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.

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.

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!

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:

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

It also works for conditions and joins:

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

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.

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

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 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 requires 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?

SQL is a must when accessing relational database

<table>
<thead>
<tr>
<th>Compatibility: SQL is the standard language to access relational DBMS.</th>
<th>Speed: Relational queries in SQL can be optimized very aggressively by DBMS.</th>
<th>Expressivity: Simpler paradigms do not allow complex access patterns.</th>
</tr>
</thead>
</table>

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:

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

It also works for conditions and joins:

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

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.

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

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 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 requires 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?

SQL is a must when accessing relational database

<table>
<thead>
<tr>
<th>Compatibility: SQL is the standard language to access relational DBMS.</th>
<th>Speed: Relational queries in SQL can be optimized very aggressively by DBMS.</th>
<th>Expressivity: Simpler paradigms do not allow complex access patterns.</th>
</tr>
</thead>
</table>

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:

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

It also works for conditions and joins:

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

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.

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

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 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 requires 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?

SQL is a must when accessing relational database

<table>
<thead>
<tr>
<th>Compatibility: SQL is the standard language to access relational DBMS.</th>
<th>Speed: Relational queries in SQL can be optimized very aggressively by DBMS.</th>
<th>Expressivity: Simpler paradigms do not allow complex access patterns.</th>
</tr>
</thead>
</table>

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:

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

It also works for conditions and joins:

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

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.

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

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 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 requires 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?

SQL is a must when accessing relational database

<table>
<thead>
<tr>
<th>Compatibility: SQL is the standard language to access relational DBMS.</th>
<th>Speed: Relational queries in SQL can be optimized very aggressively by DBMS.</th>
<th>Expressivity: Simpler paradigms do not allow complex access patterns.</th>
</tr>
</thead>
</table>

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:

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

It also works for conditions and joins:

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

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.

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