Quill -
一個 Scala 的資料庫存取利器

鄭紹志 Vito Jeng

R&D, 亦思科技

is-land
is-land Systems Inc.

vito@is-land.com.tw
@vitojeng

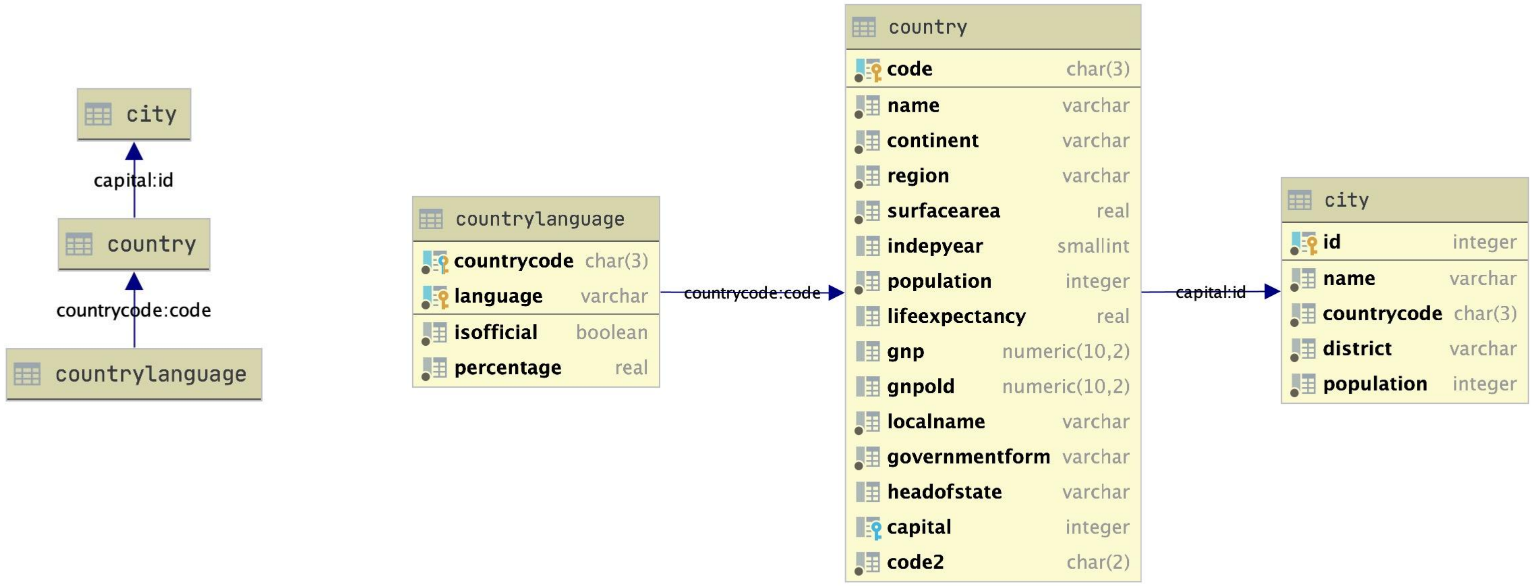
jcconf2020-quill

Join Scala Taiwan

- Gitter channel -
  - https://gitter.im/ScalaTaiwan/ScalaTaiwan/
- Facebook Group -
  - https://www.facebook.com/groups/ScalaTW/
- Meetup -
  - https://www.meetup.com/Scala-Taiwan-Meetup/

PostgreSQL dbsamples

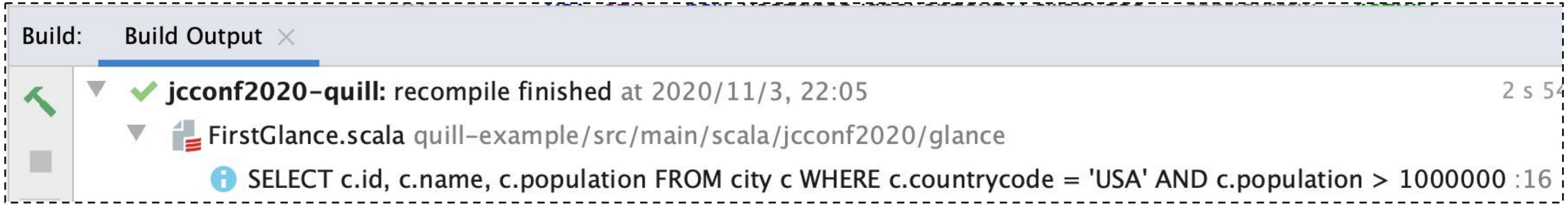
- PostgreSQL dbsamples:
  - https://www.postgresql.org/ftp/projects/pgFoundry/dbsamples/



Quill intro

Compile-time query generation

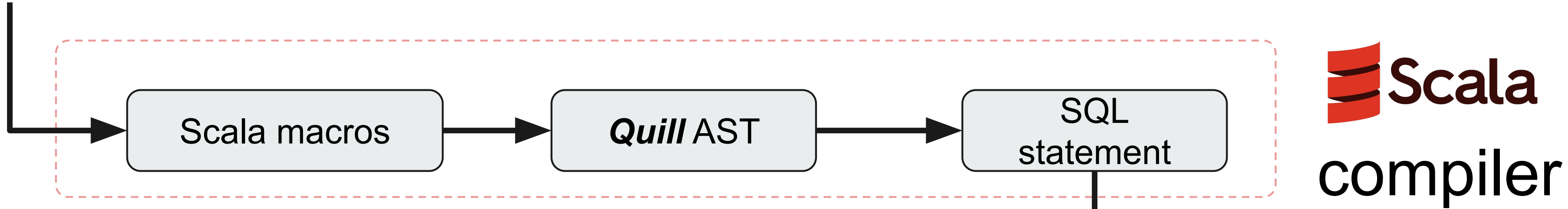
```scala
query[City]
     .filter(c=> c.countryCode=="USA" && c.population > 1000000)
     .map(c=> (c.id, c.name, c.population) )
```

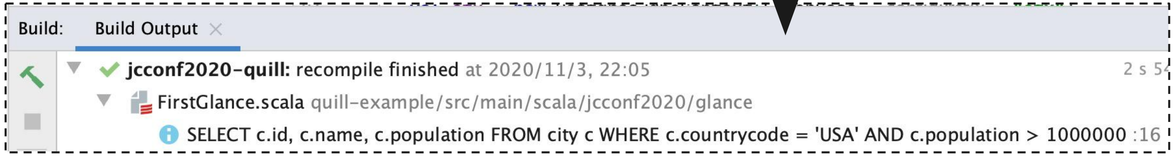


```
Build:   Build Output
▼ ✓ jcconf2020-quill: recompile finished at 2020/11/3, 22:05                2 s 54
    ▼ FirstGlance.scala quill-example/src/main/scala/jcconf2020/glance
          SELECT c.id, c.name, c.population FROM city c WHERE c.countrycode = 'USA' AND c.population > 1000000 :16
```

Compile-time query generation

```
query[City]
    .filter(c=> c.countryCode=="USA" && c.population > 1000000)
    .map(c=> (c.id, c.name, c.population) )
```



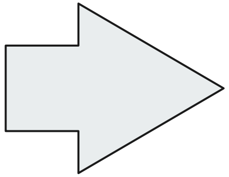


```
jcconf2020-quill: recompile finished at 2020/11/3, 22:05                                         2 s 54
  FirstGlance.scala quill-example/src/main/scala/jcconf2020/glance
    SELECT c.id, c.name, c.population FROM city c WHERE c.countrycode = 'USA' AND c.population > 1000000 :16
```

Simplify mapping

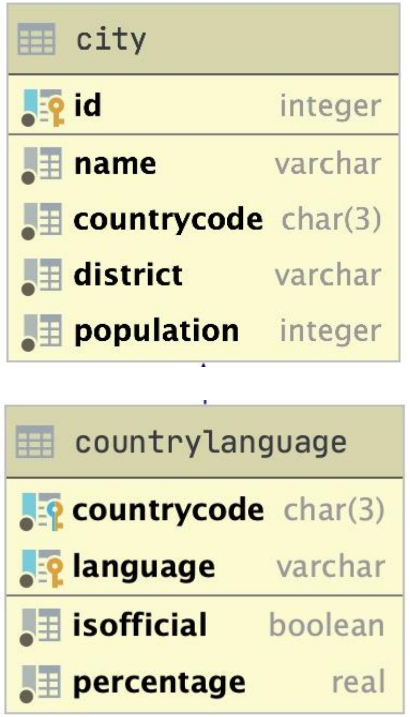
```
case class City(id: Int,
  name: String,
  countryCode: String,
  district: String,
  population: Int)


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



NamingStrategy

- **LowerCase**



city	
id	integer
name	varchar
countrycode	char(3)
district	varchar
population	integer

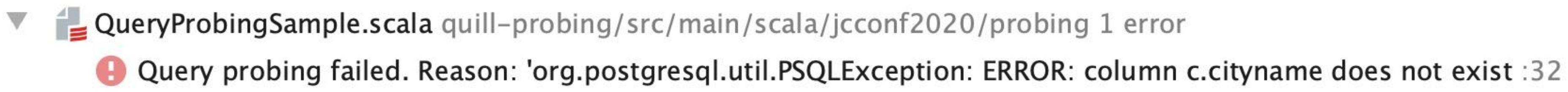
countrylanguage	
countrycode	char(3)
language	varchar
isofficial	boolean
percentage	real

Simplify mapping

Naming strategy	countryCode	country_code
LowerCase	countrycode	country_code
UpperCase	COUNTRYCODE	COUNTRY_CODE
SnakeCase	country_code	country_code
CamelCase	countryCode	countryCode
LiteralCase	countryCode	country_code

Compile-time query validation

- Query Probing - 支援在 Compile-time 進行 SQL 驗證



```
QueryProbingSample.scala quill-probing/src/main/scala/jcconf2020/probing 1 error
    Query probing failed. Reason: 'org.postgresql.util.PSQLException: ERROR: column c.cityname does not exist :32
```

At first glance

Try out in Scastie!

```
import io.getquill._
case class City(id: Int, name: String,
       countryCode: String, district: String, population: Int)
val ctx = new SqlMirrorContext(PostgresDialect, LowerCase)
import ctx._
val q = quote {
  query[City]
    .filter(c=> c.countryCode=="USA" && c.population > 1000000)
    .map(c=> (c.id, c.name, c.population) )
}
val result = ctx.run(q)
```

Quoted[EntityQuery[(Int, String, Int)]]

EntityQuery[(Int, String, Int)]

Quotation

Quill AST

SQL statement

```
SELECT c.id, c.name, c.population
FROM city c
WHERE c.countrycode = 'USA'
  AND c.population > 1000000
```

Quotation's AST(Abstract Syntax Tree)

```
query[City]
    .filter(c=> c.countryCode=="USA" && c.population > 1000000)
    .map(c=> (c.id, c.name, c.population) )
```

```
Map(
  Filter(
    Entity("City", List()),
    Ident("c"),
    BinaryOperation(
      BinaryOperation(Property(Ident("c"), "countryCode"), ==, Constant("USA")),
      &&,
      BinaryOperation(Property(Ident("c"), "population"), >, Constant(1000000))
    )
  ),
  Ident("c"),
  Tuple(
    List(
      Property(Ident("c"), "id"),
      Property(Ident("c"), "name"),
      Property(Ident("c"), "population")
    )
  )
)
```

Many databases(libraries) support

Async support

- Integrate wih
  - NDBC(Postgres)
  - JAsync(Mysql, Postgres)
  - Monix(Mysql, Postgres, Sqlite, H2, SQL Server, Oracle)
  - Finagle(Mysql, Postgres)

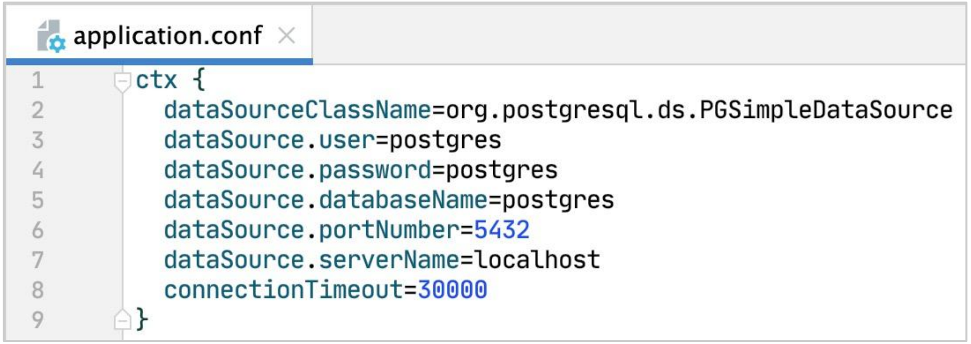
Restrictions

- Single database
- Return a flat relation type
- Limited operations

Quill basic & operations

SQL Context

- 必須先建立 Context 物件才能建立 Quotatiaon
- 自 config file 讀取設定檔
    - 使用 typesafe config library
    - 使用 HikariCP library(Connection Pool)
    - **application.conf / application.properties**
    - `val ctx = new PostgresJdbcContext(LowerCase, "ctx")`



application.conf

```
ctx {
    dataSourceClassName=org.postgresql.ds.PGSimpleDataSource
    dataSource.user=postgres
    dataSource.password=postgres
    dataSource.databaseName=postgres
    dataSource.portNumber=5432
    dataSource.serverName=localhost
    connectionTimeout=30000
}
```

Schema customization

- 使用 querySchema 自訂 mapping 關係

```scala
case class MyCity(id: Int, city_name: String,
       country_code: String, district: String,
       population: Int)

val schemaCity = quote {
  querySchema[MyCity]("city",
    _.country_code -> "countrycode",
    _.city_name -> "name"
  )
}
val queryCities = quote {
  schemaCity.filter(c=>c.country_code=="TWN")
}
ctx.run(queryCities)
```

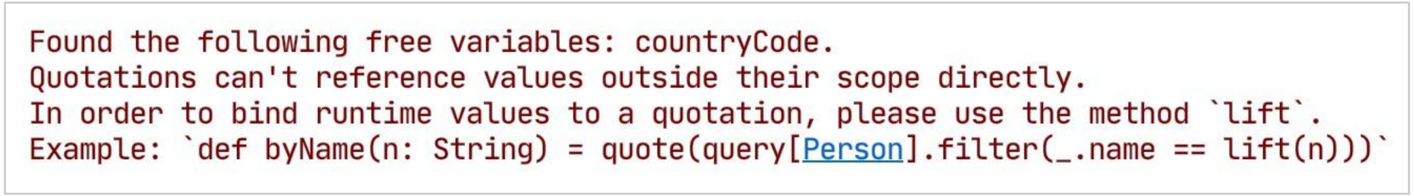
Basic operations

- filter / map / flatMap / concatMap / sortBy / drop / take
- isEmpty / nonEmpty / contains / distinct
- groupBy
- aggregation: min / max / avg / sum / size
- union / unionAll(++)

Binding value - 帶入 Runtime 參數 (1)

- Quotation 無法直接引用外部的值

```scala
def queryCountryCities(countryCode: String) = quote {
  query[City].filter(c=>c.countryCode==countryCode)
}
```



```
Found the following free variables: countryCode.
Quotations can't reference values outside their scope directly.
In order to bind runtime values to a quotation, please use the method `lift`.
Example: `def byName(n: String) = quote(query[Person].filter(_.name == lift(n)))`
```

- Binding single value: use `lift` method
- Binding multiple values: use `liftQuery` method

Binding value - 帶入 Runtime 參數 (2)

```
def queryCountryCities(countryCode: String) = quote {
  query[City].filter(c=>c.countryCode==countryCode)
}
```

```
Cause compile error!!
```

```
def queryCountryCities(countryCode: String) = quote {
  query[City].filter(c=>c.countryCode==lift(countryCode))
}
```

```
SELECT c.id, c.name, c.countrycode, c.district, c.population
FROM city c WHERE c.countrycode = ?
```

```
def queryCountryCities(countries: Seq[String]) = quote {
 query[City].filter(c=> liftQuery(countries).contains(c.countryCode))
}
```

```
SELECT c.id, c.name, c.countrycode, c.district, c.population
FROM city c WHERE c.countrycode IN (?)
```

Joins

- Applicative Joins
  - Common used when join two tables
  - Support inner join / left join / right join / full join
- Implicit Joins
  - Used in **for-comprehension** syntax
  - Only can do inner-join
- Flat Joins
  - Used in **for-comprehension** syntax
  - Support inner join / left join

Joins - Applicative Join

```
val queryAsiaCities = quote {
    query[City]
        .join(query[Country])
        .on { (ci, co) => ci.countryCode == co.code }
        .filter { case (_, co) => co.continent=="Asia" }
        .map { case (ci, co) => (co.name, ci.name) }
}
```

```
SELECT co.name, ci.name
FROM city ci
INNER JOIN country co ON ci.countrycode = co.code
WHERE co.continent = 'Asia'
```

Joins - Implicit Joins

```scala
val queryTaiwanCities = quote {
  for {
    ci <- query[City]
    co <- query[Country].filter(co0=>co0.code=="TWN")
                if (ci.countryCode==co.code)
    cl <- query[CountryLanguage]
                if (co.code==cl.countrycode)
  } yield (ci.name, co.name, cl.language)
}
```

```sql
SELECT ci.name, co0.name, cl.language
FROM city ci, country co0, countrylanguage cl
WHERE co0.code = 'TWN'
  AND ci.countrycode = co0.code
  AND co0.code = cl.countrycode
```

Joins - Flat Join

```
val queryTaiwanCities = quote {
  for {
    co <- query[Country].filter(c=>c.code=="TWN")
    ci <- query[City]
            .leftJoin(co1=>co.code==co1.countryCode)
            .filter(ci1=>ci1.exists(c=>c.population>1000000))
  } yield {
    (co, ci)
  }
}
```

```
SELECT c.code, c.name, c.continent, c.region, c.surfacearea,
  c.indepyear, c.population, c.lifeexpectancy, c.gnp, c.gnpold,
  c.localname, c.governmentform, c.headofstate, c.capital,
  c.code2,
  co1.id, co1.name, co1.countrycode, co1.district, co1.population
FROM country c LEFT JOIN city co1 ON c.code = co1.countrycode
WHERE c.code = 'TWN' AND co1.population > 1000000
```

Demo

- 查詢某個國家
- 查詢某個國家的城市
- 查詢亞洲四小龍的城市

Actions - 新增/刪除/修改

```
query[City].filter(c => c.id==10000)
           .delete
```

```
DELETE FROM city WHERE id = 10000
```

```
query[City].insert(City(10000, "my city", "MYC", "My District", 0))
```

```
INSERT INTO city (id,name,countrycode,district,population)
VALUES (10000, 'my city', 'MYC', 'My District', 0)
```

```
query[City].filter(_.district == "My District")
           .update(_.district -> "My Town")
```

```
UPDATE city SET district = 'My Town' WHERE district = 'My District'
```

Actions - Batch update

- 使用 **liftQuery**

```
val cities = List(
  City(10001, "my city1", "MYC", "My Town", 100000),
  City(10002, "my city2", "MYC", "My Village", 120000),
  City(10003, "my city3", "MYC", "My Borough", 140000)
)
val insertCities = quote {
  liftQuery(cities).foreach(e=>query[City].insert(e))
}
```

```
INSERT INTO city (id,name,countrycode,district,population)
VALUES (?, ?, ?, ?, ?)
```

```
val deleteCities = quote {
  liftQuery(List(10001, 10002, 10003))
          .foreach(id => query[City].filter(c=>c.id==id).delete)
}
```

```
DELETE FROM city WHERE id = ?
```

Transaction

- `JdbcContext` provide transaction support(connection is thread-local)

```
val cities = List(
  City(10001, "my city1", "MYC", "My Town", 100000),
  City(10002, "my city2", "MYC", "My Village", 120000),
  City(10003, "my city3", "MYC", "My Borough", 140000)
)
val insertCities = quote {
  liftQuery(cities).foreach(e=>query[City].insert(e))
}
ctx.transaction {
  ctx.run(insertCities)
  throw new Exception("transaction failed!")
}
```

Dynamic query (1)

- Quotation 指定 type

```
val query: Quoted[EntityQuery[City]] = quote {
 query[City].filter(c=>c.countryCode=="TWN")
}  // Dynamic query
```

- Quill 在 compile time 階段無法 generate sql

Dynamic query (2)

- 使用 Dyanmic query API

```
val q = dynamicQuery[City].filter { c: Quoted[City] =>
  c.countryCode == "TWN"
}
```

- 由 Quotation 轉換成 dynamic query

```
val queryCity = quote {
  query[City]
}
val q = queryCity.dynamic.filter { c: Quoted[City] =>
  quote { c.countryCode == "TWN" }
}
```

More

Infix - 無奈需要寫 SQL 時

- 使用 String Interpolator: `infix`
- 完整的 SQL
  - Map to case class / tuples
- 部份的 SQL
  - Quill Query 串接 SQL string
  - 呼叫 Database 提供的 funciton

Null handling (1)

```
CREATE TABLE country (
   code character(3) NOT NULL,
   :  :  :
   indepyear smallint,     // nullable
   population integer NOT NULL,
   lifeexpectancy real,     // nullable
   :  :  :
```

```
case class Country(code: String,
   :  :  :
   indepYear: Option[Int],
   population: Int,
   lifeExpectancy: Option[Double],
   :  :  :
```

Null handling (2)

```
query[Country].filter(c=> c.indepYear.isDefined)
```

```
SELECT c.code, c.name, ...
FROM country c
WHERE c.indepyear IS NOT NULL
```

```
query[Country]
     .map{c=> (c.name, c.indepYear.map(y=>s"Independent: $y"))
```

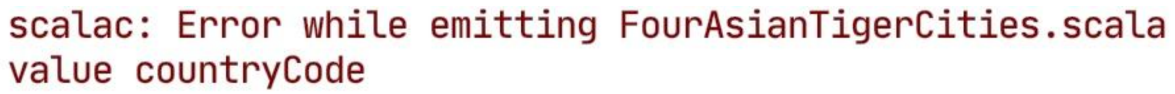
```
SELECT c.name, 'Independent: ' || c.indepyear
FROM country c
```

Query probing - 驗證 SQL 是否正確

- 在 Compile time 驗證 SQL 是否正確
- Enable query probing
  - 使用 **`QueryProbing`** trait
  - Context 須要先在另一個獨立的 project 被編譯
  - Sbt configuration

Some issues

- Query 太多導致 compile-time 變慢
- Cannot write generic function with Scala
- 在 IDE 開發使用 Query probing 可能導致 Too many clients
  - Caused by: com.zaxxer.hikari.pool.HikariPool$PoolInitializationException: Failed to initialize pool: FATAL: sorry, too many clients already
- 有些難懂的 compile error



```
scalac: Error while emitting FourAsianTigerCities.scala
value countryCode
```

Q & A

Thank you !!