Why ScalaQL

- Impedance mismatch between the relational model and the paradigm of PL (Scala).
- Embedded SQL statements are error-prone
- Cannot be checked at compile time

For-comprehension

- for-comprehensions are a natural syntactic device for representing declarative queries
- Example:

```
val underAge = for {
  p <- Person
  c <- Company

  if p.company is c
  if p.age < 14
} yield p
```

SQL correspondence

```
SELECT p.* FROM people p JOIN companies c ON p.company_id = c.id
  WHERE p.age < 14
```

For-comprehension

- Each for-comprehension is translated into series of calls on ***Map***, ***flatMap*** and ***filter*** methods.
- Objects that for-comprehension targets on should implement those methods.

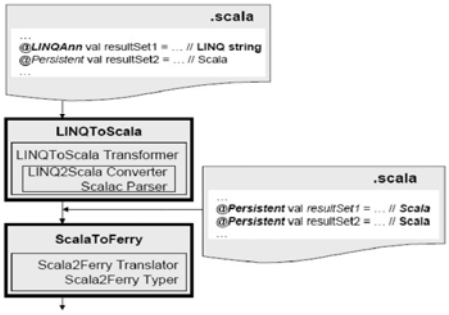
Language Features

- Deferred evaluation
  - Not evaluate until the query is used as sequence
  - Represented by AST
- Generate items of new data type in the fly
  - Query[T] where T could be dynamical generated by ***new {...}***
  - Type inference

Compilation

- Annotation
- LINQ to Scala
- Scala to Ferry

Architecture



Annotation

- ScaleQL recognizes two types of queries
  - Embedded LINQ queries with an @LINQAnn annotation
  - Scala queries, tagged with @Persistent annotation

Annotation

```
// LINQ embedded query
@LINQAnn val projectsByName = "from empl in Employees select " +
                              "   new { name = empl.name," +
                              "         listOfProjects = from proj
                              "                          where pro
                              "                          select pr


val salaries = List(200, 600, 400, 300)
// Scala query from scratch
@Persistent val namesWithSalaries = ( ( for (empl @ Employee(_, n
                                              if dept == "GE")
                                     yield name ) take 4 ) zip sal
```

LINQ to Scala

- Convert LINQ query to string with annotations
- Parse the string and generate the Scala for-comprehensions
- Tag with @Persistent

LINQ to Scala

```
"from empl in Employees select " +
      "new { name = empl.name," +
            "listOfProjects = from project in projects" +
                              "where project.employees.contains(empl.name)" +
                              "select project.name }"
```

is converted to

```
"for (empl <- Employees)" +
"yield new { val name = empl.name;" +
            "val listOfProjects = for (project <- projects;" +
                                  "if (project.employees.contains(empl.name)))" +
                              "yield project.name }"
```

Scala to Ferry

- Nested Scala tuple → flattened Ferry tuple
- Type checking
- AST representing Scala query will used for translation

Scala to Ferry

```
@Persistent val projectsByName = for (empl <- Employees)
      yield new { val name = empl.name;
                  val listOfProjects = for (project <- projects;
                                            if (project.employees.contains(empl.name)))
                                       yield project.name }
```

```
for empl in table Employees (id int, name string, dept string, salary int) with keys ((id))
return (name empl.name,
      listOfProjects for project in [(name 'project1', employees ['Alex', 'Bert', 'Cora', 'Drew', 'Erik']),
                                     (name 'project2', employees ['Fred', 'Gina', 'Herb', 'Ivan', 'Jill'])]
                  where length(filter(v → v == empl.name, project.employees)) != 0
                  return project.name)
```