

SQL is a must when accessing relational database

Compatibility: SQL is the standard language to access relational DBMS.

Speed: Relational queries in SQL can be optimized very aggressively by DBMS.

Expressivity: Simpler paradigms do not allow complex access patterns.

The Old Way: Oil and Water and SQL

Oil and water don't mix, that is a well-known fact of physics. This is also true for SQL and its various host languages. SQL usually is embedded in the host language as strings — in the same container but not mixing. The neat features of the host language, such as type checking, are not available to SQL.

```
query = "select * from persons";
statement.executeQuery(query);
```

JDBC

ODBC

Embedded SQL

The New Way: For-comprehensions

A very simple construct found in modern programming languages is equivalent to a SQL relational query: for-comprehensions. This is equivalent to the JDBC example next door:

```
for (p <- persons) ...
```

It also works for conditions and joins:

```
for (p <- persons; p.age > 18) ...
for (p <- persons;
     s <- students;
     p.name = s.name) ...
```

What about relational-object mapping?

The strength of relational databases is the capability to create new relations by applying operators to existing relations.

Consider the tables "products (pid, pname)", "suppliers (sid, sname)" and "order (pid, sid, quantity)". With a relational-object mapping, finding all supplier's names for a given product name will require multiple queries. In SQL it is "select * from products natural join suppliers natural join order". Simple enough!

But SQL is a must, isn't it?

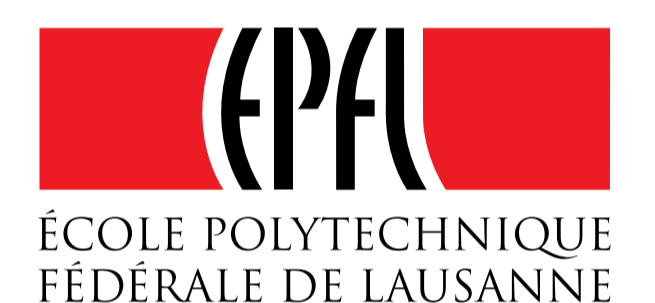
Yes, but developers don't need to know

For-comprehensions can be compiled into equivalent SQL statements (query shipping). The developer gets a better way to access relational data, the type-checker gets something to chew on, the DBMS gets SQL ... everyone is happy!

Better Relational Queries using For-comprehensions

Gilles Dubochet, Programming Methods Laboratory (LAMP1 IIF IC EPFL);

part of this work done at the LFCS (University of Edinburgh) under the supervision of P. Wadler.



Previous related work

Kleisli (by L. Wong) is a querying system using optimized for-comprehensions. It is functional and data is represented using records and variants. It was developed as a way to solve large database integration problems, particularly for bio-computing.

What we have: SLinks

Based on Kleisli but new, it improves on the data-structures and type-checking. Records and variants are extensible which allows SLinks to be used in more situations than only for querying. This is a first step in the direction of using for-comprehensions in a general-purpose programming language.

What we do: ScalaDALL

An extension of the Scala language that adds for-comprehension querying. By mixing a rich querying paradigm with a modern general-purpose programming language, we hope to significantly simplify the integration of data-access and calculation.

Accessing database tables

```
table "t" from db
```

A table is always read entirely. Filtering is done with comprehensions.

```
table "t" with {#name:string, #age:int}
from db
```

The type of tables (a named record) can be declared: the program is type-safe and

the table type is validated at run-time. SQL tables are bags by default, but can be coerced to sets (no duplicates) or lists:

```
table "t" unique          set
table "t" order [#name]  list
```

Comprehensions

```
[ x | x <- set aSet, x > 4 ]
```

Syntax uses no "for", but meaning is the same: "x" is the body that is evaluated for

every element, "x < set aSet" is a generator that binds "x" successively to each element of aSet, "x > 4" is a filter.

```
[ {x,y} | x <- set [set 1 2],
      y <- set [set 3 4] ]
```

Multiple generators are equivalent to joins. The result for the last example is:

```
[set {1,3} {1,4} {2,3} {2,4}]
```

Shipping comprehensions

Requires to extract the largest SQL query possible. If too small, the DBMS's optimizer won't be able to rearrange much. Large ones are difficult to extract because of the

limited expressiveness of SQL. For example: the following expression ...

```
[ ... | x <- set table "X" from db,
      y <- set table "Y" from db ]
```

... becomes after shipping:

```
for (select * from X) do {
  for (select * from Y) do {...}
```

```
for (select * from X join Y) do {...}
```