

Essential Slick

Dave Gurnell, @davegurnell



Slick is...

A database library

NOT an ORM

Functional

Asynchronous

We'll cover...

Tables

Queries

Actions

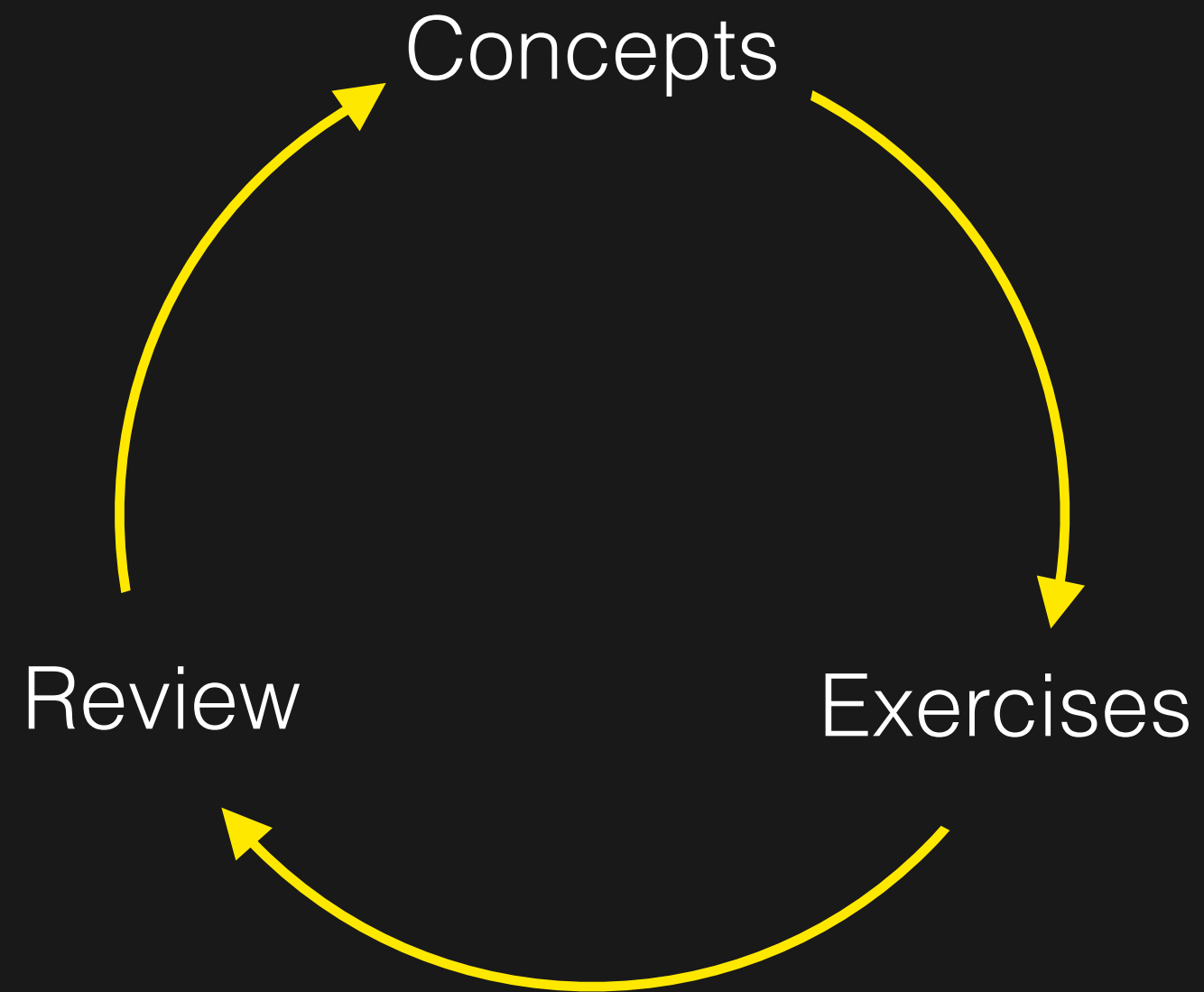
Joins

Profiles

[https://github.com/underscoreio/
scala15-slick](https://github.com/underscoreio/scala15-slick)

[https://github.com/underscoreio/
scalax15-slick](https://github.com/underscoreio/scalax15-slick)

[https://gitter.im/underscoreio/
scalax15-slick](https://gitter.im/underscoreio/scalax15-slick)



Your Turn

Do some fun programming stuff

Do this bit first

Then this bit

And this bit, just for completeness

Essential Slick

Richard Dallaway
Jonathan Ferguson



underscore

underscore



Essential Slick

Richard Dallaway
Jonathan Ferguson



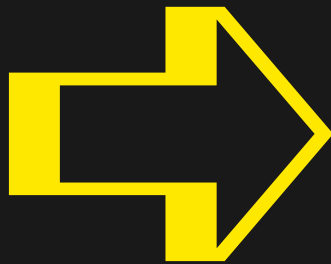
[http://underscore.io/books/
essential-slick](http://underscore.io/books/essential-slick)

Tables

Tables

Basic Construction

tables.Main



Your Turn

Add a “year” column to “Album”

Add a field to the case class

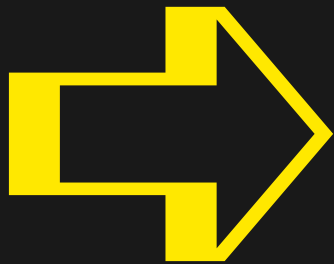
Add a column to the table

Add the column to the projection

Tables

Custom Column Types

tables.Rating



Your Turn

Add a “rating” column of type “Rating”

Add a field to the case class

Add a column to the table

Add the column to the projection

Select Queries

Select Queries

Queries vs Actions

```
db.run(myAction)
```

```
db.run(myAction)
```

```
AlbumTable  
  .filter(_.artist === "Spice Girls")  
  .result
```

```
AlbumTable  
  .filter(_.artist === "Spice Girls")  
  .result
```

Create a query

Convert to an action

```
db.run(  
  AlbumTable  
    .filter(_.artist === "Spice Girls")  
    .result  
)
```

Create a query

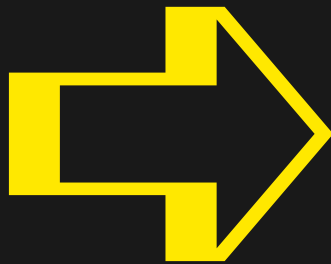
Convert to an action

Run against the DB

Select Queries

Combinators

queries.Main



```
// Select everything  
val selectAllQuery =  
    AlbumTable
```

```
// Select everything  
val selectAllQuery =  
    AlbumTable
```

```
SELECT *  
FROM albums;
```

```
// Filter results  
val selectWhereQuery =  
    AlbumTable  
        .filter(_.artist === "Spice Girls")
```

```
SELECT *  
FROM albums  
WHERE artist = 'Spice Girls';
```

```
// Filter results  
val selectWhereQuery =  
    AlbumTable  
        .filter(_.artist == "Spice Girls")
```

```
SELECT *  
FROM albums  
WHERE artist = 'Spice Girls';
```

argument is an
AlbumTable



```
// Sort results  
val selectSortedQuery1 =  
    AlbumTable  
        .sortBy(_.year.asc)
```

```
SELECT *  
FROM albums  
ORDER BY year ASC;
```

argument is an
AlbumTable



```
// Sort results  
val selectSortedQuery2 =  
    AlbumTable  
        .sortBy(a => (a.year.asc, a.rating.asc))
```

```
SELECT *  
FROM albums  
ORDER BY year ASC, rating ASC;
```

```
// Page results  
val selectPagedQuery =  
    AlbumTable  
        .drop(2).take(1)
```

```
SELECT *  
FROM albums  
OFFSET 2 LIMIT 1;
```



```
// Project results  
val selectColumnsQuery1 =  
    AlbumTable  
        .map(_.title)  
        .result
```

```
SELECT title  
FROM albums;
```

```
// Project results  
val selectColumnsQuery2 =  
    AlbumTable  
        .map(a => (a.artist, a.title))  
        .result
```

```
SELECT artist, title  
FROM albums;
```

```
// Multiple combinators  
val selectCombinedQuery =  
    AlbumTable  
        .filter(_.artist === "Keyboard Cat")  
        .map(_.title)
```

```
SELECT title  
FROM albums  
WHERE artist = 'Keyboard Cat';
```

Your Turn

Select albums **released after 1990**
with a rating of “NotBad” or higher
sorted by artist

Select **the titles of the albums**
in ascending year order

Select Queries

Types

Query[P, U, C]

P - “Packed” / query type

U - “Unpacked” / result type

C - Collection type

Query[AlbumTable, Album, Seq]

P - “Packed” / query type

U - “Unpacked” / result type

C - Collection type

```
val q0: Query[AlbumTable, Album, Seq] =  
  AlbumTable
```

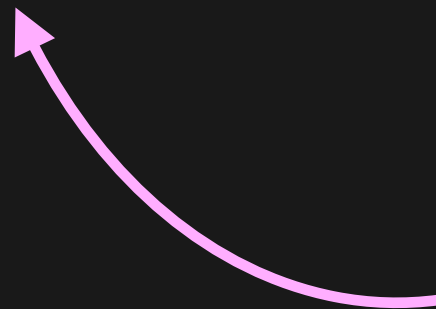


```
val q0: Query[AlbumTable, Album, Seq] =  
  AlbumTable
```

```
val q1: Query[AlbumTable, Album, Seq] =  
  q0.filter(_.year === 1987)
```

```
val q0: Query[AlbumTable, Album, Seq] =  
  AlbumTable
```

```
val q1: Query[AlbumTable, Album, Seq] =  
  q0.filter(_.year === 1987)
```



argument is an
AlbumTable

```
val q0: Query[AlbumTable, Album, Seq] =  
  AlbumTable
```

```
val q1: Query[AlbumTable, Album, Seq] =  
  q0.filter(_.year === 1987)
```

```
val q2: Query[Rep[String], String, Seq] =  
  q1.map(_.title)
```

```
val q0: Query[AlbumTable, Album, Seq] =  
  AlbumTable
```

```
val q1: Query[AlbumTable, Album, Seq] =  
  q0.filter(_.year === 1987)
```

```
val q2: Query[Rep[String], String, Seq] =  
  q1.map(_.title)
```

```
val q3: Query[???, ???, Seq] =  
  q2.map(_.???)
```

Rep[T]

“An SQL expression of type T”

```
val q =  
  AlbumTable  
    .filter(t => t.artist === "Keyboard Cat")  
    .map(t => t.id)
```

Rep[String]



```
val q =  
  AlbumTable  
    .filter(t => t.artist === "Keyboard Cat")  
    .map(t => t.id)
```

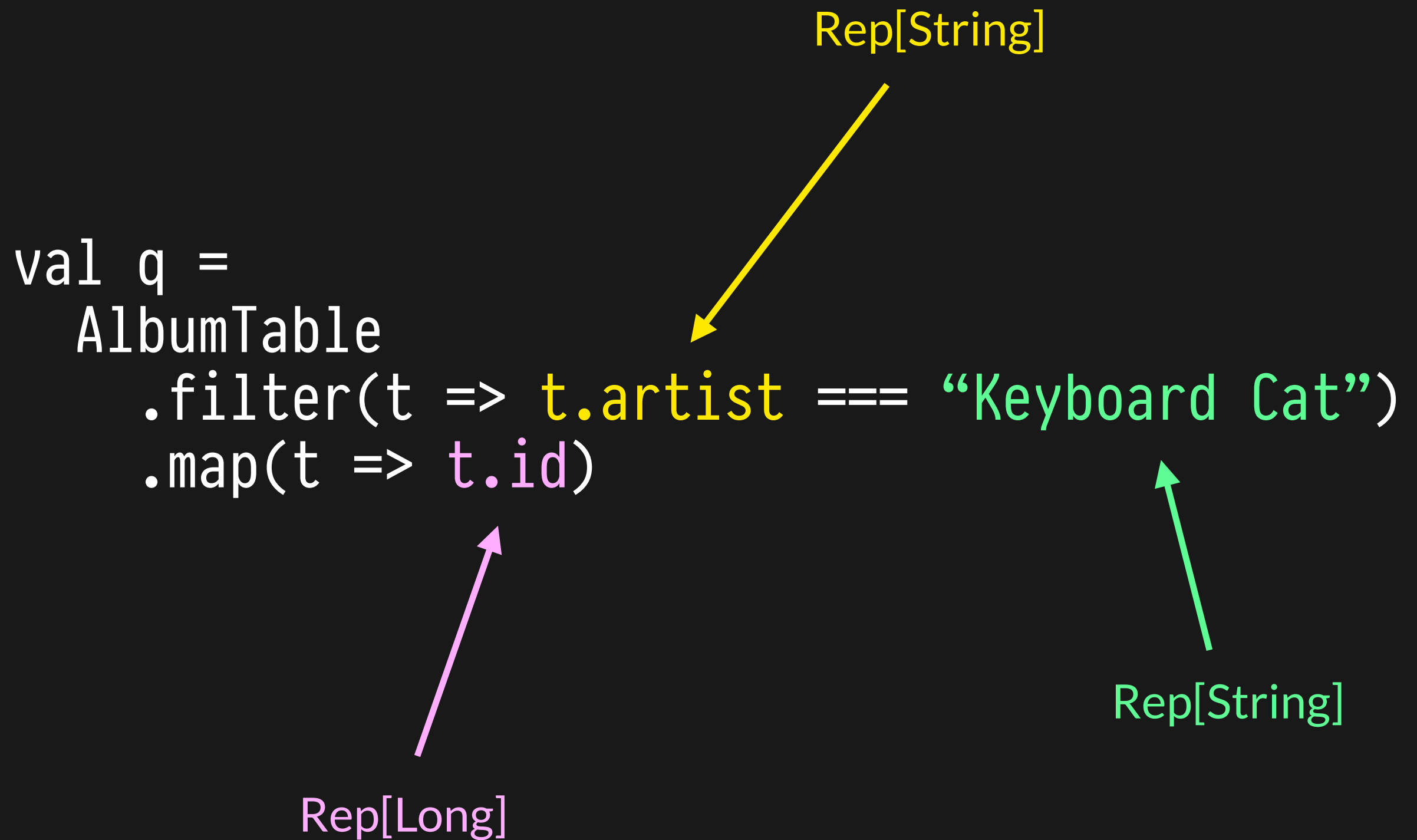
Rep[String]



```
val q =  
  AlbumTable  
    .filter(t => t.artist === "Keyboard Cat")  
    .map(t => t.id)
```



Rep[Long]



Rep[Boolean]



```
val q =  
  AlbumTable  
    .filter(t => t.artist ==> "Keyboard Cat")  
    .map(t => t.id)
```



Rep[Long]

Your Turn

Add type annotations to:

`selectWhereQuery`

`selectColumnsQuery1`

`selectColumnsQuery2`

and your answers to the previous exercise

Actions

Actions

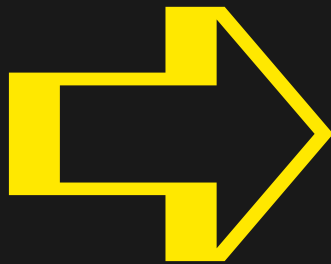
Examples

Select

Insert, update, delete

Create, drop tables

actions.Main



```
val selectAction =  
    AlbumTable  
        .filter(_.artist === "Keyboard Cat")  
        .result
```

```
SELECT *  
FROM albums  
WHERE artist = "Keyboard Cat";
```



```
val updateAction =  
  AlbumTable  
    .filter(_.artist === "Keyboard Cat")  
    .map(_.title)  
    .update("Even Greater Hits")
```

```
UPDATE albums  
SET title = "Even Greater Hits"  
WHERE artist = "Keyboard Cat";
```

```
val updateAction2 =  
  AlbumTable  
    .filter(_.artist === "Keyboard Cat")  
    .map(a => (a.title, a.year))  
    .update(("Even Greater Hits", 2010))
```

```
UPDATE albums  
SET title = "Even Greater Hits",  
    year = 2010  
WHERE artist = "Keyboard Cat";
```

```
val deleteAction =  
    AlbumTable  
        .filter(_.artist === "Keyboard Cat")  
        .delete
```

```
DELETE  
FROM albums  
WHERE artist = "Keyboard Cat";
```

```
val insertAction =  
    AlbumTable += Album(  
        "Pink Floyd",  
        "Dark Side of the Moon",  
        1973,  
        Rating.Awesome)
```

```
INSERT INTO albums  
    (artist, title, year, rating)  
VALUES ("Pink Floyd",  
        "Dark Side of the Moon", 1973, 5);
```

```
val insertAction2 =  
  AlbumTable ++= Seq(  
    album1,  
    album2,  
    album3)
```

```
INSERT INTO albums  
(artist, title, year, rating)  
VALUES (...), (...), (...);
```

```
val createAction =  
    AlbumTable.schema.create
```

```
CREATE TABLE albums (  
    artist TEXT, title TEXT,  
    year INTEGER, rating INTEGER,  
    ...);
```

```
val dropTableAction =  
    AlbumTable.schema.drop
```

```
DROP TABLE albums;
```

Your Turn

Insert **three albums**
by your favourite band

Update albums **released after a specified year**
set their rating to “Meh”

Delete **all albums**
by a user-specified artist

Actions

Types

DBIOAction[R, S, E]

DBIOAction[R, S, E]

R - Result type

S - Streaming or not streaming

E - Effect (read / write / etc...)

DBIOAction[Seq[Album], NoStream, Effect.All]

R - Result type

S - Streaming or not streaming

E - Effect (read / write / etc...)

```
DBIOAction[Seq[Album], NoStream, Effect.All]
```

```
DBIO[Seq[Album]]
```

DBIO[Seq[Album]]

```
DBIO[Seq[Album]]
```



```
db.run(myAction)
```

DBIO[Seq[Album]]



db.run(myAction)

Future[Seq[Album]]

SqlAction[R, S, E]

```
SqlAction[R, S, E] extends DBIOAction[R, S, E]
```

SqlAction[R, S, E]



myAction.statements

Seq[String]

Your Turn

Add type annotations to:

`selectAction`

`insertAction`

`updateAction`

and your answers to the previous exercise

Actions

Combinators

```
// Sequencing independent actions  
def runManyActions() = {  
    exec(action1)  
    exec(action2)  
    exec(action3)  
    exec(action4)  
}
```

```
runManyActions()
```

```
// Sequencing independent actions
val oneBigAction =
    action1 andThen
    action2 andThen
    action3 andThen
    action4

exec(oneBigAction)
```

```
// Chaining interdependent actions
def runChainOfActions(a: SomeInput) = {
  val b = exec(createAction1(a))
  val c = exec(createAction2(b))
  val d = exec(createAction3(c))
  val e = exec(createAction4(d))
  e
}
```

runChainOfActions()


```
// Chaining interdependent actions
def chainOfActions(a: SomeInput) = for {
  b <- createAction1(a)
  c <- createAction2(b)
  d <- createAction3(c)
  e <- createAction4(d)
} yield e
```

```
exec(chainOfActions(someInput))
```

```
// Transactions!  
exec(action.transactionally)
```

Your Turn

Create a single action that:

Accepts an artist, title, and year of release

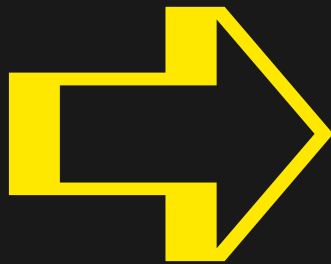
Inserts them into the database

Rate it “Awesome” if it’s their first album

Rate it “Meh” otherwise

Joins

joins.Main



Joins

“Implicit” Joins

```
SELECT *  
FROM album, artist  
WHERE album.id = artist.albumId;
```

```
val query = for {  
  album  <- AlbumTable  
  artist <- ArtistTable  
  if album.id === artist.albumId  
} yield (album, artist)
```

```
SELECT *  
FROM album, artist  
WHERE album.id = artist.albumId;
```


Your Turn

Rewrite the main method
as one big action

Create an implicit join that finds:
artists who have released albums
and the albums they released
sorted by artist name

Joins

“Explicit” Joins

```
SELECT *  
FROM artist INNER JOIN album  
    ON artist.id = album.artistId;
```

```
val query =  
    ArtistTable join AlbumTable  
    on { (artist, album) =>  
        artist.id === album.artistId  
    }
```

```
SELECT *  
FROM artist INNER JOIN album  
    ON artist.id = album.artistId;
```

Your Turn

Create an *explicit* join that finds:
artists who have released albums
and the albums they released
sorted by artist name

Profiles

Profiles

Getting Database Profiles

Shipped With Slick

Derby, H2, HyperSQL, MySQL, PostgreSQL, SQLite

Commercially Available

Oracle, DB2, MSSQL

Shipped With Slick

Derby, H2, HyperSQL, MySQL, PostgreSQL, SQLite

Part of Freeslick

Oracle, DB2, MSSQL

Shipped With Slick

Derby, H2, HyperSQL, MySQL, PostgreSQL, SQLite

Part of Freeslick

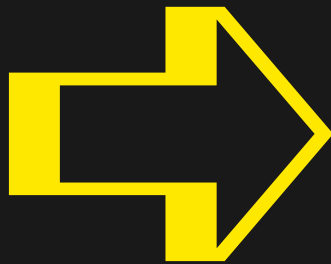
Oracle, DB2, MSSQL

<https://github.com/smootoo/freeslick>

Profiles

Supporting Multiple Databases

profiles.Main



Summary

Defining Tables

Running Single Queries

Sequencing Queries

Joins

~~Aggregate Functions~~

~~Streaming Queries~~

~~Plain SQL Queries~~

~~Query Compilation~~

Essential Slick

Richard Dallaway
Jonathan Ferguson



[http://underscore.io/books/
essential-slick](http://underscore.io/books/essential-slick)

[https://gitter.im/underscoreio/
scalax15-slick](https://gitter.im/underscoreio/scalax15-slick)

Thanks!

Dave Gurnell, @davegurnell

