult, singleResult ##### ##.

```
query
    .where(doc.title.like("*"))
    .list(doc.title, doc.path);
```

title# path ## ##.

2.7. ##

querydsl-collections ##### # ##. # ## ## ## ## ## ## ## ## ## ##.

#####

querydsl-collections ### #####, Querydsl ## ### ##### ##. ### # ## ##.

#####.

```
// needed for access of the Querydsl Collections API
import static com.mysema.query.collections.CollQueryFactory.*;
// needed, if you use the $-invocations
import static com.mysema.query.alias.Alias.*;
```

Cat ##### ## ## ##### #####. ## ##### ## non-final ##### ##### Alias ##### ## ##.

\$ ##### ## ## ## Cat ### ## ##### # ### getter ### ## ## ##. ## ##, c.getKittens()# ## ## \$
c.kittends ### ##.

```
Cat c = alias(Cat.class, "cat");
for (String name : from$(c),cats)
    .where$(c.getKittens().size().gt(0))
    .list$(c.getName())){
    System.out.println(name);
}
```

##. ## ## List# size() ##### \$ ##### ##.

```
Cat c = alias(Cat.class, "cat");
for (String name : from$(c),cats)
    .where$(c.getKittens().size().gt(0))
    .list$(c.getName())){
    System.out.println(name);
}
```

#-##### non-final ## ##### ## # #####. ###, \$ ### ##### ## ##### non-final ## (#, java.lang.String)
##.

##,

```
$(c.getMate().getName())
```

c.mate.name## ###. ###, ## ### ##### ## ##.

```
$(c.getMate().getName().toLowerCase())
```

toLowerCase() ### ##### ## #####.

getter, size(), contains(Object), get(int) # ### # ##. ### ## ##### ## ## ##### #####.

#####

##.

```
QCat cat = new QCat("cat");
for (String name : from(cat,cats)
    .where(cat.kittens.size().gt(0))
    .list(cat.name)){
    System.out.println(name);
}
```

#####, ## ##### ## ##### ### # ## \$ ##### ### ## ## ##### ### ## ### # ##.

##

#####.

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-apt</artifactId>
  <version>${querydsl.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-collections</artifactId>
  <version>${querydsl.version}</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.6.1</version>
</dependency>
```

JPA# JDO# ##### ##, ### ## com.mysema.query.annotations.QueryEntity ##### ##
##(pom.xml)# ## ##### ## ##### ## ## ## # ##.

```
<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.maven</groupId>
        <artifactId>apt-maven-plugin</artifactId>
        <version>1.0.9</version>
        <executions>
          <execution>
            <goals>
              <goal>process</goal>
```

```

        </goals>
        <configuration>
            <outputDirectory>target/generated-sources/java</outputDirectory>
            <processor>com.mysema.query.apt.QuerydslAnnotationProcessor</processor>
        </configuration>
    </execution>
</executions>
</plugin>
...
</plugins>
</build>
</project>

```

Ant

full-deps# ### jar ##### ## ##### Querydsl ### #####.

```

<!-- APT based code generation -->
<javac srcdir="${src}" classpathref="cp">
    <compilerarg value="-proc:only"/>
    <compilerarg value="-processor"/>
    <compilerarg value="com.mysema.query.apt.QuerydslAnnotationProcessor"/>
    <compilerarg value="-s"/>
    <compilerarg value="${generated}"/>
</javac>

<!-- compilation -->
<javac classpathref="cp" destdir="${build}">
    <src path="${src}"/>
    <src path="${generated}"/>
</javac>

```

src# ## ## #####, generated# ### ## ## ## ##, build# ### ## ## ##.

Hamcrest matchers

Querydsl Collections ### Hamcrest matchers# #####. ### import# ## ##### ##.

```

import static org.hamcrest.core.IsEqual.equalTo;
import static com.mysema.query.collections.PathMatcher.hasValue;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertThat;

```

##.

```

Car car = new Car();
car.setHorsePower(123);

assertThat(car, hasValue($.horsePower));
assertThat(car, hasValue($.horsePower, equalTo(123)));

```

[Jeroen van Schagen](#) # Hamcrest matchers # ####.

2.8. Scala ##

querydsl-scala ### ## Scala ## Querydsl # ### # ##. # ### ##### ### ### ## ## ##.

```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-scala</artifactId>
  <version>${querydsl.version}</version>
</dependency>
```

Scala # ## DSL

Scala # Querydsl # ### ## ## DSL # #####. Scala DSL # ##### ##### ##### ## ## ## ##, ## ##, ## #
##.

DSL # ## ##.

//Standard	Alternative
expr isNotNull	expr is not(null)
expr isNull	expr is null
expr eq "Ben"	expr == "Ben"
expr ne "Ben"	expr != "Ben"
expr append "X"	expr + "X"
expr isEmpty	expr is empty
expr isNotEmpoty	expr not empty
// boolean	
left and right	left && right
left or right	left right
expr not	!expr
// comparison	
expr lt 5	expr < 5
expr loe 5	expr <= 5
expr gt 5	expr > 5
expr goe 5	expr >= 5
expr notBetween(2,6)	expr not between (2,6)
expr negate	-expr
// numeric	
expr add 3	expr + 3
expr subtract 3	expr - 3
expr divide 3	expr / 3
expr multiply 3	expr * 3
expr mod 5	expr % 5
// collection	
list.get(0)	list(0)
map.get("X")	map("X")

Querydsl Scala ### Querydsl# ## ##### Scala# ## ### ### ## ## ## ### #####.

```
## ##,## API# ### ## ### Object[] ### java.util.List# #####.
```

##.

```
# ## ## #### List[(String,String,Integer)) #, Tuple3[String,String,Integer)# List# ##.
```

Querydsl SQL# #####,### ##### ## ## ## ##.### ## ## ## ## ## ##.

```
val directory = new java.io.File("target/jdbcgen1")
val namingStrategy = new DefaultNamingStrategy()
val exporter = new MetadataExporter()
exporter.setNamePrefix("Q")
exporter.setPackageName("com.mysema")
exporter.setSchemaPattern("PUBLIC")
exporter.setTargetFolder(directory)
exporter.setSerializerClass(classOf[ScalaMetadataSerializer])
exporter.setCreateScalaSources(true)
exporter.setTypeMappings(ScalaTypeMappings.create)
exporter.export(connection.getMetaData)
```

```
val directory = new java.io.File("target/jdbcgen2")
val namingStrategy = new DefaultNamingStrategy()
val exporter = new MetadataExporter()
exporter.setNamePrefix("Q")
exporter.setPackageName("com.mysema")
```



```

exporter.setSchemaPattern("PUBLIC")
exporter.setTargetFolder(directory)
exporter.setSerializerClass(classOf[ScalaMetaDataSerializer])
exporter.setBeanSerializerClass(classOf[ScalaBeanSerializer])
exporter.setCreateScalaSources(true)
exporter.setTypeMappings(ScalaTypeMappings.create)
exporter.export(connection.getMetaData)

```

##

Querydsl Scala# Querydls SQL# ## ### ## ## ##. # ### Rogue ##### ## ## ## ## ## ##.

RelationalPath ##### ## ## ## ## ## ## ## ##. ##### DAO #####
com.mysema.query.scala.sql.SQlHelpers ##### ## ## ## ##.

##, ## ## ## ## ## ## ## ## ##.

##

```

query().from(employee).select(employee.firstName, employee.lastName)

```

Employee ## QEmployee# companio ### ## ## ##.

```

Employee.select(_.firstName, _.lastName)

```

orderBy, where, select, single, unique# ##### ##, ## ## ## ## ## ## ## ## ## ## ##
#. ## ## ## ## ## ## ## ##.

```

Employee.select({ e => e.firstName }, { e => e.lastName })

```

com.mysema.query.scala.sql.RichSimpleQuery# ##### ## ## ##.

##

querydsl-maven-plugin# ##### SQL ##### ## Scala ### ## ##. ## ## ##.

```

<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.querydsl</groupId>
        <artifactId>querydsl-maven-plugin</artifactId>
        <version>${querydsl.version}</version>
        <configuration>
          <jdbcDriver>com.mysql.jdbc.Driver</jdbcDriver>
          <jdbcUrl>jdbc:mysql://localhost:3306/test</jdbcUrl>
          <jdbcUser>matko</jdbcUser>

```

```

    <jdbcPassword>matko</jdbcPassword>
    <packageName>com.example.schema</packageName>
    <targetFolder>${project.basedir}/src/main/scala</targetFolder>
    <exportBeans>true</exportBeans>
    <createScalaSources>true</createScalaSources>
  </configuration>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.16</version>
    </dependency>
    <dependency>
      <groupId>com.mysema.querydsl</groupId>
      <artifactId>querydsl-scala</artifactId>
      <version>${querydsl.version}</version>
    </dependency>
    <dependency>
      <groupId>org.scala-lang</groupId>
      <artifactId>scala-library</artifactId>
      <version>${scala.version}</version>
    </dependency>
  </dependencies>
</plugin>
...
</plugins>
</build>
</project>

```

querydsl:export ### ## #####.

##

##, Expression ### ##### ##### ## ### ##### ##.

JPA# ### ## ##.

```

@Entity
class User {
  @BeanProperty
  @Id
  var id: Integer = _;
  @BeanProperty
  var userName: String = _;
  @BeanProperty
  @ManyToOne
  var department: Department = _;
}

@Entity
class Department {
  @BeanProperty
  @Id
  var id: Integer = _;
  @BeanProperty

```

```
var name: String = _;  
}
```

##.

List

```
val person = Person as "person"  
  
query.from(person).where(person.firstName like "Rob%").list(person)
```

Unique result

```
query.from(person).where(person.firstName like "Rob%").unique(person)
```

Long where

```
query.from(person)  
  .where(person.firstName like "Rob%", person.lastName like "An%")  
  .list(person)
```

Order

```
query.from(person).orderBy(person.firstName asc).list(person)
```

Not null

```
query.from(person)  
  .where(person.firstName isEmpty, person.lastName isNotNull)  
  .list(person)
```

##

```
def query() = new JPAQuery(entityManager)
```

##.

```
val person = Person as "person"
```

XML ## #### ##, ## Scala #### #### # ##. HibernateDomainExporter# ## ## ## ## ## #####.

##.

```
Querydsl## Query# ##### ### ### #### query ##### #####. query ##### ### ## ### ## ##### #####, #
##### ##### ### ## ###.
```

#####. ## ## # ## #, ##### ## ## ## ##.

```
## ### ##### ##### com.mysema.query.BooleanBuilder ##### #####. # ##### Predicate# ##### #
# ### ## ##### ## # ##.
```

```
BooleanBuilder# ##### ##(mutable) ##### null#, # and ## or ## ### ##### ### #####.
```

[illegible]

##, ### ##### ### ##### ## Fluent DSL ### ### # ## ### ## Expressions

##.

O## #### ##### #### #### ## ## #### ##### ## # ##.

3.5.1

##

```
## ## ### ## com.mysema.query.types.path.PathBuilder ##### ## # ##. # ####  
EntityPathBase ##### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##.
```

Strign ####:

List ####:

##:

#####:

##:

Case

3.5.1

```
Expression<String> cases = new CaseBuilder()
    .when(customer.annualSpending.gt(10000)).then("Premier")
    .when(customer.annualSpending.gt(5000)).then("Gold")
    .when(customer.annualSpending.gt(2000)).then("Silver")
    .otherwise("Bronze");
// The cases expression can now be used in a projection or condition
```

equals-operations# ## case ##### ## ## ## ## ## ##.

```
QCustomer customer = QCustomer.customer;
Expression<String> cases = customer.annualSpending
    .when(10000).then("Premier")
    .when(5000).then("Gold")
    .when(2000).then("Silver")
    .otherwise("Bronze");
// The cases expression can now be used in a projection or condition
```

JDOQL### ## Case ##### ##### ##.

Casting

```
###      ####      ###      #####      ##      ##      ###      #####.      #      ##
#      ##      ###      ##      ###      com.mysema.query.types.path.EntityPathBase#
com.mysema.query.types.path.BeanPath# ## #####, ##### ## ##### ## ##### ##.
```

##, _super ## ## ## ## ## ## ## ##. ## ## ## # # ## ## ## ##, _super
##.

```
// from Account
QAccount extends EntityPathBase<Account>{
    // ...
}

// from BankAccount extends Account
QBankAccount extends EntityPathBase<BankAccount>{

    public final QAccount _super = new QAccount(this);

    // ...
}
```

```
## ##### ## ##### ##### EntityPathBase ##### as ##### ##### ##.
```

```
QAccount account = new QAccount("account");
QBankAccount bankAccount = account.as(QBankAccount.class);
```

##

Constant ##### ## ##### ### # ##. ### ## ##.

```
query.list(Expressions.constant(1),
           Expressions.constant("abc"));
```

Constant ##### ## #####.

3.2. ##

Querydsl# ## #### ##### ## ## # ## #### ## FactoryExpressions# #### ## ResultTransformer# ##### ##.

com.mysema.query.types.FactoryExpression ##### # ##, #### ## #### # #### ##### ## #
com.mysema.query.types.Projections ##### ##### FactoryExpression #### ##### # ##.

com.mysema.query.ResultTransformer ##### ## ##### GroupBy #####.

##

Querydsl 3.0 ## ## ## #### ## ## #### com.mysema.query.Tuple ##. Tuple# #### ##### Map# #####, #
Tuple # ##### ## ##### #### # ##.

```
List<Tuple> result = query.from(employee).list(employee.firstName, employee.lastName);
for (Tuple row : result) {
    System.out.println("firstName " + row.get(employee.firstName));
    System.out.println("lastName " + row.get(employee.lastName));
}
```

QTuple ##### ##### ## ## ## # ##.

```
List<Tuple> result = query.from(employee).list(new QTuple(employee.firstName, employee.lastName));
for (Tuple row : result) {
    System.out.println("firstName " + row.get(employee.firstName));
    System.out.println("lastName " + row.get(employee.lastName));
}
```

##(population)

####, Bean ##### ##### ##.

```
List<UserDTO> dtos = query.list(
    Projections.bean(UserDTO.class, user.firstName, user.lastName));
```

setter #### ## #### ## ##### #### ## #### ##### ##.

```
List<UserDTO> dtos = query.list(
    Projections.fields(UserDTO.class, user.firstName, user.lastName));
```

##

##.

```
List<UserDTO> dtos = query.list(
    Projections.bean(UserDTO.class, user.firstName, user.lastName));
```

##, QueryProjection ##### ### ##### ### ## ##.

```
class CustomerDTO {

    @QueryProjection
    public CustomerDTO(long id, String name){
        ...
    }

}
```

###, # ##### ### ## ##### ## #####.

```
QCustomer customer = QCustomer.customer;
JPQLQuery query = new HibernateQuery(session);
List<CustomerDTO> dtos = query.from(customer).list(new QCustomerDTO(customer.id, customer.name));
```

Hibernate# ##### ###, ## ## ##### # ### ### # ##.

QueryProjection ##### ### ## ##(@Entity) ### #####, ##### ### ##### ### # ##. ###, ##### ###
##(@Entity) ##### ## ### ## create ##### ##### ### ##### ## ### ##.

```
@Entity
class Customer {

    @QueryProjection
    public Customer(long id, String name){
        ...
    }

}
```

```
QCustomer customer = QCustomer.customer;
JPQLQuery query = new HibernateQuery(session);
List<Customer> dtos = query.from(customer).list(QCustomer.create(customer.id, customer.name));
```

###, ### ## ### ##### ### # ##.

```
List<Customer> dtos = query.from(customer)
```



```
.list(ConstructorExpression.create(Customer.class, customer.id, customer.name));
```

##(aggregation)

```
com.mysema.query.group.GroupBy ##### ##### ## ### ## ## ### ##### ## ### #####. ### ## ###.
```

###

```
import static com.mysema.query.group.GroupBy.*;

Map<Integer, List<Comment>> results = query.from(post, comment)
    .where(comment.post.id.eq(post.id))
    .transform(groupBy(post.id).as(list(comment)));
```

```
# ### post id# ### #####.
```

##

```
Map<Integer, Group> results = query.from(post, comment)
    .where(comment.post.id.eq(post.id))
    .transform(groupBy(post.id).as(post.name, set(comment.id)));
```

```
# ### post id# Group# #####. Group# post name# comment id# ###.
```

Group# GroupBy# ## Tuple ##### ## ### ##.

#####. .

3.3. ##

Querydsl# JPA, JDO, Mongoddb ##### ## ### ## ##6 APT ##### ## ### #####. # ##### ## ### ## ### #
APT# ## ### #####.

###

```
##### Querydsl# ## 2### ##### #####. # ## ### ##### ##,
com.mysema.query.annotations.QueryInit ##### ## ## ## ##. # ## ### ##### ## ##
### QueryInit ##### #####. ### ## ## ##### ##.
```

```
@Entity
class Event {
    @QueryInit("customer.address")
    Account account;
}
```

```

@Entity
class Account{
    Customer customer;
}

@Entity
class Customer{
    String name;
    Address address;
    // ...
}

```

```

### Event ## ## /# ##### #, account.customer ### ##### ##. ## ## ## ##### ##(customer.* #
## * #)###.

```

```

## ## ##### ## ##### ##, ## ## final### # ##. ## ## ## ## ## ## ## ## ## ## final ## #
## ## ## ## ## ##.

```

```

##### ## ## ## ##, ## ## Config ##### ## ## ## ## ##.

```

#####

```

##### ## Config ##### ## Querydsl# ##### ##. Querydsl# ##### ## ## ## ## ##
# #####.

```

```

### ## ## ##.

```

3.1. Config

##	##
entityAccessors	public final ## ## ## ## ## ## (###: false)
listAccessors	listProperty(int index) ### ## (###: false)
mapAccessors	mapProperty(Key key) ### ## ## (###: false)
createDefaultVariable	## ## ## (###: true)
defaultVariableName	## ## ##

```

### # ## ##.

```

```

### ## ## #####::

```

```

@Config(entityAccessors=true)
@Entity
public class User {
    //...
}

```

```
}
```

```
### ## ### #####::
```

```
@Config(listAccessors=true)
package com.mysema.query.domain.rel;

import com.mysema.query.annotations.Config;
```

```
## ### ### ##### ##### ###, ### APT ### ##### ##.
```

```
# 3.2. APT ##
```

##	##
querydsl.entityAccessors	##### ## ## ##
querydsl.listAccessors	### ### ### ## ## ##
querydsl.mapAccessors	# ## # ## ## ##
querydsl.prefix	## ### ## ### (###: Q)
querydsl.suffix	## ### ## ###
querydsl.packageSuffix	## ## ##### ## ###
querydsl.createDefaultVariable	## ## ### ##
querydsl.unknownAsEmbeddable	##### ### ##### embeddable# ##### ## (###: false)
querydsl.includedPackages	## ### ### ### ## (### ##) (default: all)
querydsl.includedClasses	## ### ### ### ## ## (### ##) (default: all)
querydsl.excludedPackages	## ##### ### ### ## (### ##) (default: none)
querydsl.excludedClasses	## ##### ### ### ## (### ##) (default: none)

```
### ### APT ##### ## ## ##.
```

```
<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.maven</groupId>
        <artifactId>apt-maven-plugin</artifactId>
        <version>1.0.9</version>
        <executions>
          <execution>
```

```
<goals>
  <goal>process</goal>
</goals>
<configuration>
  <outputDirectory>target/generated-sources/java</outputDirectory>
  <processor>com.mysema.query.apt.jpa.JPAAnnotationProcessor</processor>
  <options>
    <querydsl.entityAccessors>true</querydsl.entityAccessors>
  </options>
</configuration>
</execution>
</executions>
</plugin>
...
</plugins>
</build>
</project>
```

##

```
## ### ### ### ### ## ### ##### ##. ## String ### ## ### String ### ### ### ### ## Date/Time ##
# ##### ## ### ### ## ### ##### ### # ##. Joda time API# JDK(java.util.Date, Calendar ### ## ##)# ##
### ##### #####, ## API# ### ## # ### ##### ##.
```

##.

```
@Entity
public class MyEntity{

    @QueryType(PropertyType.SIMPLE)
    public String stringAsSimple;

    @QueryType(PropertyType.COMPARABLE)
    public String stringAsComparable;

    @QueryType(PropertyType.NONE)
    public String stringNotInQuerydsl;

}
```

```
PropertyType.NONE# ## ## ### ##### ### # ####. @Transient# @QueryTransient ##### ## ##
## ## ##### ## ## ## ##. PropertyType.NONE# ## Querydsl ## ##### ## ##### ##.
```

###(Delegate methods)

```
## ##### ## ##### #####, ##### ## ## ## ## QueryDelegate ##### ## ##### ##, ## ##### # ## #####
# ##### Querydsl ## ## ## ##.
```

###.

```
@QueryEntity
public static class User{
```

```
String name;

User manager;

}
```

```
@QueryDelegate(User.class)
public static BooleanPath isManagedBy(QUser user, User other){
    return user.manager.eq(other);
}
```

QUser ## ### ### ##### ## #.

```
public BooleanPath isManagedBy(QUser other) {
    return com.mysema.query.domain.DelegateTest.isManagedBy(this, other);
}
```

#. ### # ## ###.

```
public class QueryExtensions {

    @QueryDelegate(Date.class)
    public static BooleanExpression inPeriod(DatePath<Date> date, Pair<Date,Date> period){
        return date.goe(period.getFirst()).and(date.loe(period.getSecond()));
    }

    @QueryDelegate(Timestamp.class)
    public static BooleanExpression inDatePeriod(DateTimePath<Timestamp> timestamp, Pair<Date,Date> period){
        Timestamp first = new Timestamp(DateUtils.truncate(period.getFirst(), Calendar.DAY_OF_MONTH).getTime());
        Calendar second = Calendar.getInstance();
        second.setTime(DateUtils.truncate(period.getSecond(), Calendar.DAY_OF_MONTH));
        second.add(1, Calendar.DAY_OF_MONTH);
        return timestamp.goe(first).and(timestamp.lt(new Timestamp(second.getTimeInMillis())));
    }

}
```

#####, ## ##### ## ##### ## ##### #####.

```
public class QDate extends DatePath<java.sql.Date> {

    public QDate(BeanPath<? extends java.sql.Date> entity) {
        super(entity.getType(), entity.getMetadata());
    }

    public QDate(PathMetadata<?> metadata) {
        super(java.sql.Date.class, metadata);
    }

}
```

```

    public BooleanExpression inPeriod(com.mysema.commons.lang.Pair<java.sql.Date, java.sql.Date> period) {
        return QueryExtensions.inPeriod(this, period);
    }
}

public class QTimestamp extends DateTimePath<java.sql.Timestamp> {

    public QTimestamp(BeanPath<? extends java.sql.Timestamp> entity) {
        super(entity.getType(), entity.getMetadata());
    }

    public QTimestamp(PathMetadata<?> metadata) {
        super(java.sql.Timestamp.class, metadata);
    }

    public BooleanExpression inDatePeriod(com.mysema.commons.lang.Pair<java.sql.Date, java.sql.Date> period) {
        return QueryExtensions.inDatePeriod(this, period);
    }
}

```

##

@QueryEntities ##### ###, ##### ##### ## ### ##### Querydsl ## ### ##### ## #####. QueryEntities #
##, value ### ### ##### ## #####.

com.mysema.query.apt.QuerydslAnnotationProcessor# #####. ### ## ##
##.

```

<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.maven</groupId>
        <artifactId>apt-maven-plugin</artifactId>
        <version>1.0.9</version>
        <executions>
          <execution>
            <goals>
              <goal>process</goal>
            </goals>
            <configuration>
              <outputDirectory>target/generated-sources/java</outputDirectory>
              <processor>com.mysema.query.apt.QuerydslAnnotationProcessor</processor>
            </configuration>
          </execution>
        </executions>
      </plugin>
      ...
    </plugins>
  </build>
</project>

```

```
##### ## # ## # ## # ## ##(## ##, Scala# Groovy# ## ## JVM ## ## ## ##, ##### ## ## ## ## ## #
## ## #), GenericExporter ##### ##### ##### ##### ##### ##### ## ## ## ## ## # ##.

GenericExporter# ##### querydsl-codegen ### ### ##### ##. (# #####
com.mysema.querydsl:querydsl-codegen:${querydsl.version} ##)

### JPA# ## ##.
```

```
# ### DomainClass# ### # # ##### ### ## JPA ##### ## ##### ## target/generated-sources/java #####
# ## ### #####.
```

```
querydsl-maven-plugin# generic-export, jpa-export#jdo-export ## ## GenericExporter# #### # ##.
```

##.

##	####	##
File	targetFolder	### ### ### ## ##
boolean	scala	Scala ### ##### true (###: false)
String[]	packages	### ##### ### ###
boolean	handleFields	### ##### ### ## true (###: true)
boolean	handleMethods	getter# ##### ### ## true (###: true)
String	sourceEncoding	### ## ### ### ###
boolean	testClasspath	### ##### ##### true

JPA ##### ## ## ##.

```
<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.querydsl</groupId>
        <artifactId>querydsl-maven-plugin</artifactId>
        <version>${querydsl.version}</version>
        <executions>
          <execution>
            <phase>process-classes</phase>
            <goals>
              <goal>jpa-export</goal>
            </goals>
            <configuration>
              <targetFolder>target/generated-sources/java</targetFolder>
              <packages>
                <package>com.example.domain</package>
              </packages>
            </configuration>
          </execution>
        </executions>
      </plugin>
      ...
    </plugins>
  </build>
</project>
```

com.example.domain # # ## JPA ##### ## ## ## target/generated-sources/java #
##.

##, ## ## ## ## ## ## ## compile ## ## ##.

```
<execution>
  <goals>
    <goal>compile</goal>
  </goals>
  <configuration>
    <sourceFolder>target/generated-sources/scala</targetFolder>
  </configuration>
</execution>
```

compile ## ## ## ## ##.

3.4. ###

##	####	##
File	sourceFolder	### ## ## ##
String	sourceEncoding	### ## ##

##	####	##
String	source	##### -source ##
String	target	##### -target ##
boolean	testClasspath	### ##### ## ## true
Map	compilerOptions	#### ##

sourceFolder# ### ## ### #####.

Scala

Scala ### ###, ## ### #####.

```

<project>
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>com.mysema.querydsl</groupId>
        <artifactId>querydsl-maven-plugin</artifactId>
        <version>${querydsl.version}</version>
        <dependencies>
          <dependency>
            <groupId>com.mysema.querydsl</groupId>
            <artifactId>querydsl-scala</artifactId>
            <version>${querydsl.version}</version>
          </dependency>
          <dependency>
            <groupId>org.scala-lang</groupId>
            <artifactId>scala-library</artifactId>
            <version>${scala.version}</version>
          </dependency>
        </dependencies>
        <executions>
          <execution>
            <goals>
              <goal>jpa-export</goal>
            </goals>
            <configuration>
              <targetFolder>target/generated-sources/scala</targetFolder>
              <scala>true</scala>
              <packages>
                <package>com.example.domain</package>
              </packages>
            </configuration>
          </execution>
        </executions>
      </plugin>
      ...
    </plugins>
  </build>

```

```
</project>
```

3.4. ##

```
## ### # # ## ##, ### ### ## ## ## ##### ## ### ## ## # ##. ### ### ##### ##### ## ## ## getter ##### ##
# ### # ##.
```

```
## ### ### ### ### ### ### ##### ## ## ## ## ## # ## ## ##.
```

```
## ## ### APT# ### ### ### ##### ## ## ##### ## ##.
```

```
QCat cat = new QCat("cat");
for (String name : query.from(cat,cats)
    .where(cat.kittens.size().gt(0))
    .list(cat.name)){
    System.out.println(name);
}
```

```
###, ### Cal ##### ##### ## ##### ##### ###. $ ### ##### c.getKittens()# ##### ##### ##
c.kittens# #####.
```

```
Cat c = alias(Cat.class, "cat");
for (String name : query.from($(c),cats)
    .where($(c.getKittens()).size().gt(0))
    .list($(c.getName()))){
    System.out.println(name);
}
```

```
## ### ##### ## # import# ##### ##.
```

```
import static com.mysema.query.alias.Alias.$;
import static com.mysema.query.alias.Alias.alias;
```

```
## ## # ### #####. $ ### ### ##### size()# ##### ##.
```

```
Cat c = alias(Cat.class, "cat");
for (String name : query.from($(c),cats)
    .where($(c.getKittens()).size().gt(0))
    .list($(c.getName()))){
    System.out.println(name);
}
```

```
## ### ### ### final# ## ## ## ##### ## #####. ###, $ ### ### ## ## ## ##### final ### ### ## ## ## ##
# ### # ##. ### ##.
```

```
$(c.getMate().getName())
```

is transformed into `*c.mate.name*` internally, but

```
$(c.getMate().getName().toLowerCase())
```

`##### toLowerCase()### #####.`

`## ##### getters, size(), contains(Object), get(int)#####. ## ## ### ### ##### #####.`

4#.

4.1. ##### ##

Querydsl# ## ### ### List, Set, Collection, Map ##### ##### ### ## ### ##.

getter# ### ##, ### ## ### #####.

```
java.lang.RuntimeException: Caught exception for field com.mysema.query.jdoql.testdomain.Store#products
    at com.mysema.query.apt.Processor$2.visitType(Processor.java:117)
    at com.mysema.query.apt.Processor$2.visitType(Processor.java:80)
    at com.sun.tools.javac.code.Symbol$ClassSymbol.accept(Symbol.java:827)
    at com.mysema.query.apt.Processor.getClassModel(Processor.java:154)
    at com.mysema.query.apt.Processor.process(Processor.java:191)
    ...
Caused by: java.lang.IllegalArgumentException: Insufficient type arguments for List
    at com.mysema.query.apt.APTTypeModel.visitDeclared(APTTypeModel.java:112)
    at com.mysema.query.apt.APTTypeModel.visitDeclared(APTTypeModel.java:40)
    at com.sun.tools.javac.code.Type$ClassType.accept(Type.java:696)
    at com.mysema.query.apt.APTTypeModel.<init>(APTTypeModel.java:55)
    at com.mysema.query.apt.APTTypeModel.get(APTTypeModel.java:48)
    at com.mysema.query.apt.Processor$2.visitType(Processor.java:114)
    ... 35 more
```

##.

```
private Collection names; // WRONG

private Collection<String> names; // RIGHT

private Map employeesByName; // WRONG

private Map<String,Employee> employeesByName; // RIGHT
```

4.2. ##### ##### Querydsl Q###

Q### ## ### ## ##, ## ### ##### Q### ##### ##### ## ##.

##.

com.mysema.util.ClassPathUtils ##### ## ##.

```
ClassPathUtils.scanPackage(Thread.currentThread().getContextClassLoader(), packageToLoad);
```

packageToLoad# ## ##### ## ## ## ## ## ##.

4.3. JDK5

JDK 5# ##### ##### #, ### ## ##### ## # ##.

```
[INFO] -----  
[ERROR] BUILD FAILURE  
[INFO] -----  
[INFO] Compilation failure  
...  
class file has wrong version 50.0, should be 49.0
```

##6## ##### ## ## ## 50.0## ##5# 49.0##.

JDK 6.0 ## ##### ##### APT# ##### ## ##, Querydsl# JDK 6.0### ## ##.

JDK 5.0## Querydsl# ##### ## ## Querydsl ### ## ##### ##.