Michal Balda

Querier:
simple relational database access

What is Querier?

- A library for querying relational databases.
- Focused on simplicity.
- Heavily inspired by NotORM (http://www.notorm.com/).

Why?

Accessing a relational database in Pharo:

Why?

Accessing a relational database in Pharo:

1) Write SQL by hand.

Why?

Accessing a relational database in Pharo:

1) Write SQL by hand.

2) Use an object-relational mapper (GLORP).

Why?

Accessing a relational database in Pharo:

1) Write SQL by hand.
1.5) Use Querier.
2) Use an object-relational mapper (GLORP).

Database structure

Primary Key Column	id
Foreign Key Column	{table}_id
Table Name	{table}

Setup

```
| driver structure db |

driver := "...".
structure := QRRConventionalStructure new.
db := Querier withDriver: driver structure: structure
```

Accessing a table

```
db table: #song
"or use a shortcut:"
db song
```

song	
id	Integer, Primary Key
title	Varchar
length	Integer
album_id	Integer, Foreign Key

Accessing a table

```
db table: #song
"or use a shortcut:"
db song

    SELECT *
    FROM song
```

Accessing a table

```
db song do: [ :row |
        Transcript show: row title; cr ]
```

Two principles

1) A table is a collection of rows.

2) A row is a dictionary of values.

WHERE

```
db song select: [ :row |
        (row length >= 180)
        & (row length <= 300) ]
```

```
SELECT *
FROM song
WHERE length >= 180
AND length <= 300
```

WHERE

```
db song select: [ :row |
        (row length >= 180)
        & (row length <= 300) ]
```

Udo Schneider: Block Translators - parsing magic
http://readthesourceluke.blogspot.com/2014/09/block-translators-parsing-magic.html

ORDER BY

```
db song sorted: [ :a :b |
        a length < b length ]
```

```
SELECT *
FROM song
ORDER BY length ASC
```

LIMIT and OFFSET

```
(db song sorted: [ :a :b |
        a length < b length ])
            first: 10
```

```
SELECT *
FROM song
ORDER BY length ASC
LIMIT 10
```

LIMIT and OFFSET

```
(db song sorted: [ :a :b |
        a length < b length ])
            allButFirst: 10
```

```
SELECT *
FROM song
ORDER BY length ASC
OFFSET 10
```

Selecting a single row

```
row := db song detect: [ :each |
        each id = 123 ]
```

```
SELECT *
FROM song
WHERE id = 123
LIMIT 1
```

Selecting by primary key

```
row := db song at: 123
```

```
SELECT *
FROM song
WHERE id = 123
LIMIT 1
```

Selecting by primary key

```
row := db song at: 123
```

```
SELECT *
FROM song
WHERE id = 123
LIMIT 1
```

Aggregations

```
db song average: [ :row |
        row length ]

db song average: #length
```

```
SELECT AVG(length)
FROM song
```

Enumerating the result

```
db song collect: [ :row |
        row title ]

db song do: [ :row |
        Transcript show: row title; cr ]

db song size
```

Accessing related tables

album	
id	Integer, Primary Key
name	Varchar

song	
id	Integer, Primary Key
title	Varchar
length	Integer
album_id	Integer, Foreign Key

Accessing related tables

```
db song select: [ :row |
        row album name = 'Unknown Album' ]
```

album	
id	Integer, Primary Key
name	Varchar

song	
id	Integer, Primary Key
title	Varchar
length	Integer
album_id	Integer, Foreign Key

Accessing related tables

```
db song select: [ :row |
      row album name = 'Unknown Album' ]
```

```
SELECT *
FROM song
LEFT JOIN album
  ON song.album_id = album.id
WHERE album.name = 'Unknown Album'
```

Accessing related tables

```
db song select: [ :row |
      row album name = 'Unknown Album' ]
```

```
SELECT *
FROM song
LEFT JOIN album
  ON song.album_id = album.id
WHERE album.name = 'Unknown Album'
```

Accessing related tables

```
db song select: [ :row |
      row album artist name = 'Unknown Artist' ]
```

```
SELECT *
FROM song
LEFT JOIN album
  ON song.album_id = album.id
LEFT JOIN artist
  ON album.artist_id = artist.id
WHERE artist.name = 'Unknown Artist'
```

Accessing related tables

```
db song do: [ :row |
        Transcript show: row album name ]
```

Accessing related tables

```
db song do: [ :row |
        Transcript show: row album name ]
```

1) SELECT *
   FROM song

Accessing related tables

```
db song do: [ :row |
        Transcript show: row album name ]
```

1) SELECT *
   FROM song

Accessing related tables

```
db song do: [ :row |
        Transcript show: row album name ]
```

1) SELECT *
   FROM song

2) SELECT *
   FROM album
   WHERE id IN (1, 2, 3, ...)

Accessing related tables

album	
id	Integer, Primary Key
name	Varchar

song	
id	Integer, Primary Key
title	Varchar
length	Integer
album_id	Integer, Foreign Key

The opposite direction

```
db album do: [ :row |
        row songCollection do: [ :song |
            Transcript show: song name; cr ] ]
```

album	
id	Integer, Primary Key
name	Varchar

song	
id	Integer, Primary Key
title	Varchar
length	Integer
album_id	Integer, Foreign Key

The opposite direction

```
db album do: [ :row |
        row songCollection do: [ :song |
            Transcript show: song name; cr ] ]
```

1) SELECT *
   FROM album

The opposite direction

```
db album do: [ :row |
        row songCollection do: [ :song |
            Transcript show: song name; cr ] ]
```

1) SELECT *
   FROM album

2) SELECT *
   FROM song
   WHERE album_id IN (1, 2, 3, ...)

UPDATE

```
db song do: [ :row |
        row length: row length + 10.
        row save ]
```

1) SELECT * FROM song
2) UPDATE song SET length = 325
   WHERE id = 1
3) UPDATE song SET length = 648
   WHERE id = 2
4) … *and many more*

Better UPDATE

```
db song update: [ :row |
        row length: row length + 10 ]
```

```
UPDATE song
SET length = length + 10
```

Better UPDATE

```
(db song select: [ :row |
    row length < 180 ])
        update: [ :row |
            row length: row length + 10 ]
```

```
UPDATE song
SET length = length + 10
WHERE length < 180
```

DELETE

```
(db song select: [ :row |
    row length < 180 ])
        removeAll

db song delete: [ :row |
    row length < 180 ]
```

```
DELETE FROM song
WHERE length < 180
```

INSERT

```
| row |
row := db song new.
row title: 'New Song'.
row length: 316.
row album: (db album detect: [ :album |
        album name = 'Unknown Album' ]).
row save.

Transcript show: row id
```

Current Status

- A proof-of-concept for Pharo + Postgres.
- Working on polishing all features + querying other RDBMS through Garage (https://guillep.github.io/DBXTalk/garage/).

Future work

- Add ORM-like features (instantiate your entity classes instead of dictionaries).
- Add at least partial support for non-relational databases (like MongoDB).

Questions?

http://querier.xmb.cz/

Thank you for your attention!

http://querier.xmb.cz/