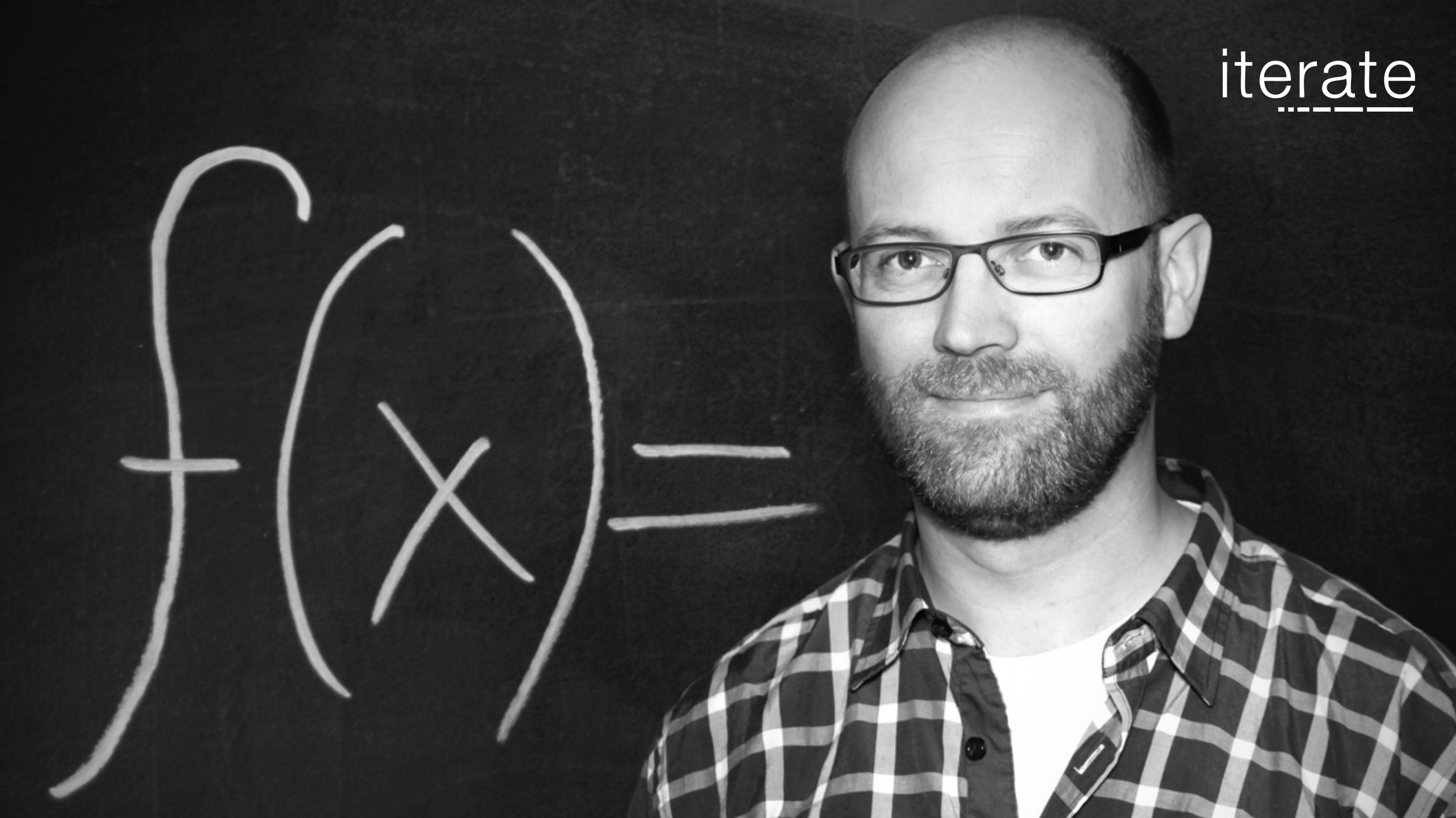
Databaseintegrasjon i Scala

Fredrik Vraalsen
vraalsen@iterate.no / @fredriv

Booster 2016


f(x)=
iterate

Exercises

- git clone http://github.com/fredriv/slick-ws.git
  - or download https://github.com/fredriv/slick-ws/archive/master.zip
- Maven: `mvn test`
- SBT: `sbt/bin/sbt`

```
> ~ test-quick
```

Why not just use …?

MyBatis

- Simple to get started

```
FROM  Message
WHERE storedStatus = ‘distributed’
AND   replacedByANewVersion = false
AND   validToDate > :now
```


We are not amused
© Fredrik Vraalsen 2012

- Functional Relational Mapping (FRM)
- Part of the ~~Typesafe~~ Lightbend stack
- Slick 3.0 aka “Reactive Slick” released last year

- Pure Scala
- Type safe
- Composable

Schemas

```
class Employees(tag: Tag)
extends Table[(Int, String, Int)](tag, "EMPLOYEES") {




}
```

Schemas

```
class Employees(tag: Tag)
extends Table[(Int, String, Int)](tag, "EMPLOYEES") {

  def id     = column[Int]    (“EMPLOYEE_ID", O.PrimaryKey)
  def name   = column[String] ("NAME")
  def salary = column[Int]    ("SALARY")


}
```

Schemas

```
class Employees(tag: Tag)
extends Table[(Int, String, Int)](tag, "EMPLOYEES") {

  def id     = column[Int]    (“EMPLOYEE_ID", O.PrimaryKey)
  def name   = column[String] ("NAME")
  def salary = column[Int]    ("SALARY")

  def * = (id, name, salary)
}
```

Schemas

```
class Employees(tag: Tag)
extends Table[(Int, String, Int)](tag, "EMPLOYEES") {

  def id     = column[Int]    (“EMPLOYEE_ID", O.PrimaryKey)
  def name   = column[String] ("NAME")
  def salary = column[Int]    ("SALARY")

  def * = (id, name, salary)
}

val employees = TableQuery[Employees]
```

Queries

```
employees
```

Queries

```
// SELECT * FROM employees
employees
```

Queries

```
// SELECT * FROM employees
employees

for {
  emp <- employees
} yield emp
```

Queries

```
// SELECT name FROM employees
employees.map(_.name)

for {
  emp <- employees
} yield emp.name
```

Queries

```
// SELECT name, salary FROM employees
employees.map(emp => (emp.name, emp.salary))

for {
  emp <- employees
} yield (emp.name, emp.salary)
```

Queries

```
employees.filter(_.salary > 600000)
```

Queries

```
// SELECT * FROM employees WHERE salary > 600000
employees.filter(_.salary > 600000)
```

Queries

```
// SELECT * FROM employees WHERE salary > 600000
employees.filter(_.salary > 600000)


for {
  emp <- employees if emp.salary > 600000
} yield emp
```

Queries

```
// SELECT name FROM employees WHERE salary > 600000
employees.filter(_.salary > 600000)
         .map(_.name)

for {
  emp <- employees if emp.salary > 600000
} yield emp.name
```

Queries

```
// SELECT name, salary FROM employees WHERE salary > 600000
employees.filter(_.salary > 600000)
         .map(emp => (emp.name, emp.salary))

for {
  emp <- employees if emp.salary > 600000
} yield (emp.name, emp.salary)
```

Getting results

```
val names =
        employees.map(_.name)
```

Getting results

```
val names =
        employees.map(_.name).result
```

Getting results

```
val names =
  db.run(employees.map(_.name).result)
```

Type safe

```
val names: Future[Seq[String]] =
  db.run(employees.map(_.name).result)
```

Type safe

```
// Does not compile!!
employees.filter(_.salary > "600000")
```

Composable

```
val highEarners = employees.filter(_.salary > 600000)
```

Composable

```
val highEarners = employees.filter(_.salary > 600000)

val topEarners =
```

Composable

```
val highEarners = employees.filter(_.salary > 600000)

val topEarners = highEarners.sortBy(_.salary.desc)
```

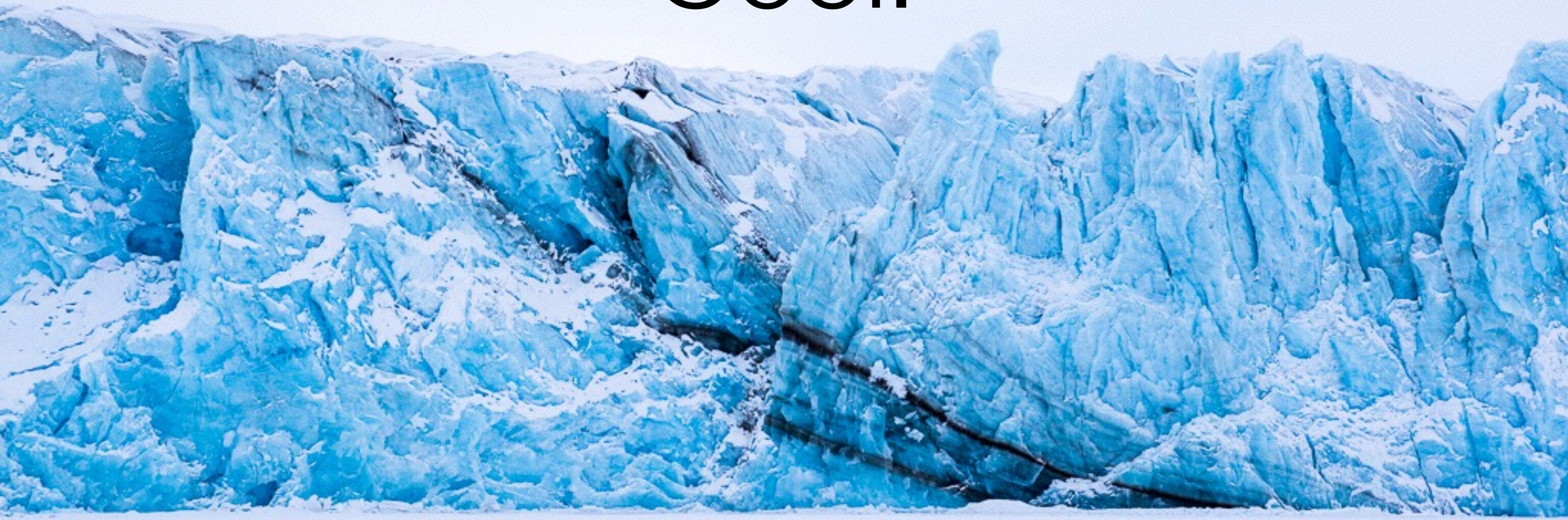
Composable

```
val highEarners = employees.filter(_.salary > 600000)

val topEarners = highEarners.sortBy(_.salary.desc)
                            .map(_.name)
```

Composable

```
val highEarners = employees.filter(_.salary > 600000)

val topEarners = highEarners.sortBy(_.salary.desc)
                            .map(_.name)
                            .take(10)
```

Cool!



Now onwards...

Domain model

```
case class Room(id: RoomId, name: String)
```

Domain model

```
case class Room(id: RoomId, name: String)

case class Sensor(id: SensorId,
                  roomId: RoomId,
                  sensorType: String)
```

Domain model

```
case class Room(id: RoomId, name: String)

case class Sensor(id: SensorId,
                  roomId: RoomId,
                  sensorType: String)

case class Reading(id: Option[Int],
                   sensorId: SensorId,
                   timestamp: LocalDateTime,
                   value: BigDecimal)
```

Mapped tables

```
case class Room(id: RoomId, name: String)
```

Mapped tables

```
case class Room(id: RoomId, name: String)

class Rooms(tag: Tag) extends Table[Room](tag, "ROOMS") {



}
```

Mapped tables

```
case class Room(id: RoomId, name: String)

class Rooms(tag: Tag) extends Table[Room](tag, "ROOMS") {
  def id   = column[RoomId] ("ID", O.PrimaryKey)
  def name = column[String] ("NAME")


}
```

Mapped tables

```
case class Room(id: RoomId, name: String)

class Rooms(tag: Tag) extends Table[Room](tag, "ROOMS") {
  def id   = column[RoomId] ("ID", O.PrimaryKey)
  def name = column[String] ("NAME")

  def * = (id, name) <> (Room.tupled, Room.unapply)
}
```

Mapped tables

```
case class Room(id: RoomId, name: String)

class Rooms(tag: Tag) extends Table[Room](tag, "ROOMS") {
  def id   = column[RoomId] ("ID", O.PrimaryKey)
  def name = column[String] ("NAME")

  def * = (id, name) <> (Room.tupled, Room.unapply)
}

val rooms = TableQuery[Rooms]
```

Joins

```
def roomsWithSensor(sensorType: String)
                   (implicit db: Database): Future[Seq[Room]] =
{









}
```

Joins

```
def roomsWithSensor(sensorType: String)
                   (implicit db: Database): Future[Seq[Room]] =
{
  val q = for {
    sensor <- sensors if sensor.sensorType === sensorType




}
```

Joins

```
def roomsWithSensor(sensorType: String)
                   (implicit db: Database): Future[Seq[Room]] =
{
  val q = for {
    sensor <- sensors if sensor.sensorType === sensorType
    room   <- rooms   if room.id === sensor.roomId



}
```

Joins

```
def roomsWithSensor(sensorType: String)
                   (implicit db: Database): Future[Seq[Room]] =
{
  val q = for {
    sensor <- sensors if sensor.sensorType === sensorType
    room   <- rooms   if room.id === sensor.roomId
  } yield room


}
```

Joins

```
def roomsWithSensor(sensorType: String)
                   (implicit db: Database): Future[Seq[Room]] =
{
  val q = for {
    sensor <- sensors if sensor.sensorType === sensorType
    room   <- rooms   if room.id === sensor.roomId
  } yield room

  db.run(q.result)
}
```

Foreign keys

```
def roomsWithSensor(sensorType: String)
                   (implicit db: Database): Future[Seq[Room]] =
{
  val q = for {
    sensor <- sensors if sensor.sensorType === sensorType
    room   <- sensor.room
  } yield room

  db.run(q.result)
}
```

Foreign keys

```
class Sensors(tag: Tag) extends Table[Sensor](tag, "SENSORS") {
  def id          = column[SensorId] ("ID", O.PrimaryKey)
  def roomId      = column[RoomId]   ("ROOM_ID")
  def sensorType = column[String]   ("SENSOR_TYPE")

  def * = (id, roomId, sensorType) <> (Sensor.tupled,
                                       Sensor.unapply)


}
```

Foreign keys

```
class Sensors(tag: Tag) extends Table[Sensor](tag, "SENSORS") {
  def id          = column[SensorId] ("ID", O.PrimaryKey)
  def roomId      = column[RoomId]   ("ROOM_ID")
  def sensorType = column[String]   ("SENSOR_TYPE")

  def * = (id, roomId, sensorType) <> (Sensor.tupled,
                                       Sensor.unapply)

  def room = foreignKey("ROOM_FK", roomId, rooms)(_.id)
}
```

CRUD!

```
val q = rooms

// Read all rows
q.result
```

CRUD!

```
val q = rooms.filter(_.id === 1)

// Read single row
q.result.headOption
```

CRUD!

```
val q = rooms.filter(_.id === 1)

// Update
q.map(_.name).update(“Living room”)
```

CRUD!

```
val q = rooms.filter(_.id === 1)

// Delete
q.delete
```

CRUD!

```
// Create
rooms += Room(RoomId(1), "Living room”)
```

Exercises

- git clone http://github.com/fredriv/slick-ws.git
  - or download https://github.com/fredriv/slick-ws/archive/master.zip
- Maven: `mvn test`
- SBT: `sbt/bin/sbt`

```
> ~ test-quick
```

Performance


Questions?
vraalsen@iterate.no / @fredriv
© Fredrik Vraalsen 2012

iterate