Coupang Catalog Platform & Quality
debop@coupang.com

Requery

Agenda

- Requery Overview
  - Why Requery
  - Requery Build process
  - Define Mapping
  - Usage EntityDataStore
  - EntityDataStore for Kotlin (Coroutines)
- Introduction of Spring Data Requery

Requery Overview

- ORM Library for Java, Kotlin, Android
- No Reflection (vs Hibernate proxy)
- Typed Query language (vs Hibernate Criteria)
- Upsert/Partial objects refresh
- Compile time entity/query validation (vs Hibernate)
- Entity is stateless (vs Hibernate stateful)
- Thread 에 제한 받지 않음 (JPA EntityManager)
- Support RxJava, Async Operations, Java 8

Why Requery

- Provide benefit of ORM
  - Entity Mapping
  - Schema Generation
  - Compile time error detecting
- Performance
  - When bulk job, max 100x than JPA
  - REST API - 2~10x throughput
  - Support Upsert, Lazy loading …

Requery Build Process – Java


```
buildscript {

    repositories {
        jcenter()
        maven { url "https://plugins.gradle.org/m2/" }
    }

    dependencies {
        // for Java apt
        classpath "net.ltgt.gradle:gradle-apt-plugin:0.15"
    }
}

// lombok을 gradle 에서 사용하기 위한 plugin
plugins {
    id 'io.franzbecker.gradle-lombok' version '1.14'
}
```

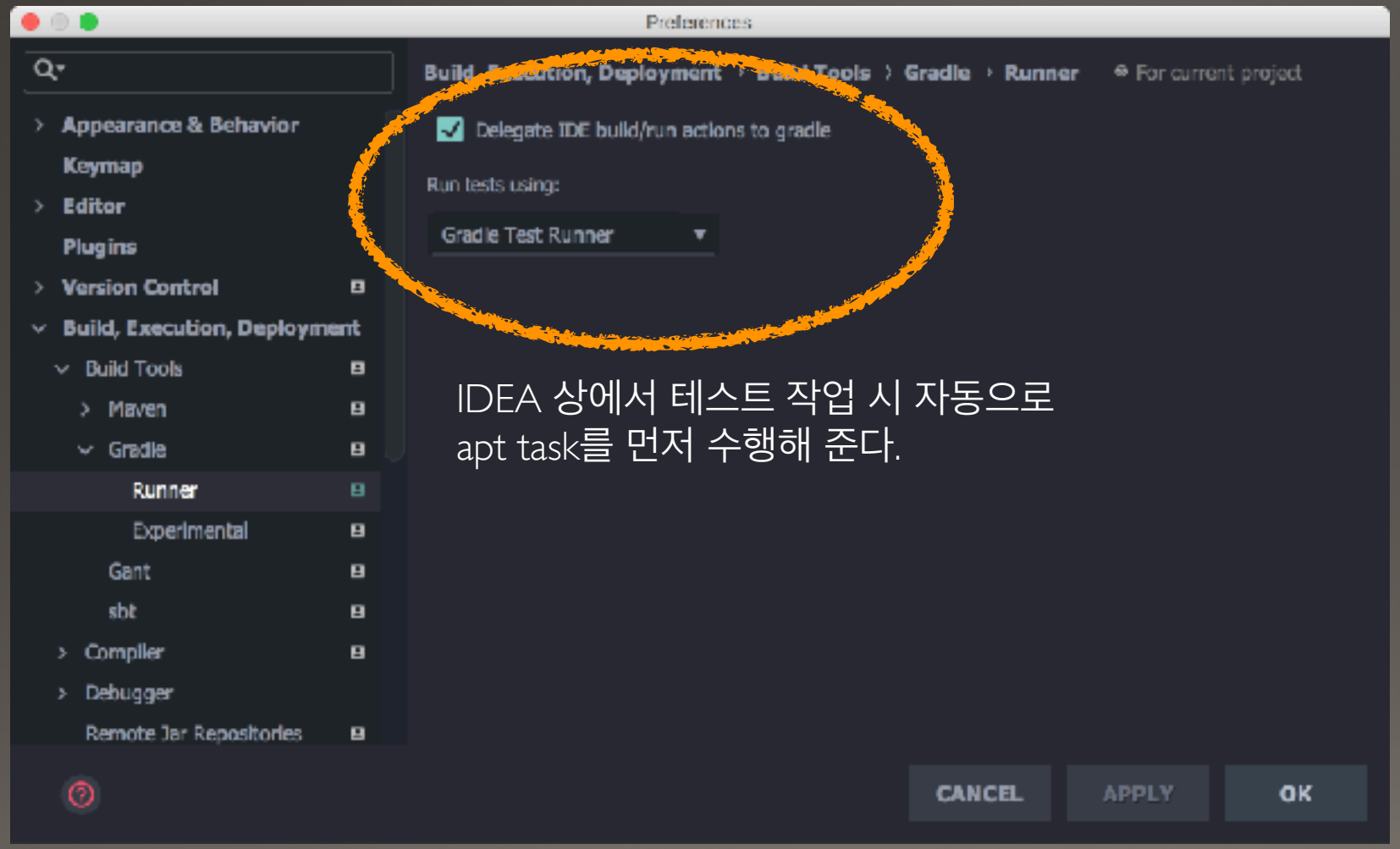
```
// lombok을 gradle 에서 사용하기 위해 annotation process를 설정해주어야 합니다.
compileOnly "org.projectlombok:lombok"
annotationProcessor "org.projectlombok:lombok"
testAnnotationProcessor "org.projectlombok:lombok"

annotationProcessor "io.requery:requery-processor"
testAnnotationProcessor "io.requery:requery-processor"
```

Requery Build Process – Kotlin


```
// for kotlin entity
kapt "io.requery:requery-processor"
kaptTest "io.requery:requery-processor"
```

IntelliJ IDEA Settings



IDEA 상에서 테스트 작업 시 자동으로
apt task를 먼저 수행해 준다.

Define Entity – Java

```java
@Getter
@Entity(name = "BasicUser", copyable = true)
@Table(name = "basic_user")
public abstract class AbstractBasicUser extends AuditableLongEntity {

    @Key
    @Generated
    protected Long id;

    protected String name;
    protected String email;
    protected LocalDate birthday;
    protected Integer age;

    @ForeignKey
    @OneToOne
    protected AbstractBasicLocation address;

    @ManyToMany(mappedBy = "members")
    protected Set<AbstractBasicGroup> groups;

    @Column(unique = true)
    protected UUID uuid;

    @Override
    public int hashCode() {
        return Objects.hash(name, email, birthday);
    }

    @Transient
    @Override
    protected @NotNull ToStringBuilder buildStringHelper() {
        return super.buildStringHelper()
            .add("name", name)
            .add("email", email)
            .add("birthday", birthday);
    }

    private static final long serialVersionUID = -2693264826800934057L;
}
```

Define Entity – Kotlin

```kotlin
@Entity(model = "functional")
interface Person: Persistable {

    @get:Key
    @get:Generated
    val id: Long

    @get:Index(value = ["idx_person_name_email"])
    var name: String

    @get:Index(value = ["idx_person_name_email", "idx_person_email"])
    var email: String

    var birthday: LocalDate

    @get:Column(value = "'empty'")
    var description: String?

    @get:Nullable
    var age: Int?

    @get:ForeignKey
    @get:OneToOne(mappedBy = "person", cascade = [CascadeAction.DELETE, CascadeAction.SAVE])
    var address: Address?

    @get:OneToMany(mappedBy = "owner", cascade = [CascadeAction.DELETE, CascadeAction.SAVE])
    val phoneNumbers: MutableSet<Phone>

    @get:OneToMany
    val phoneNumberList: MutableList<Phone>

    @get:ManyToMany(mappedBy = "members")
    val groups: MutableResult<Group>

    @get:ManyToMany(mappedBy = "owners")
    val ownedGroups: MutableResult<Group>

    @get:ManyToMany(mappedBy = "id")
    @get:JunctionTable
    val friends: MutableSet<Person>

    @get:Lazy
    var about: String?

    @get:Column(unique = true)
    var uuid: UUID

    var homepage: URL
    var picture: String
}
```

EntityDataStore<Object>

- findByKey
- select / insert / update / **upsert** / delete
- where / eq, lte, lt, gt, gte, like, in, not …
- groupBy / having / limit / offset
- support SQL Functions
  - count, sum, avg, upper, lower …
- raw query

```kotlin
@Test
fun `insert user`() {
    val user = RandomData.randomUser()
    withDb(Models.DEFAULT) {
        insert(user)
        assertThat(user.id).isGreaterThan(0)

        val loaded = select(User::class) where (User::id eq user.id) limit 10
        assertThat(loaded.get().first()).isEqualTo(user)
    }
}
```

```kotlin
val result = select(Location::class)
    .join(User::class).on(User::location eq Location::id)
    .where(User::id eq user.id)
    .orderBy(Location::city.desc())
    .get()
```

```kotlin
val rowCount = update(UserEntity::class)
    .set(UserEntity.ABOUT, "nothing")
    .set(UserEntity.AGE, 50)
    .where(UserEntity.AGE eq 100)
    .get()
    .value()
```

```kotlin
val count = insert(PersonEntity::class, PersonEntity.NAME, PersonEntity.DESCRIPTION)
    .query(select(GroupEntity.NAME, GroupEntity.DESCRIPTION))
    .get()
    .first()
    .count()
```

```kotlin
val result = raw(User::class, "SELECT * FROM Users")
```

CoroutineEntityDataStore

```
val store = CoroutineEntityStore(this)

runBlocking {
    val users = store.insert(RandomData.randomUsers(10))

    users.await().forEach { user ->
        assertThat(user.id).isGreaterThan(0)
    }

    store
        .count(UserEntity::class)
        .get()
        .toDeferred()
        .await()
        .let {
            assertThat(it).isEqualTo(10)
        }
}
```

```
with(coroutineTemplate) {
    val user = randomUser()

    // can replace with `withContext { }`
    async { insert(user) }.await()
    assertThat(user.id).isNotNull()

    val group = RandomData.randomGroup()
    group.members.add(user)

    async { insert(group) }.await()

    assertThat(user.groups).hasSize(1)
    assertThat(group.members).hasSize(1)
}
```

spring-data-requery

- RequeryOperations
  - Wrap EntityDataStore
- RequeryTransactionManager for PlatformTransactionManager
-

spring-data-requery

- Repository built in SQL
- ByPropertyName Auto generation methods
- @Query for raw SQL Query
- Query By Example
- Not Supported
  - Association Path (not specified join method)
  - Named parameter in @Query (just use `?`)

Setup spring-data-requery

```java
@Configuration
@EnableTransactionManagement
public class RequeryTestConfiguration extends AbstractRequeryConfiguration {
    @Override
    @Bean
    public EntityModel getEntityModel() {
        return Models.DEFAULT;
    }
    @Override
    public TableCreationMode getTableCreationMode() {
        return TableCreationMode.CREATE_NOT_EXISTS;
    }
    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.H2)
            .build();
    }
}
```

Provided Beans

```java
@Bean
public io.requery.sql.Configuration requeryConfiguration() {
    return new ConfigurationBuilder(dataSource, getEntityModel())
        // .useDefaultLogging()
        .setEntityCache(new EmptyEntityCache())
        .setStatementCacheSize(1024)
        .setBatchUpdateSize(100)
        .addStatementListener(new LogbackListener())
        .build();
}

@Bean
public EntityDataStore<Object> entityDataStore() {
    log.info("Create EntityDataStore instance.");
    return new EntityDataStore<>(requeryConfiguration());
}

@Bean
public RequeryOperations requeryOperations() {
    log.info("Create RequeryTemplate instance.");
    return new RequeryTemplate(entityDataStore(), requeryMappingContext());
}
```

EntityCache 설정 Tip :
개발 시에는 EmptyEntityCache,
운영 시에는 Cache2kEntityCache 사용

Use @Query in Repository

```
interface DeclaredQueryRepository extends RequeryRepository<BasicUser, Long> {

    @Query("select * from basic_user u where u.email = ?")
    BasicUser findByAnnotatedQuery(String email);

    @Query("select * from basic_user u where u.email like ?")
    List<BasicUser> findAllByEmailMatches(String email);

    @Query("select * from basic_user u limit ?")
    List<BasicUser> findWithLimits(int limit);

    @Query("select * from basic_user u where u.name=? and u.email=? limit 1")
    BasicUser findAllBy(String name, String email);

    @Query("select u.id, u.name from basic_user u where u.email=?")
    List<Tuple> findAllIds(String email);

    @Query("select * from basic_user u where u.birthday = ?")
    List<BasicUser> findByBirthday(LocalDate birthday);
}
```

Query By Example

```
BasicUser user = RandomData.randomUser();
user.setName("example");
requeryTemplate.insert(user);

BasicUser exampleUser = new BasicUser();
exampleUser.setName("EXA");

ExampleMatcher matcher = matching()
    .withMatcher("name", startsWith().ignoreCase())
    .withIgnoreNullValues();

Example<BasicUser> example = Example.of(exampleUser, matcher);

Return<? extends Result<BasicUser>> query = buildQueryByExample(example);

BasicUser foundUser = query.get().firstOrNull();
assertThat(foundUser).isNotNull().isEqualTo(user);
```

Query by Property – Not Yet

```
List<User> findByFirstnameOrLastname(@Param("lastname") String lastname, @Param("firstname") String firstname);

List<User> findByLastnameLikeOrderByFirstnameDesc(String lastname);

List<User> findByLastnameNotLike(String lastname);

List<User> findByLastnameNot(String lastname);

List<User> findByManagerLastname(String name);          Note: Association Path is not supported

List<User> findByColleaguesLastname(String lastname);   Note: Association Path is not supported

List<User> findByLastnameNotNull();

@Query("select u.lastname from SD_User u group by u.lastname")
Page<String> findByLastnameGrouped(Pageable pageable);

long countByLastname(String lastname);

int countUsersByFirstname(String firstname);

boolean existsByLastname(String lastname);
```

Resources

- requery.io
- kotlinx-data-requery in coupang gitlab
- spring-data-requery in coupang gitlab

Q&A

Thank you!