7±2 things you didn't know about Exposed

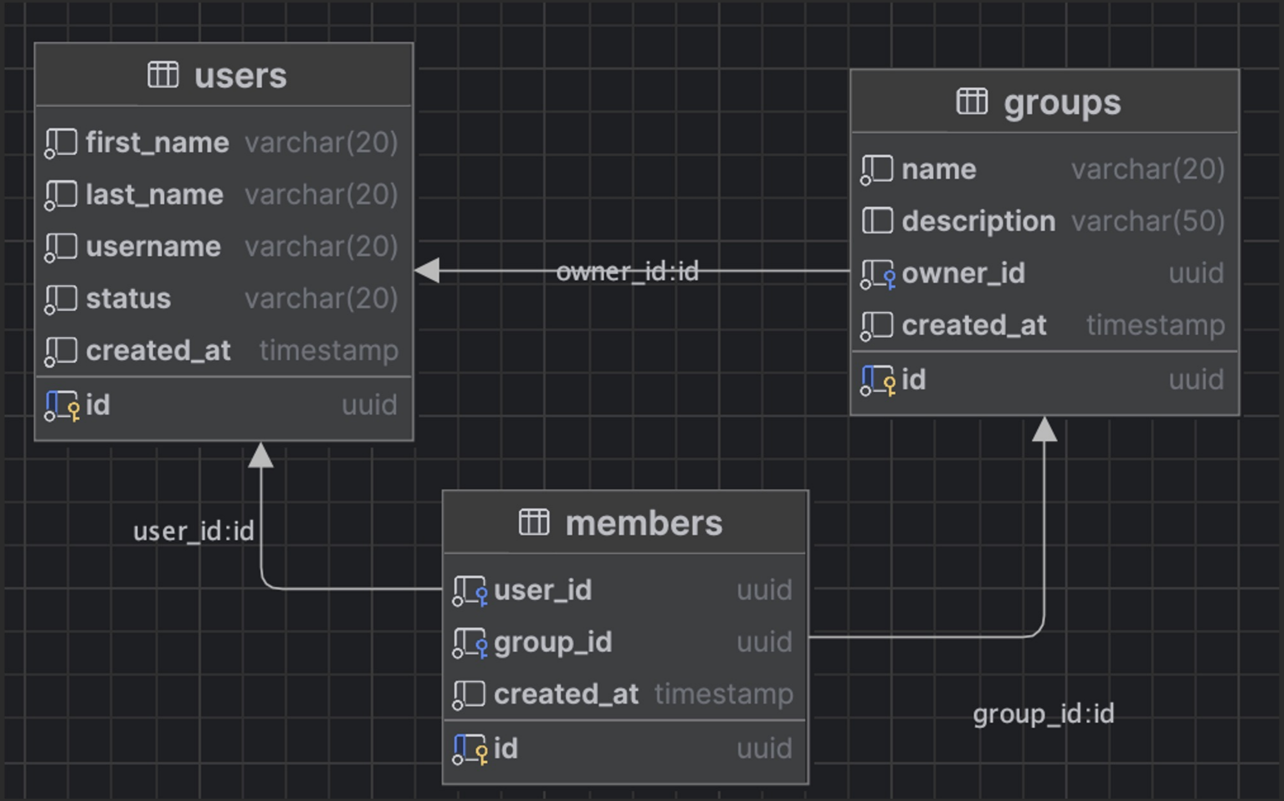
Alexey Soshin

@alexey_soshin

KotlinConf'23

Amsterdam

The data model



users

first_name	varchar(20)
last_name	varchar(20)
username	varchar(20)
status	varchar(20)
created_at	timestamp
id	uuid

groups

name	varchar(20)
description	varchar(50)
owner_id	uuid
created_at	timestamp
id	uuid

members

user_id	uuid
group_id	uuid
created_at	timestamp
id	uuid

Selecting using criteria with an annotation

```java
@Query("select u from User u where u.firstname = :firstname")
List<User> findByFirstname(String firstname);
```

Selecting using criteria with an annotation

```
@Query("select u from User u where u.firstname = :firstname")
List<User> findByFirstname(String firstname);
```

Deleting using conventions

```
Long removeByLastname(String lastname);
```

Selecting using criteria: SQL DSL

```
Users
    .select { Users.firstName eq firstName }
    .toList()
```

Deleting: SQL DSL

```
val count: Int = Users
    .deleteWhere { Users.lastName eq lastName }
```

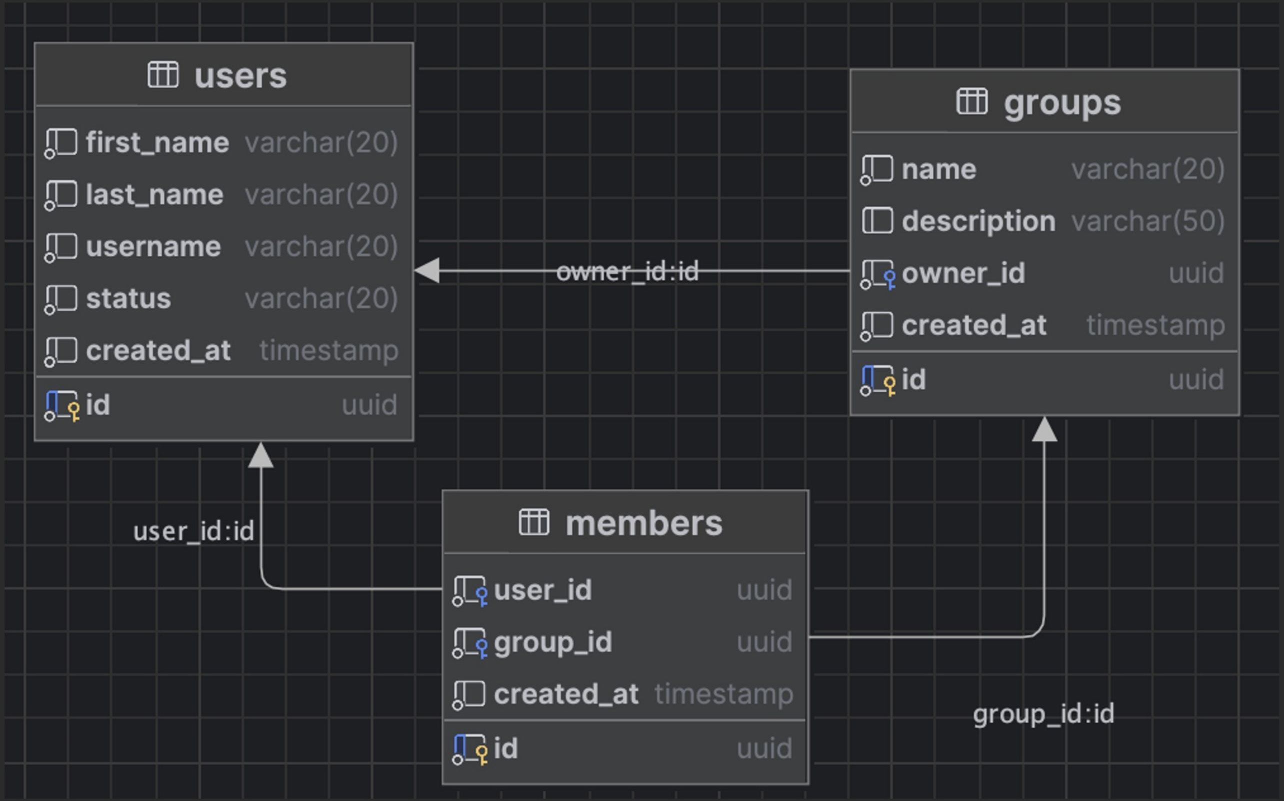
Selecting using criteria: DAO API

```
val users = User.find {
    Users.firstName eq firstName
}.toList()
```

Deleting: DAO API

```
users.forEach {
    it.delete()
}
```

Many–to-many relationship



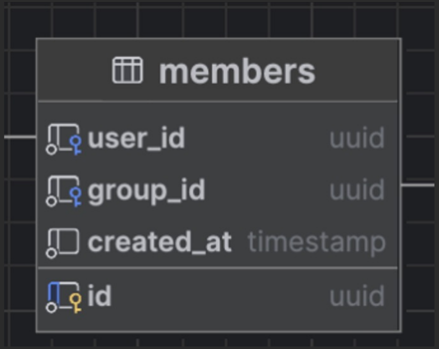
Many–to-many relationship

```
val group = Group.findById(groupId)
val groupsUsers: List<User> =
    group.users.toList()


val user = User.findById(userId)
val usersGroups: List<Group> =
    user.groups.toList()
```

Many–to-many relationship

```
object Members : UUIDTable() {
    val user = reference("user_id", Users, onDelete =
ReferenceOption.CASCADE)
    val group = reference("group_id", Groups, onDelete =
ReferenceOption.CASCADE)
    val createdAt = datetime("created_at")
}
```



```
members
user_id      uuid
group_id     uuid
created_at   timestamp
id           uuid
```

Many–to-many relationship

```
class User(id: EntityID<UUID>) : UUIDEntity(id) {
    companion object : UUIDEntityClass<User>(Users)

    var username by Users.username
    val groups by Group.via(Members.user, Members.group)
}
```

Many–to-many relationship

```
class Group(id: EntityID<UUID>) : UUIDEntity(id) {
    companion object : UUIDEntityClass<Group>(Groups)

    var name by Groups.name
    var description by Groups.description
    var owner by User referencedOn Groups.owner
    val members by User via Members
}
```

Many–to-many relationship

```
class Group(id: EntityID<UUID>) : UUIDEntity(id) {
    companion object : UUIDEntityClass<Group>(Groups)

    var name by Groups.name
    var description by Groups.description
    var owner by User referencedOn Groups.owner
    val members by User via Members
}
```

Date support

```kotlin
object Users : UUIDTable() {
    val firstName = varchar("first_name", 20)
    val lastName = varchar("last_name", 20)
    val username = varchar("username", 20)
    val status = enumeration<UserStatus>("status")
    val createdAt = datetime("created_at")
        .defaultExpression(CurrentDateTime)
}
```

Date support

```
// JDK 7, legacy
"org.jetbrains.exposed:exposed-jodatime"
```

Date support

```
// JDK 7, legacy
"org.jetbrains.exposed:exposed-jodatime"
// JDK 8+
"org.jetbrains.exposed:exposed-java-time"
```

Date support

```
// JDK 7, legacy
"org.jetbrains.exposed:exposed-jodatime"
// JDK 8+
"org.jetbrains.exposed:exposed-java-time"
// Kotlin
"org.jetbrains.exposed:exposed-kotlin-datetime"
```

Logging

```
transaction {


    Users.select { Users.firstName eq firstName }
        .toList()
}
```

Logging

```
transaction {
    addLogger(StdOutSqlLogger)

    Users.select { Users.firstName eq firstName }
        .toList()
}
```

Logging

```
transaction {
    addLogger(StdOutSqlLogger)

    Users.select { Users.firstName eq firstName }
        .toList()
}
```

```
SQL: SELECT users.id, users.first_name,
users.last_name FROM users WHERE users.first_name =
'Alexey'
```

Logging

```
transaction {
    if (firstName != null) {
        Users.select { Users.firstName eq firstName }
            .toList()
    } else {
        Users.select { Users.lastName eq lastName }
            .toList()
    }
}
```

Logging

```
transaction {
    val sql = Users
        .select { Users.firstName eq firstName }
        .prepareSQL(this)

    println(sql)
}
```

Logging

```
transaction {
    val sql = Users
        .select { Users.firstName eq firstName }
        .prepareSQL(this)

    println(sql)
}
```

```
SELECT users.id, …
FROM users
WHERE users.last_name = ?
```

Coroutines support

```
suspend fun doSomeIO() {
    delay(10L)
}
```

Coroutines support

```
suspend fun doSomeIO() {
    delay(10L)
}



transaction {
    doSomeIO()
}
```

Coroutines support

```
newSuspendedTransaction {
    doSomeIO()
}
```

👍

Coroutines support

```
newSuspendedTransaction {
    doSomeIO()
}
```

```
newSuspendedTransaction {
    // Nesting suspended transactions
    suspendedTransaction {
        doSomeIO()
    }
    doSomeIO() // Failure here will rollback everything
}
```

Coroutines support

```
newSuspendedTransaction {
    // Nesting suspended transactions
    newSuspendedTransaction {
        doSomeIO()
    }
    doSomeIO() // Failure here won’t rollback everything!
}
```

Coroutines support

```
newSuspendedTransaction {
    User.new {
        username = "jstiles"; firstName = "John"; lastName = "Stiles"
    }
    suspendedTransaction {
        User.new {
            username = "mmajor"; firstName = "Mary"; lastName = "Major"
        }
    }
    rollback()
    println(User.all().toList())

}
```

Coroutines support

```
newSuspendedTransaction {
    User.new {
        username = "jstiles"; firstName = "John"; lastName = "Stiles"
    }
    suspendedTransaction {
        User.new {
            username = "mmajor"; firstName = "Mary"; lastName = "Major"
        }
    }
    rollback()
    println(User.all().toList())
    // []
}
```

Coroutines support

```
newSuspendedTransaction {
    User.new {
        username = "jstiles"; firstName = "John"; lastName = "Stiles"
    }
    suspendedTransaction {
        User.new {
            username = "mmajor"; firstName = "Mary"; lastName = "Major"
        }
        rollback()
    }
    println(User.all().toList())

}
```

Coroutines support

```
newSuspendedTransaction {
    User.new {
        username = "jstiles"; firstName = "John"; lastName = "Stiles"
    }
    suspendedTransaction {
        User.new {
            username = "mmajor"; firstName = "Mary"; lastName = "Major"
        }
        rollback()
    }
    println(User.all().toList())
    // []
}
```

Coroutines support

```
newSuspendedTransaction {
   User.new {
       username = "jstiles"; firstName = "John"; lastName = "Stiles"
   }
   newSuspendedTransaction {
       User.new {
           username = "mmajor"; firstName = "Mary"; lastName = "Major"
       }
       rollback()
   }
   println(User.all().toList())

}
```

Coroutines support

```
newSuspendedTransaction {
   User.new {
       username = "jstiles"; firstName = "John"; lastName = "Stiles"
   }
   newSuspendedTransaction {
       User.new {
           username = "mmajor"; firstName = "Mary"; lastName = "Major"
       }
       rollback()
   }
   println(User.all().toList())
   // [User(username='jstiles', firstName='John', lastName='Stiles')]
}
```

Coroutines support

```
newSuspendedTransaction {
    User.new {
        username = "jstiles"; firstName = "John"; lastName = "Stiles"
    }
    newSuspendedTransaction {
        User.new {
            username = "mmajor"; firstName = "Mary"; lastName = "Major"
        }
    }
    rollback()
    println(User.all().toList())

}
```

Coroutines support

```
newSuspendedTransaction {
   User.new {
       username = "jstiles"; firstName = "John"; lastName = "Stiles"
   }
   newSuspendedTransaction {
       User.new {
           username = "mmajor"; firstName = "Mary"; lastName = "Major"
       }
   }
   rollback()
   println(User.all().toList())
   // [User(username='mmajor', firstName='Mary', lastName='Major')]
}
```

Null expression

```
SELECT COUNT(
   CASE  WHEN users.status = 'BANNED' THEN NULL
   ELSE 1 END)
FROM users
```

Null expression

```
Users.slice(
    Count(
        case()
            .When(status eq UserStatus.BANNED, null)
            .Else(1)
    )
).selectAll().toList()
```

```
SELECT COUNT(
    CASE  WHEN users.status = 'BANNED' THEN NULL
    ELSE 1 END)
FROM users
```

Null expression

```
Users.slice(
    Count(
         case()
              .When(status eq UserStatus.BANNED, null)
              .Else(intLiteral(1))
    )
).selectAll().toList()
```

Null expression

```
Users.slice(
    Count(
        case()
            .When(status eq UserStatus.BANNED,
                Op.nullOp<Int>())
            .Else(intLiteral(1))
    )
).selectAll().toList()
```

Null expression

```
Users.slice(
   Count(
       case()
           .When(status neq UserStatus.BANNED,
                 intLiteral(1))
           .Else(Op.nullOp())
   )
).selectAll().toList()
```

```
SELECT
   COUNT(CASE  WHEN users.status <> 'BANNED' THEN 1
   ELSE NULL END)
FROM users
```

From SQL DSL to DAO Entities

```
val query: Query = Groups
    .innerJoin(Users)
    .innerJoin(Members)
    .slice(Groups.columns)
    .selectAll()
    .withDistinct()


fun doStuff(groupEntity: Group) {

}
```

🤔

From SQL DSL to DAO Entities

```
val groups: List<Group> = query.map { rs ->
    val group = Group(rs[Groups.id])
    group.name = rs[Groups.name]
    group.owner = User(rs[Groups.owner])
    group.description = rs[Groups.description]

    group
}
```

From SQL DSL to DAO Entities

```kotlin
val groups: List<Group> = query.map { rs ->
    val group = Group(rs[Groups.id])
    group.name = rs[Groups.name]
    group.owner = User(rs[Groups.owner])
    group.description = rs[Groups.description]

    group
}
```

😱

Property klass should be initialized before get.

From SQL DSL to DAO Entities

```kotlin
val query: Query = Groups
    .innerJoin(Users)
    .innerJoin(Members)
    .slice(Groups.columns)
    .selectAll()
    .withDistinct()

val groups: List<Group> = query.map { rs ->
    Group.wrapRow(rs)
}
```

From SQL DSL to DAO Entities

```
val query: Query = Groups
    .innerJoin(Users)
    .innerJoin(Members)
    .slice(Groups.columns)
    .selectAll()
    .withDistinct()

val groups: List<Group> =
    Group.wrapRows(query).toList()
```

Listening to changes

```
User.new {
    firstName = "Alexey"
    lastName = "Soshin"
    username = "asoshin"
}

val u = User.find { Users.username eq "asoshin" }.first()

u.status = UserStatus.ACTIVE
u.flush()

u.delete()
```

Listening to changes

```
EntityHook.subscribe { change ->
    println("${change.entityClass} with id
               ${change.entityId} was
               ${change.changeType}")
}
```

Listening to changes

```
EntityHook.subscribe { change ->
    println("${change.entityClass} with id
            ${change.entityId} was
            ${change.changeType}")
}
```

```
…User with id 28f9… was Created
…User with id 28f9… was Updated
…User with id 28f9… was Removed
```

Listening to changes

```
val action = EntityHook.subscribe { change ->
    println("${change.entityClass} with id
${change.entityId} was ${change.changeType}")
}

…
EntityHook.unsubscribe(action)
u.delete()
```

Listening to changes

```
val action = EntityHook.subscribe { change ->
    println("${change.entityClass} with id 
${change.entityId} was ${change.changeType}")
}

…
EntityHook.unsubscribe(action)
u.delete()
```

…User with id 28f9… was Created
…User with id 28f9… was Updated

Custom DB enums

```
enum class UserStatus {
    BANNED,    // baned
    ACTIVE,    // active
    DISABLED, // not active
    CREATED    // new
}
```

Custom DB enums

```kotlin
object Users : UUIDTable() {
    …
    val status =
        enumerationByName<UserStatus>("status", 20)
}
```

Custom DB enums

```
enum class UserStatus {
    BANNED,    // BANNED
    ACTIVE,    // ACTIVE
    DISABLED,  // DISABLED
    CREATED    // CREATED
}
```

Custom DB enums

```
val status =
      enumerationByName<UserStatus>("status", 20)

->

val status = customEnumeration(
   "status", "varchar(20)",
   fromDB, toDB
)
```

Custom DB enums

```
val status =
      enumerationByName<UserStatus>("status", 20)

->

val status = customEnumeration(
   "status", "varchar(20)",
   fromDB, toDB
)
```

Custom DB enums

```
val fromDB = { dbValue: Any ->
    when (dbValue) {
        "baned" -> UserStatus.BANNED
        "new" -> UserStatus.CREATED
        "not active" -> UserStatus.DISABLED
        "active" -> UserStatus.ACTIVE
        else -> throw RuntimeException("Unknown user
status: $dbValue")
    }
}
```

Custom DB enums

```
val toDB = { enumValue: UserStatus ->
    when (enumValue) {
        UserStatus.BANNED -> "baned"
        UserStatus.ACTIVE -> "active"
        UserStatus.DISABLED -> "not active"
        UserStatus.CREATED -> "new"
    }
}
```

Breaking the rules

```
exec("select * from users") { rs ->
    while (rs.next()) {
        // Not zero based!
        println(rs.getString(1))
    }
}
```

Summary

1. Support for many-to-many relationship
2. Choose what date library to work with
3. Log a single query
4. Coroutine support
5. Expression that represents SQL NULL

Summary

6. Easy mapping from SQL DSL to DAO Entities
7. DAO Event listeners
8. Custom DB enums
9. Arbitrary statements

Questions and contributions

Questions:

stackoverflow.com/questions/tagged/kotlin-exposed

kotlinlang.slack.com => #exposed

Bugs:

github.com/JetBrains/Exposed/issues

PRs are welcome!

github.com/JetBrains/Exposed/pulls

Thank you, and don't forget to vote

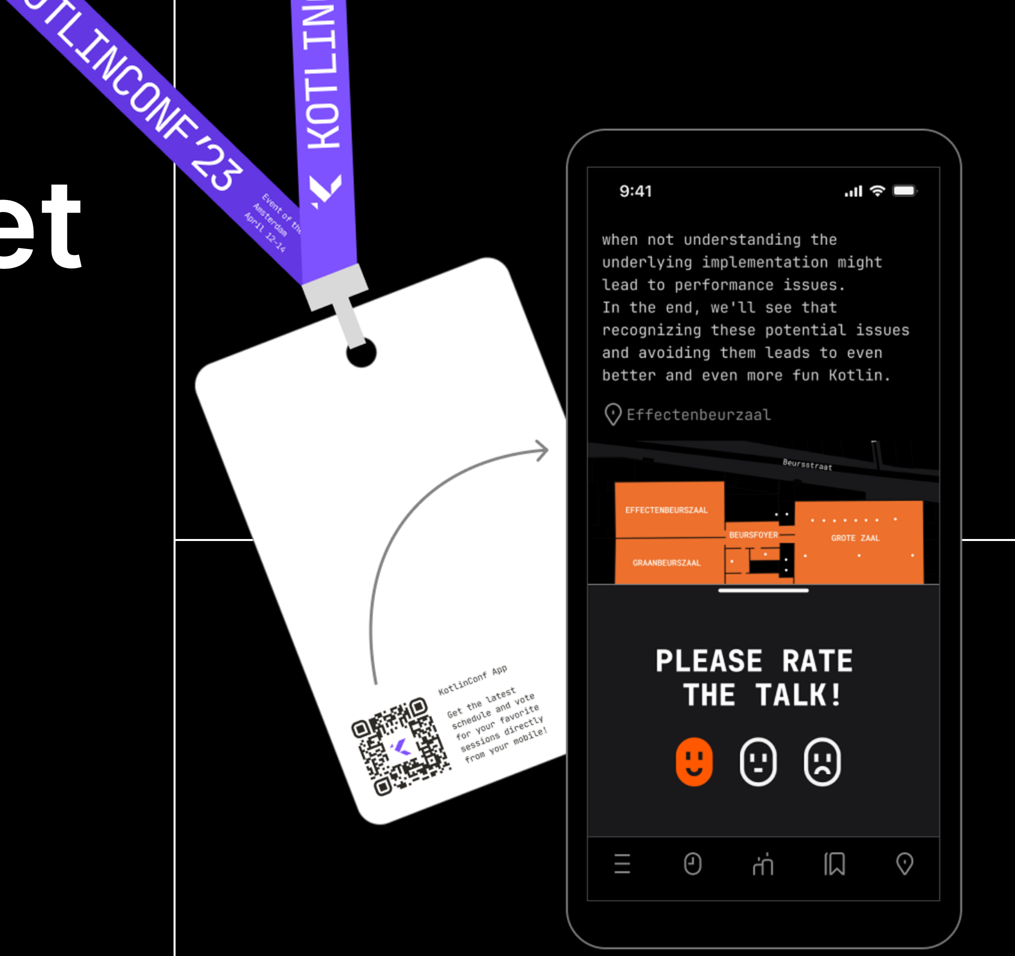
www.linkedin.com/in/alexeysoshin

alexey-soshin.medium.com

@alexey_soshin

KotlinConf'23

Amsterdam


KOTLINCONF'23
KOTLIN
Event of the
Amsterdam
April 12-14
KotlinConf App
Get the latest schedule and vote for your favorite sessions directly from your mobile!
9:41

when not understanding the underlying implementation might lead to performance issues. In the end, we'll see that recognizing these potential issues and avoiding them leads to even better and even more fun Kotlin.

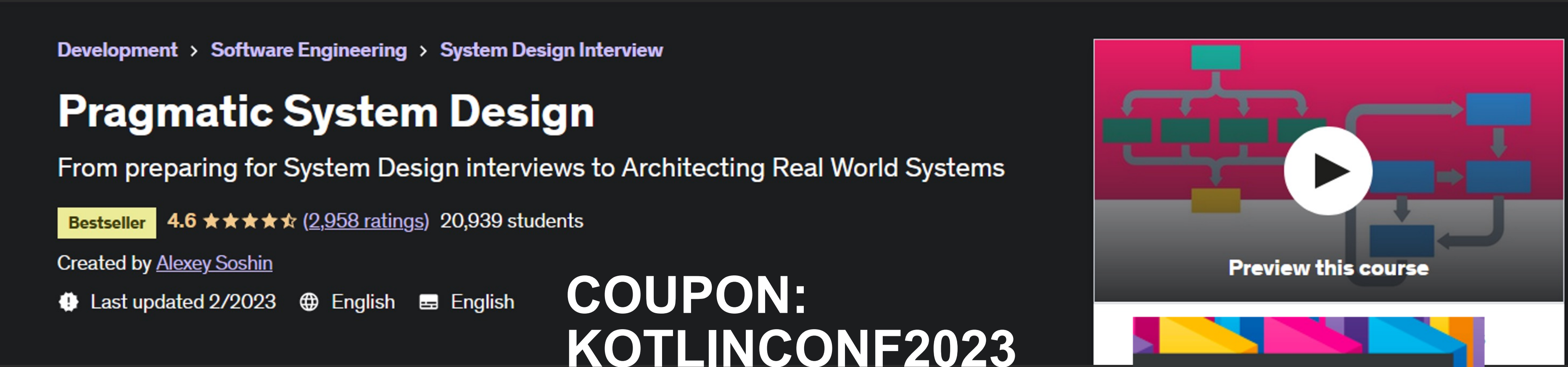
Let’s stay in touch!

Advanced Kotlin Database Development

LinkedIn Learning

www.linkedin.com/learning/advanced-kotlin-database-development

Let's stay in touch!



Development › Software Engineering › System Design Interview

Pragmatic System Design

From preparing for System Design interviews to Architecting Real World Systems

Bestseller 4.6 ★★★★☆ (2,958 ratings) 20,939 students

Created by Alexey Soshin

Last updated 2/2023 English English

Preview this course

COUPON: KOTLINCONF2023

Kotlin Design Patterns and Best Practices

Second Edition

Build scalable applications using traditional, reactive, and concurrent design patterns in Kotlin

Alexey Soshin

Foreword by Anton Arhipov, Kotlin Developer Advocate at JetBrains

Glory to Ukraine!

🇺🇦