Opaleye
Haskell Embedded Relational Query Language

Tom Ellis

10th January 2013

Tables

```
personTable :: Query (Wire PersonId, Wire String,
                      Wire CityId)

runQueryDefault personTable :: [(PersonId, String,
                                 CityId)]

SELECT person_id, person_name, person_city_id
FROM person_table

[ (PersonId 1, "Tom",     CityId 1),
  (PersonId 2, "Simon",   CityId 1),
  (PersonId 3, "Duncan",  CityId 2),
  (PersonId 4, "Clemens", CityId 3) ]
```

A query

```
countries :: Query (Wire String)
countries = proc () -> do
  (_, personName, personCityId) <- personTable  -< ()
  (cityId, _, cityCountryId)    <- cityTable    -< ()
  (countryId, _, countryCode)   <- countryTable -< ()

  restrict <<< eq -< (personCityId, cityId)
  restrict <<< eq -< (cityCountryId, countryName)

  lives_in_string <- constant " lives in " -< ()
  cat3 -< (personName, lives_in_string, countryCode)
```

A query

```
SELECT person_name || ' lives in ' || country_name
FROM   person_table, city_table, country_table
WHERE  person_city_id = city_id
AND    city_country_id = country_id
```

Type safety

Errors prevented at compile time

- Comparing an (integer) ID and a string

```
...
restrict <<< eq -< (cityId, personName)
...
```

- Comparing two incompatible IDs even though they are both integers in the database

```
...
restrict <<< eq -< (cityId, personId)
...
```

Composability

```
livesIn :: QueryArr (Wire String, Wire String) (Wire String)
livesIn = proc (personName, countryName) -> do
    lives_in_string <- constant " lives in " -< ()
    cat3 -< (personName, lives_in_string, countryName)

countryOfCity :: QueryArr (Wire CityId) (Wire CountryId)
countryOfCity = proc cityId -> do
    (cityId', _, cityCountryId)   <- countryTable -< ()
    restrict <<< eq -< (cityId, cityId')
    returnA -< cityCountryId

countryNameOfCountry :: QueryArr (Wire CountryId) (Wire String)
...

countryNameOfCity :: QueryArr (Wire CityId) (Wire String)
countryNameOfCity = countryNameOfCountry <<< countryOfCity
```

Composability

```
countries :: Query (Wire String)
countries = proc () -> do
  (_, personName, personCityId) <- personTable  -< ()
  countryName <- countryNameOfCity -< personCityId
  livesIn -< (personName, countryName)
```

```
[ "Tom lives in UK",
  "Simon lives in UK",
  "Duncan lives in UK",
  "Clemens lives in NL" ]
```

Summary

Opaleye is an approach to relational queries which is

- type safe
- composable

I didn't have time to talk about

- aggregation is type safe and composable
- this is why we use Arrows rather than Monads
- both HaskellDB and Esqueleto get this wrong