Есть довольно много способов разбить на страницы.

**HQL и setFirstResult, API setMaxResults**

Session session = sessionFactory.openSession();

Query query = session.createQuery("From Foo");

query.setFirstResult(0);

query.setMaxResults(10);

List<Foo> fooList = query.list();

//Total count

String countQ = "Select count (f.id) from Foo f";

Query countQuery = session.createQuery(countQ);

Long countResults = (Long) countQuery.uniqueResult();

//Last Page

int pageSize = 10;

int lastPageNumber = (int) ((countResult / pageSize) + 1);

**HQL и API ScrollableResults**

String hql = "FROM Foo f order by f.name";

Query query = session.createQuery(hql);

int pageSize = 10;

ScrollableResults resultScroll = query.scroll(ScrollMode.FORWARD\_ONLY);

resultScroll.first();

resultScroll.scroll(0);

List<Foo> fooPage = Lists.newArrayList();

int i = 0;

while (pageSize > i++) {

fooPage.add((Foo) resultScroll.get(0));

if (!resultScroll.next())

break;

}

//Total count

resultScroll.last();

int totalResults = resultScroll.getRowNumber() + 1;

**Просто Criteria API**

Criteria criteria = session.createCriteria(Foo.class);

criteria.setFirstResult(0);

criteria.setMaxResults(pageSize);

List<Foo> firstPage = criteria.list();

//Total count

Criteria criteriaCount = session.createCriteria(Foo.class);

criteriaCount.setProjection(Projections.rowCount());

Long count = (Long) criteriaCount.uniqueResult();

[baeldung](http://www.baeldung.com/hibernate-pagination) перечисляет их все с примерами.

Anand Rockzz 23 авг. 2015, в 08:39

Поделиться

* 0

Ваш пример использования API ScrollableResults будет читать все в память. [Смотрите здесь](http://stackoverflow.com/questions/2826319/using-hibernates-scrollableresults-to-slowly-read-90-million-records) Вы можете заставить это работать, как объяснено там @Sean S., но это определенно не предлагает простого решения.

thisismydesign 14 окт. 2016, в 11:25

* 0

СУБД зависит и **не** все СУБД имеют эту проблему. Полезно знать, но это не был вопрос / проблема оригинального плаката

Чтобы ограничить размер ResultSet базового запроса, интерфейс Query JPA предоставляет [метод setMaxResults](https://docs.oracle.com/javaee/7/api/javax/persistence/Query.html#setMaxResults-int-%23setMaxResults-int-).

Перемещение по следующей странице требует размещения результирующего набора там, где заканчивалась последняя страница. Для этого интерфейс JPA Query предоставляет [метод setFirstResult](https://docs.oracle.com/javaee/7/api/javax/persistence/Query.html#setFirstResult-int-%23setFirstResult-int-).

**JPQL**

List<Post> posts = entityManager

.createQuery(

"select p " +

"from Post p " +

"order by p.createdOn ")

.setFirstResult(10)

.setMaxResults(10)

.getResultList();

**DTO проекционные запросы**

Нумерация запросов JPA не ограничивается запросами сущностей, которые возвращают только сущности. Вы также можете использовать его для прогнозов DTO.

List<PostCommentSummary> summaries = entityManager

.createQuery(

"select new " +

" com.vladmihalcea.book.hpjp.hibernate.fetching.PostCommentSummary( " +

" p.id, p.title, c.review " +

" ) " +

"from PostComment c " +

"join c.post p " +

"order by c.createdOn")

.setMaxResults(10)

.getResultList();

**Собственные запросы SQL**

Разбиение на страницы запроса JPA не ограничивается запросами сущностей, такими как JPQL или Criteria API. Вы также можете использовать его для собственных запросов SQL.

List<Tuple> posts = entityManager

.createNativeQuery(

"select p.id as id, p.title as title " +

"from post p " +

"order by p.created\_on", Tuple.class)

.setFirstResult(10)

.setMaxResults(10)

.getResultList();

**ПРИСОЕДИНЯЙТЕСЬ к FETCH и нумерации страниц**

Однако, если мы попытаемся использовать предложение JOIN FETCH в запросе сущности, в то же время используя пагинацию JPA:

List<Post> posts = entityManager.createQuery(

"select p " +

"from Post p " +

"left join fetch p.comments " +

"order by p.createdOn", Post.class)

.setMaxResults(10)

.getResultList();

Hibernate выдаст следующее предупреждающее сообщение:

HHH000104: firstResult/maxResults specified with collection fetch; applying in memory!

И в выполненном SQL-запросе не будет предложения о разбиении на страницы:

SELECT p.id AS id1\_0\_0\_,

c.id AS id1\_1\_1\_,

p.created\_on AS created\_2\_0\_0\_,

p.title AS title3\_0\_0\_,

c.created\_on AS created\_2\_1\_1\_,

c.post\_id AS post\_id4\_1\_1\_,

c.review AS review3\_1\_1\_,

c.post\_id AS post\_id4\_1\_0\_\_,

c.id AS id1\_1\_0\_\_

FROM post p

LEFT OUTER JOIN post\_comment c ON p.id=c.post\_id

ORDER BY p.created\_on

Это связано с тем, что Hibernate хочет полностью извлекать сущности вместе со своими коллекциями, как указано предложением JOIN FETCH, в то время как разбиение на страницы на уровне SQL может обрезать ResultSet возможно, оставляя родительскую сущность Post с меньшим количеством элементов в коллекции comments.

Проблема с предупреждением HHH000104 заключается в том, что Hibernate будет HHH000104 продукт сущностей Post и PostComment, и из-за размера набора результатов время ответа на запрос будет значительным.

Чтобы обойти это ограничение, вы должны использовать запрос [оконной функции](https://vladmihalcea.com/why-you-should-definitely-learn-sql-window-functions/rel=):

@NamedNativeQuery(

name = "PostWithCommentByRank",

query =

"SELECT \* " +

"FROM ( " +

" SELECT \*, dense\_rank() OVER (ORDER BY \"p.created\_on\", \"p.id\") rank " +

" FROM ( " +

" SELECT p.id AS \"p.id\", " +

" p.created\_on AS \"p.created\_on\", " +

" p.title AS \"p.title\", " +

" pc.id as \"pc.id\", " +

" pc.created\_on AS \"pc.created\_on\", " +

" pc.review AS \"pc.review\", " +

" pc.post\_id AS \"pc.post\_id\" " +

" FROM post p " +

" LEFT JOIN post\_comment pc ON p.id = pc.post\_id " +

" WHERE p.title LIKE :titlePattern " +

" ORDER BY p.created\_on " +

" ) p\_pc " +

") p\_pc\_r " +

"WHERE p\_pc\_r.rank <= :rank ",

resultSetMapping = "PostWithCommentByRankMapping"

)

@SqlResultSetMapping(

name = "PostWithCommentByRankMapping",

entities = {

@EntityResult(

entityClass = Post.class,

fields = {

@FieldResult(name = "id", column = "p.id"),

@FieldResult(name = "createdOn", column = "p.created\_on"),

@FieldResult(name = "title", column = "p.title"),

}

),

@EntityResult(

entityClass = PostComment.class,

fields = {

@FieldResult(name = "id", column = "pc.id"),

@FieldResult(name = "createdOn", column = "pc.created\_on"),

@FieldResult(name = "review", column = "pc.review"),

@FieldResult(name = "post", column = "pc.post\_id"),

}

)

}

)