Scala DB and Future

Десятая лекция

Setup

```
val server = EmbeddedPostgres.builder().setPort(5432).start()

val pgDataSource = new org.postgresql.ds.PGSimpleDataSource()
pgDataSource.setUser("postgres")
val config = new HikariConfig()
config.setDataSource(pgDataSource)
val ctx = new PostgresJdbcContext(LowerCase, new HikariDataSource(config))
```

Connection Pool

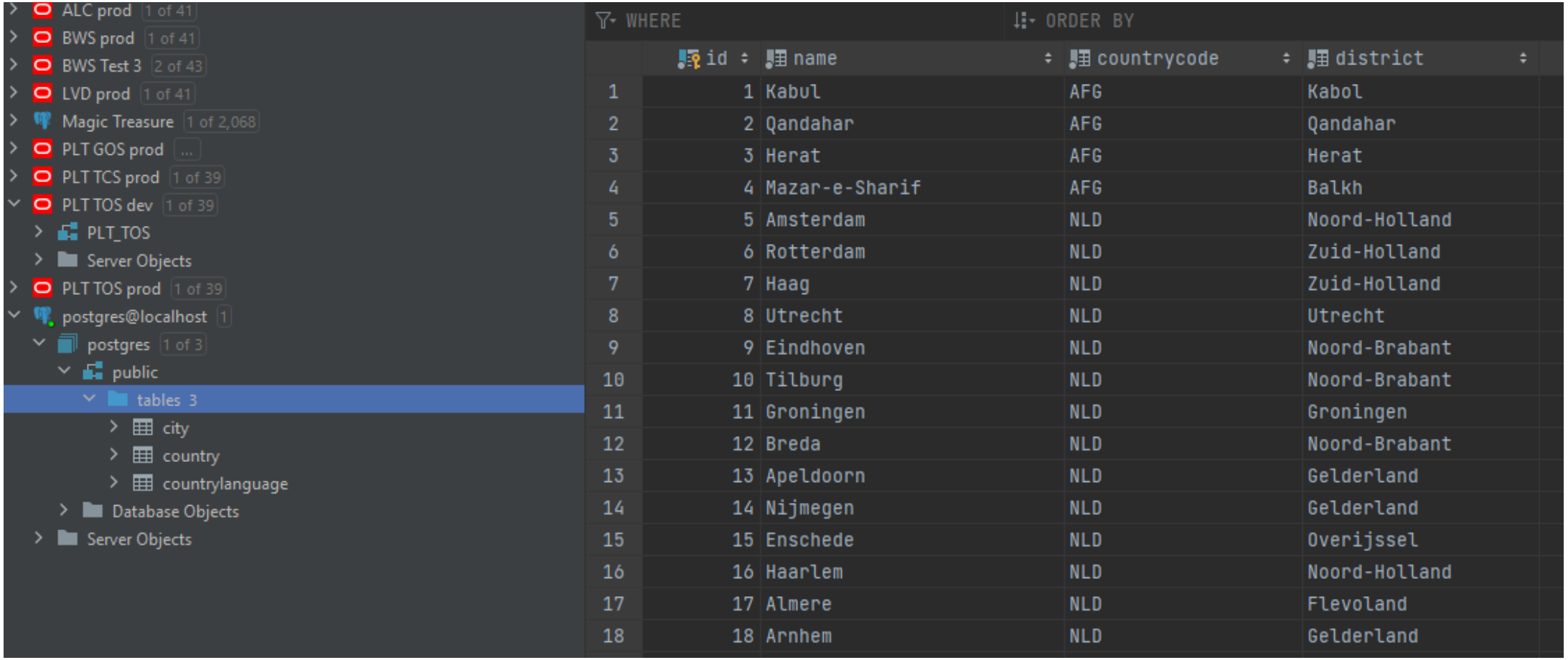
- New connection is expensive
- Reuse connections
- Limits number of connections
- Protects from DDoS
- Cleaner code

Example Data

```sql
CREATE TABLE IF NOT EXISTS city (
    id integer NOT NULL,
    name varchar NOT NULL,
    countrycode character(3) NOT NULL,
    district varchar NOT NULL,
    population integer NOT NULL
);

CREATE TABLE IF NOT EXISTS country (
    code character(3) NOT NULL,
```

Example Data



Type Mapping

Postgres	Scala
real	Float, Double
boolean	Boolean
integer, smallint, bigint	Int, Long
character(n), character varying	String
numberic(n,m)	BigDecimal

City

```
create table city
(
    id          integer not null
        constraint city_pkey
            primary key,
    name        varchar not null,
    countrycode char(3) not null,
    district    varchar not null,
    population  integer not null
);
```

```
case class City(
  id: Int,
  name: String,
  countryCode: String,
  district: String,
  population: Int
)
```

Country

```sql
create table country
(
    code           char(3) not null
        constraint country_pkey
            primary key,
    name           varchar not null,
    continent      varchar not null,
    region         varchar not null,
    surfacearea    real    not null,
    indepyear      smallint,
    population     integer not null,
    lifeexpectancy real,
    gnp            numeric(10, 2),
    gnpold         numeric(10, 2),
    localname      varchar not null,
    governmentform varchar not null,
    headofstate    varchar,
    capital        integer
        constraint country_capital_fkey
            references city,
    code2          char(2) not null
);
```

```scala
case class Country(
  code: String,
  name: String,
  continent: String,
  region: String,
  surfaceArea: Double,
  indepYear: Option[Int],
  population: Int,
  lifeExpectancy: Option[Double],
  gnp: Option[math.BigDecimal],
  gnpold: Option[math.BigDecimal],
  localName: String,
  governmentForm: String,
  headOfState: Option[String],
  capital: Option[Int],
  code2: String
)
```

Country Language

```
create table countrylanguage
(
    countrycode char(3) not null
        constraint countrylanguage_countrycode_fkey
            references country,
    language    varchar not null,
    isofficial  boolean not null,
    percentage  real    not null,
    constraint countrylanguage_pkey
        primary key (countrycode, language)
);
```

```
case class CountryLanguage(
  countrycode: String,
  language: String,
  isOfficial: Boolean,
  percentage: Double
)
```

Select City

```
  println(translate(query[City]))
  val cities = ctx.run(query[City])
  pprint.log(cities.take(5))
```

QueriesExample

CountyExample × QueriesExample ×

```
SELECT x.id, x.name, x.countrycode, x.district, x.population FROM city x
C:\Users\ilya2\Desktop\itmo\lecture10\quill-example\src\main\scala\ru\ifmo\backend_2021\quill_examples\QueriesExample.scala:18
  City(id = 1, name = "Kabul", countryCode = "AFG", district = "Kabol", population = 1780000),
  City(id = 2, name = "Qandahar", countryCode = "AFG", district = "Qandahar", population = 237500),
  City(id = 3, name = "Herat", countryCode = "AFG", district = "Herat", population = 186800),
  City(
    id = 4,
    name = "Mazar-e-Sharif",
    countryCode = "AFG",
    district = "Balkh",
    population = 127800
  ),
  City(
    id = 5,
    name = "Amsterdam",
    countryCode = "NLD",
    district = "Noord-Holland",
    population = 731200
  )
```

Select Country

```
println(translate(query[Country]))
val countries = ctx.run(query[Country])
pprint.log(countries.take(5))
```

```
QueriesExample
CountyExample  QueriesExample
SELECT x.code, x.name, x.continent, x.region, x.surfacearea, x.indepyear, x.population, x.lifeexpectancy, x.gnp, x.gn
C:\Users\ilya2\Desktop\itmo\lecture10\quill-example\src\main\scala\ru\ifmo\backend_2021\quill_examples\QueriesExample
  Country(
    code = "AFG",
    name = "Afghanistan",
    continent = "Asia",
    region = "Southern and Central Asia",
    surfaceArea = 652090.0,
    indepYear = Some(value = 1919),
    population = 22720000,
    lifeExpectancy = Some(value = 45.9000015),
    gnp = Some(value = 5976.00),
    gnpold = None,
    localName = "Afganistan/Afqanestan",
    governmentForm = "Islamic Emirate",
    headOfState = Some(value = "Mohammad Omar"),
    capital = Some(value = 1),
    code2 = "AF"
  ),
```

Select Language

```
println(translate(query[CountryLanguage]))
val languages = ctx.run(query[CountryLanguage])
pprint.log(languages.take(5))
```

```
QueriesExample
CountyExample × QueriesExample ×
SELECT x.countrycode, x.language, x.isofficial, x.percentage FROM countrylanguage x
C:\Users\ilya2\Desktop\itmo\lecture10\quill-example\src\main\scala\ru\ifmo\backend_202
  CountryLanguage(
    countrycode = "AFG",
    language = "Pashto",
    isOfficial = true,
    percentage = 52.4000015
  ),
  CountryLanguage(
    countrycode = "NLD",
    language = "Dutch",
    isOfficial = true,
    percentage = 95.5999985
  ),
  CountryLanguage(
    countrycode = "ANT",
    language = "Papiamento",
    isOfficial = true,
    percentage = 86.1999969
  ),
```

Select City Again

```
1 ✔ SELECT x.id, x.name, x.countrycode, x.district, x.population FROM city x;
```

1-500 of 501+ | Tx: Auto | DDL | CSV

	id	name	countrycode	district	population
1	1	Kabul	AFG	Kabol	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200

```
2
```

Filtering

```
object Querying extends App {
  val ctx = CreateCtx()
  import ctx._

  pprint.pprintln(ctx.translate(query[City].filter(_.name == "Singapore")))
  pprint.pprintln(ctx.run(query[City].filter(_.name == "Singapore")))
}
```

```
Run:  CountyExample  Querying
C:\Users\ilya2\.jdks\openjdk-16.0.1\bin\java.exe ...
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
"SELECT x1.id, x1.name, x1.countrycode, x1.district, x1.population FROM city x1 WHERE x1.name = 'Singapore'"
List(
  City(
    id = 3208,
    name = "Singapore",
    countryCode = "SGP",
    district = "\u0096",
    population = 4017733
  )
)
```

Filtering by Id

```
10      pprint.pprintln(ctx.translate(query[City].filter(_.id == 3208)))
11      pprint.pprintln(ctx.run(query[City].filter(_.id == 3208)))
12  }
```

Querying

Run: CountyExample × Querying ×

```
)
"SELECT x3.id, x3.name, x3.countrycode, x3.district, x3.population FROM city x3 WHERE x3.id = 3208"
List(
  City(
    id = 3208,
    name = "Singapore",
    countryCode = "SGP",
    district = "\u0096",
    population = 4017733
  )
)
```

Filtering by population

```
    pprint.pprintln(ctx.translate(query[City].filter(_.population > 9000000)))
    pprint.pprintln(ctx.run(query[City].filter(_.population > 9000000)))
}
```

Querying

CountyExample × Querying ×

```
"SELECT x5.id, x5.name, x5.countrycode, x5.district, x5.population FROM city x5 WHERE x5.population > 9000000"
List(
  City(
    id = 206,
    name = "S\u00e3o Paulo",
    countryCode = "BRA",
    district = "S\u00e3o Paulo",
    population = 9968485
  ),
  City(
    id = 939,
    name = "Jakarta",
    countryCode = "IDN",
```

Filtering &&

```
  pprint.pprintln(ctx.translate(query[City].filter(c ⇒ c.population > 9000000 && c.countryCode == "CHN")))
  pprint.pprintln(ctx.run(query[City].filter(c ⇒ c.population > 9000000 && c.countryCode == "CHN")))
}
```

Querying › λ(c: Any)

CountyExample × Querying ×

```
  City(id = 2822, name = "Karachi", countryCode = "PAK", district = "Sindh", population = 9269265)
)
"SELECT c.id, c.name, c.countrycode, c.district, c.population FROM city c WHERE c.population > 9000000 AND c.countrycode = 'CHN'"
List(
  City(
    id = 1890,
    name = "Shanghai",
    countryCode = "CHN",
    district = "Shanghai",
    population = 9696300
  )
)
```

Lifting

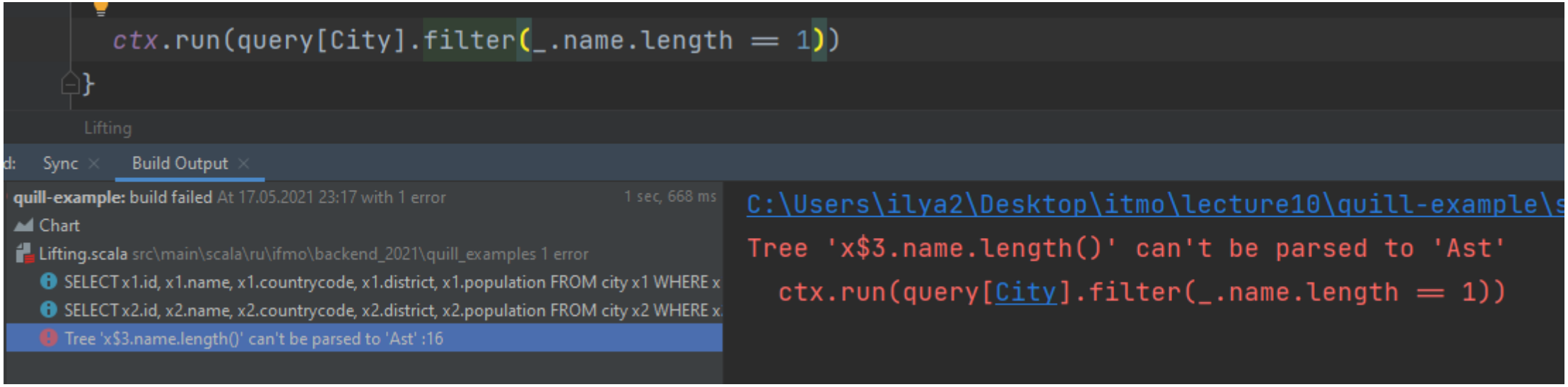
```
  pprint.pprintln(ctx.translate(query[City].filter(_.id == lift(stub))))
  def find(cityId: Int): List[City] = {
    ctx.run(query[City].filter(_.id == lift(cityId)))
  }
  pprint.pprintln(find( cityId = 3208))
  pprint.pprintln(find( cityId = 3209))
}
```

Lifting

CountyExample | Lifting

```
"SELECT x1.id, x1.name, x1.countrycode, x1.district, x1.population FROM city x1 WHERE x1.id = 1"
List(
  City(
    id = 3208,
    name = "Singapore",
    countryCode = "SGP",
    district = "\u0096",
    population = 4017733
  )
)
List(
  City(
    id = 3209,
    name = "Bratislava",
    countryCode = "SVK",
    district = "Bratislava",
    population = 448292
  )
)
```

Lifting limitations



Mapping

```
pprint.pprintln(ctx.translate(query[Country].map(c ⇒ (c.name, c.continent))))
pprint.pprintln(ctx.run(query[Country].map(c ⇒ (c.name, c.continent))))
```

Mapping

CountyExample × Mapping ×

```
"SELECT c.name, c.continent FROM country c"
List(
  ("Afghanistan", "Asia"),
  ("Netherlands", "Europe"),
  ("Netherlands Antilles", "North America"),
  ("Albania", "Europe"),
  ("Algeria", "Africa"),
  ("American Samoa", "Oceania"),
  ("Andorra", "Europe"),
  ("Angola", "Africa"),
  ("Anguilla", "North America"),
```

Mapping more

```
  pprint.pprintln(ctx.translate(query[Country].map(c ⇒ (c.name, c.continent, c.population))))
  pprint.pprintln(ctx.run(query[Country].map(c ⇒ (c.name, c.continent, c.population))))
}
```

Mapping

CountyExample × Mapping ×

```
"SELECT c.name, c.continent, c.population FROM country c"
List(
  ("Afghanistan", "Asia", 22720000),
  ("Netherlands", "Europe", 15864000),
  ("Netherlands Antilles", "North America", 217000),
  ("Albania", "Europe", 3401200),
  ("Algeria", "Africa", 31471000),
  ("American Samoa", "Oceania", 68000),
  ("Andorra", "Europe", 78000),
  ("Angola", "Africa", 12878000),
```

Mapping and Lifting

```
  def findName(cityId: Int): List[String] = {
    pprint.pprintln(ctx.translate(query[City].filter(_.id == lift(cityId)).map(_.name)))
    ctx.run(query[City].filter(_.id == lift(cityId)).map(_.name))
  }
  pprint.pprintln(findName( cityId = 3208))
  pprint.pprintln(findName( cityId = 3209))
}
```

Mapping

CountyExample × Mapping ×

```
  ("United States", "North America", 278357000),
  ("Virgin Islands, U.S.", "North America", 93000),
  ("Zimbabwe", "Africa", 11669000),
  ("Palestine", "Asia", 3101000),
  ("Antarctica", "Antarctica", 0),
  ("Bouvet Island", "Antarctica", 0),
  ("British Indian Ocean Territory", "Africa", 0),
  ("South Georgia and the South Sandwich Islands", "Antarctica", 0),
  ("Heard Island and McDonald Islands", "Antarctica", 0),
  ("French Southern territories", "Antarctica", 0),
  ("United States Minor Outlying Islands", "Oceania", 0)
)
"SELECT x1.name FROM city x1 WHERE x1.id = 3208"
List("Singapore")
"SELECT x1.name FROM city x1 WHERE x1.id = 3209"
List("Bratislava")
```

Joins

```
  pprint.pprintln(ctx.run(
    query[City]                                                    : EntityQuery[City]
      .join(query[Country])                                        : JoinQuery[City, Country, (City, Country)]
      .on(_.countryCode == _.code)                                 : Query[(City, Country)]
      .filter{case (city, country) => country.continent == "Asia"} : Query[(City, Country)]
      .map{case (city, country) => city.name}                      : Query[String]
  ))
}
```

Joins

CountyExample × Joins ×

```
"SELECT x1.name FROM city x1 INNER JOIN country x2 ON x1.countrycode = x2.code WHERE x2.continent = 'Asia'"
List(
  "Kabul",
  "Qandahar",
  "Herat",
  "Mazar-e-Sharif",
  "Dubai",
  "Abu Dhabi",
  "Sharja",
  "al-Ayn",
  "Ajman",
  "Yerevan",
  "Gjumri",
```

Inserts

```
pprint.pprintln(ctx.translate(query[City].insert(City(10000, "test", "TST", "Test County", 0))))
pprint.pprintln(ctx.run(query[City].insert(City(10000, "test", "TST", "Test County", 0))))

pprint.pprintln(ctx.run(query[City].filter(_.population == 0)))
```

Inserts

CountyExample × Inserts ×

```
C:\Users\ilya2\.jdks\openjdk-16.0.1\bin\java.exe ...
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
"INSERT INTO city (id,name,countrycode,district,population) VALUES (10000, 'test', 'TST', 'Test County', 0)"
1L
List(City(id = 10000, name = "test", countryCode = "TST", district = "Test County", population = 0))
```

Batch insert

```
pprint.pprintln(ctx.translate(liftQuery(cities).foreach(e ⇒ query[City].insert(e))))
pprint.pprintln(ctx.run(liftQuery(cities).foreach(e ⇒ query[City].insert(e))))
pprint.pprintln(ctx.run(query[City].filter(_.population == 0)))
```

Inserts

CountyExample × Inserts ×

```
List(
  "INSERT INTO city (id,name,countrycode,district,population) VALUES (10001, 'testville', 'TSV', 'Test County', 0)",
  "INSERT INTO city (id,name,countrycode,district,population) VALUES (10002, 'testopolis', 'TSO', 'Test County', 0)",
  "INSERT INTO city (id,name,countrycode,district,population) VALUES (10003, 'testberg', 'TSB', 'Test County', 0)"
)
List(1L, 1L, 1L)
List(
  City(id = 10000, name = "test", countryCode = "TST", district = "Test County", population = 0),
  City(
    id = 10001,
    name = "testville",
    countryCode = "TSV",
    district = "Test County",
    population = 0
  ),
  City(
    id = 10002,
    name = "testopolis",
    countryCode = "TSO"
```

Updates

```
  pprint.pprintln(ctx.translate(query[City].filter(_.id == 10000).update(City(10000, "testham", "TST", "Test County", 0))))
  pprint.pprintln(ctx.run(
    query[City]
      .filter(_.id == 10000)
      .update(City(10000, "testham", "TST", "Test County", 0))
  ))

  pprint.pprintln(ctx.run(query[City].filter(_.id == 10000)))
}
```

Updates

CountyExample × Updates ×

```
C:\Users\ilya2\.jdks\openjdk-16.0.1\bin\java.exe ...
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
"UPDATE city SET id = 10000, name = 'testham', countrycode = 'TST', district = 'Test County', population = 0 WHERE id = 10000"
1L
List(
  City(id = 10000, name = "testham", countryCode = "TST", district = "Test County", population = 0)
)
```

Update fields

```
pprint.pprintln(ctx.translate(query[City].filter(_.id == 10000).update(_.name → "testford")))
pprint.pprintln(ctx.run(
  query[City]
    .filter(_.id == 10000)
    .update(_.name → "testford")
)
)
pprint.pprintln(ctx.run(query[City].filter(_.id == 10000)))
```

Updates

CountyExample × Updates ×

```
"UPDATE city SET name = 'testford' WHERE id = 10000"
1L
List(
  City(id = 10000, name = "testford", countryCode = "TST", district = "Test County", population = 0)
)
```

Update multiple rows

```
  pprint.pprintln(ctx.translate(query[City].filter(_.district == "Test County").update(_.district → "Test Borough")))
  pprint.pprintln(ctx.run(
    query[City]
      .filter(_.district == "Test County")
      .update(_.district → "Test Borough")
  )
  )
  pprint.pprintln(ctx.run(query[City].filter(_.population == 0)))
}
```

Updates

CountyExample × Updates ×

```
"UPDATE city SET district = 'Test Borough' WHERE district = 'Test County'"
4L
List(
  City(
    id = 10001,
    name = "testville",
    countryCode = "TSV",
    district = "Test Borough",
    population = 0
  ),
  City(
    id = 10002,
    name = "testopolis"
```

Transactions

```
pprint.pprintln(ctx.translate(query[City].filter(_.district == "Test Borough").update(_.district → "Test County")))
Try {
  pprint.pprintln(ctx.transaction {
    ctx.run(
      query[City].filter(_.district == "Test Borough").update(_.district → "Test County")
    )
    throw new Exception()
  }
  )
}.failed.foreach(println(_))

pprint.pprintln(ctx.run(query[City].filter(_.population == 0)))
```

Transactions

CountyExample × Transactions ×

```
"UPDATE city SET district = 'Test County' WHERE district = 'Test Borough'"
java.lang.Exception
List(
  City(id = 10000, name = "test", countryCode = "TST", district = "Test Borough", population = 0),
  City(
    id = 10003,
    name = "testberg",
    countryCode = "TSB",
    district = "Test Borough",
    population = 0
  ),
  City(
    id = 10001,
```

MessageDB

```
class DBAdapter() extends MessageDB {
  val server: EmbeddedPostgres = EmbeddedPostgres.builder()
    .setDataDirectory("./data")
    .setCleanDataDirectory(false)
    .setPort(5432)
    .start()
  val pgDataSource = new org.postgresql.ds.PGSimpleDataSource()
  pgDataSource.setUser("postgres")
  val hikariConfig = new HikariConfig()
  hikariConfig.setDataSource(pgDataSource)
  val ctx = new PostgresJdbcContext(LowerCase, new HikariDataSource(hikariConfig))
  ctx.executeAction( sql = "CREATE TABLE IF NOT EXISTS message (username text, message text);")

  import ctx._

  def getMessages: List[Message] =
    ctx.run(query[Message])
  def addMessage(message: Message): Unit =
    ctx.run(query[Message].insert(lift(message)))
}
```

Web crawler

```
object FetchLinks extends App {
  def fetchLinks(title: String): Seq[String] = {
    val resp = requests.get(
      "https://en.wikipedia.org/w/api.php",
      params = Seq(
        "action" → "query",
        "titles" → title,
        "prop" → "links",
        "format" → "json"
      )
    )
    for{
      page ← ujson.read(resp)("query")("pages").obj.values.toSeq
      links ← page.obj.get("links").toSeq
      link ← links.arr
    } yield link("title").str
  }

  pprint.pprintln(fetchLinks( title = "Albert Einstein"))
}
```

Sequential crawler

```scala
object SequentialCrawler extends App {
  def fetchAllLinks(startTitle: String, depth: Int): Set[String] = {
    var seen = Set(startTitle)
    var current = Set(startTitle)
    for (i ← Range(0, depth)) {
      val nextTitleLists = for (title ← current) yield fetchLinks(title)
      current = nextTitleLists.flatten.filter(!seen.contains(_))
      seen = seen ++ current
    }
    seen
  }
  TimeColored {
    pprint.pprintln(fetchAllLinks( startTitle = "Albert Einstein",  depth = 2))
  }
}
```

SequentialCrawler

ParallelCrawler × SequentialCrawler ×

```
  "Apparent horizon",
  "1. Liga (ice hockey)",
  "Adam Smith",
  "Amal Kumar Raychaudhuri",
  "A series and B series",
  "Abdus Salam"
)
[time] 2842,966ms
```

Parallel crawler

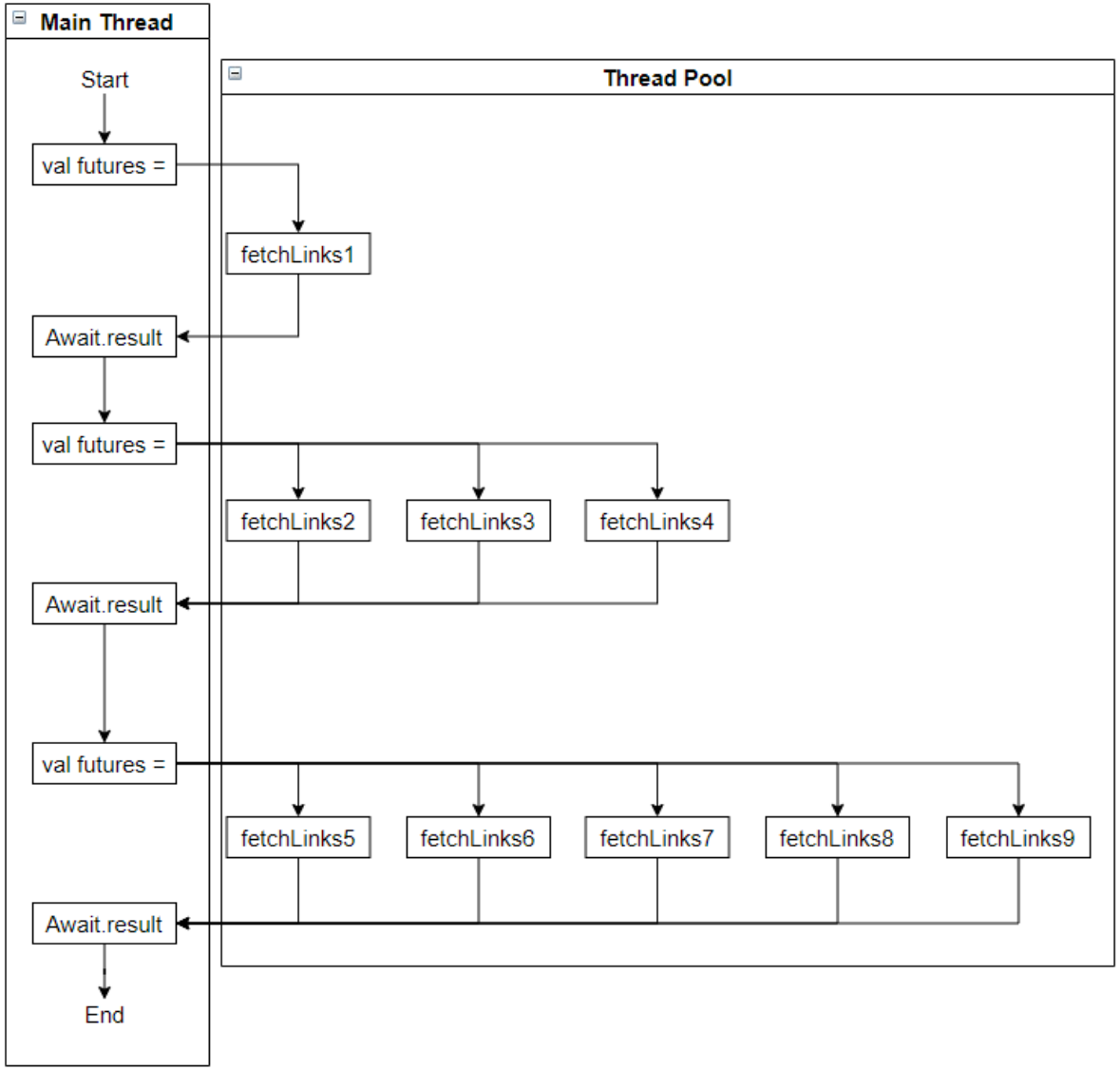
```scala
object ParallelCrawler extends App {
  implicit val ec = ExecutionContext.fromExecutorService(Executors.newFixedThreadPool( nThreads = 8))
  def fetchAllLinksParallel(startTitle: String, depth: Int): Set[String] = {
    var seen = Set(startTitle)
    var current = Set(startTitle)
    for (i ← Range(0, depth)) {
      val futures = for (title ← current) yield Future {
        fetchLinks(title)
      }
      val nextTitleLists = futures.map(Await.result(_, Inf))
      current = nextTitleLists.flatten.filter(!seen.contains(_))
      seen = seen ++ current
    }
    seen
  }
```

ParallelCrawler

ParallelCrawler × SequentialCrawler ×

```
  "ISBN (identifier)",
  "Abba Eban",
  "Apparent horizon",
  "1. Liga (ice hockey)",
  "Adam Smith",
  "Amal Kumar Raychaudhuri",
  "A series and B series",
  "Abdus Salam"
)
[time] 1396,216ms
```

Fork-join parallelism



Recursive crawler

```
object RecursiveCrawler extends App {
  implicit val ec = ExecutionContext.fromExecutorService(Executors.newFixedThreadPool( nThreads = 8))
  def fetchAllLinksRec(startTitle: String, depth: Int): Set[String] = {
    def innerFetchAllLinks(current: Set[String], seen: Set[String], recDepth: Int): Set[String] = {
      if (recDepth ⩾ depth) seen
      else {
        val futures = for (title ← current) yield Future {
          fetchLinks(title)
        }
        val nextTitles = futures.flatMap(Await.result(_, Inf))
        innerFetchAllLinks(nextTitles.filter(!seen.contains(_)), seen ++ nextTitles, recDepth + 1)
      }
    }

    innerFetchAllLinks(Set(startTitle), Set(startTitle),  recDepth = 0)
  }
  TimeColored {
    pprint.pprintln(fetchAllLinksRec( startTitle = "Albert Einstein",  depth = 2))
  }
  ec.shutdown()
```

RecursiveCrawler › fetchAllLinksRec(...) › innerFetchAllLinks(...)

ParallelCrawler × RecursiveCrawler ×

```
  "Apparent horizon",
  "1. Liga (ice hockey)",
  "Adam Smith",
  "Amal Kumar Raychaudhuri",
  "A series and B series",
  "Abdus Salam"
)
[time] 1392,648ms
```

Fetch links asynchronously

```scala
val asyncHttpClient: AsyncHttpClient = Dsl.asyncHttpClient()

def fetchLinksAsync(title: String)(implicit ec: ExecutionContext): Future[Seq[String]] = {
  val p = Promise[String]
  val listenableFuture: ListenableFuture[Response] =
    asyncHttpClient.prepareGet( url = "https://en.wikipedia.org/w/api.php")
      .addQueryParam( name = "action",  value = "query")
      .addQueryParam( name = "titles", title)
      .addQueryParam( name = "prop",  value = "links")
      .addQueryParam( name = "format",  value = "json")
      .execute()

  listenableFuture.addListener(() ⇒ p.success(listenableFuture.get().getResponseBody),  exec = null)
  val scalaFuture: Future[String] = p.future
  scalaFuture.map { responseBody ⇒
    for {
      page ← ujson.read(responseBody)("query")("pages").obj.values.toSeq
      links ← page.obj.get("links").toSeq
      link ← links.arr
    } yield link("title").str
  }
}
```

Async crawler

```
object AsyncCrawler extends App {
  implicit val ec = ExecutionContext.fromExecutorService(Executors.newFixedThreadPool( nThreads = 8))
  def fetchAllLinksAsync(startTitle: String, depth: Int): Future[Set[String]] = {
    def rec(current: Set[String], seen: Set[String], recDepth: Int): Future[Set[String]] = {
      if (recDepth ⩾ depth) Future.successful(seen)
      else {
        val futures = for (title ← current) yield fetchLinksAsync(title)
        Future.sequence(futures).map { nextTitleLists ⇒
          val nextTitles = nextTitleLists.flatten
          rec(nextTitles.filter(!seen.contains(_)), seen ++ nextTitles, recDepth + 1)
        }.flatten
      }
    }

    rec(Set(startTitle), Set(startTitle),  recDepth = 0)
  }
  TimeColored {
    pprint.pprintln(Await.result(fetchAllLinksAsync( startTitle = "Albert Einstein",  depth = 2), Inf))
  }
```

AsyncCrawler › fetchAllLinksAsync(...)

ParallelCrawler × AsyncCrawler ×

```
  "Abba Eban",
  "Apparent horizon",
  "1. Liga (ice hockey)",
  "Adam Smith",
  "Amal Kumar Raychaudhuri",
  "A series and B series",
  "Abdus Salam"
)
[time] 1502,955ms
```

Useful links

- Quill example - https://github.com/Backend-ITMO-2021/quill-example
- WTF Connection pools - https://www.youtube.com/watch?v=xiBd5kTOoYo
- Quill - https://getquill.io/
- ScalikeJDBC - http://scalikejdbc.org/
- Scala slick - https://scala-slick.org/
- Scala Future - https://docs.scala-lang.org/overviews/core/futures.html
- AsyncHttpClient - https://github.com/AsyncHttpClient/async-http-client