Leveraging Synergy Between Database and Programming Language Courses

Brian Howard
DePauw University

This work was supported by the 2008–11 Donald E. Town Faculty Fellowship from DePauw University.

Motivation and Overview

- Enhance DB or PL course by building on connections with the other
- Examples:
  - Syntax-Directed SQL Translation
  - Comprehension Syntax
  - Object-Relational Mapping
  - Transactional Memory
  - Document-Oriented Databases
  - MapReduce

Syntax-Directed SQL Translation

Grammar for a subset of SQL

$$
\begin{array}{rcl}
Expr & ::= & \texttt{SELECT} * FWGH \\
 & | & \texttt{SELECT}\ (ColName \mid Agg)^{+}_{,}\ FWGH \\
FWGH & ::= & FWG\ (\texttt{HAVING}\ Cond)^{?} \\
FWG & ::= & FW\ (\texttt{GROUP BY}\ ColName^{+}_{,})^{?} \\
FW & ::= & F\ (\texttt{WHERE}\ Cond)^{?} \\
F & ::= & \texttt{FROM}\ TableName^{+}_{,}
\end{array}
$$

Translation Functions

$$
\begin{aligned}
\mathcal{E}[\![\texttt{SELECT} * FWGH]\!] &= \mathcal{H}[\![FWGH, \emptyset]\!] \\
\mathcal{E}[\![\texttt{SELECT} (ColName \mid Agg)^{+}_{,} \; FWGH]\!] &= \mathbf{project}(\mathcal{H}[\![FWGH, \{Agg^{*}_{,}\}]\!], \{(ColName \mid Agg)^{+}_{,}\}) \\
\mathcal{H}[\![FWG, aggs]\!] &= \mathcal{G}[\![FWG, aggs]\!] \\
\mathcal{H}[\![FWG \; \texttt{HAVING} \; Cond, aggs]\!] &= \mathbf{select}(\mathcal{G}[\![FWG, aggs]\!], Cond) \\
\mathcal{G}[\![FW, \emptyset]\!] &= \mathcal{W}[\![FW]\!] \\
\mathcal{G}[\![FW, aggs]\!] &= \mathbf{groupby}(\mathcal{W}[\![FW]\!], \emptyset, aggs) \\
\mathcal{G}[\![FW \; \texttt{GROUP BY} \; ColName^{+}_{,}, aggs]\!] &= \mathbf{groupby}(\mathcal{W}[\![FW]\!], \{ColName^{+}_{,}\}, aggs) \\
\mathcal{W}[\![F]\!] &= \mathcal{F}[\![F]\!] \\
\mathcal{W}[\![F \; \texttt{WHERE} \; Cond]\!] &= \mathbf{select}(\mathcal{F}[\![F]\!], Cond) \\
\mathcal{F}[\![\texttt{FROM} \; TableName]\!] &= TableName \\
\mathcal{F}[\![\texttt{FROM} \; TableName^{+}_{,} \; , \; TableName]\!] &= \mathbf{product}(\mathcal{F}[\![\texttt{FROM} \; TableName^{+}_{,}]\!], TableName)
\end{aligned}
$$

Example Translation

What is the difference between a `HAVING` and a `WHERE` condition when there is no `GROUP BY`?

$$
\begin{aligned}
&\mathcal{E}[\![\texttt{SELECT Min(Year) as Y FROM Student HAVING } \mathit{Cond}]\!] \\
&= \mathbf{project}(\mathcal{H}[\![\texttt{FROM Student HAVING } \mathit{Cond}, \{\texttt{Min(Year) as Y}\}]\!], \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathbf{select}(\mathcal{G}[\![\texttt{FROM Student}, \{\texttt{Min(Year) as Y}\}]\!], \mathit{Cond}), \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathbf{select}(\mathbf{groupby}(\mathcal{W}[\![\texttt{FROM Student}]\!], \emptyset, \{\mathbf{Min(Year)\ as\ Y}\}), \mathit{Cond}), \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathbf{select}(\mathbf{groupby}(\mathcal{F}[\![\texttt{FROM Student}]\!], \emptyset, \{\mathbf{Min(Year)\ as\ Y}\}), \mathit{Cond}), \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathbf{select}(\mathbf{groupby}(\mathbf{Student}, \emptyset, \{\mathbf{Min(Year)\ as\ Y}\}), \mathit{Cond}), \{\mathbf{Y}\})
\end{aligned}
$$

versus

$$
\begin{aligned}
&\mathcal{E}[\![\texttt{SELECT Min(Year) as Y FROM Student WHERE } \mathit{Cond}]\!] \\
&= \mathbf{project}(\mathcal{H}[\![\texttt{FROM Student WHERE } \mathit{Cond}, \{\texttt{Min(Year) as Y}\}]\!], \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathcal{G}[\![\texttt{FROM Student WHERE } \mathit{Cond}, \{\texttt{Min(Year) as Y}\}]\!], \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathbf{groupby}(\mathcal{W}[\![\texttt{FROM Student WHERE } \mathit{Cond}]\!], \emptyset, \{\mathbf{Min(Year)\ as\ Y}\}), \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathbf{groupby}(\mathbf{select}(\mathcal{F}[\![\texttt{FROM Student}]\!], \mathit{Cond}), \emptyset, \{\mathbf{Min(Year)\ as\ Y}\}), \{\mathbf{Y}\}) \\
&= \mathbf{project}(\mathbf{groupby}(\mathbf{select}(\mathbf{Student}, \mathit{Cond}), \emptyset, \{\mathbf{Min(Year)\ as\ Y}\}), \{\mathbf{Y}\})
\end{aligned}
$$

Comprehension Syntax

Generalized `for` loop, based on set builder notation

Scala Example

```
val mentorPairs = for {
  mentor <- students
  other <- students
  if mentor.year < other.year &&
     mentor.major == other.major
} yield (mentor, other)
```

This is equivalent to

```
val mentors = students.flatMap(mentor =>
  students.withFilter(other =>
    mentor.year < other.year &&
    mentor.major == other.major
  ).map(other =>
    (mentor, other)
  )
)
```

C# LINQ Equivalent

```
var mentors =
  from mentor in students
  from other in students
  where mentor.year < other.year
     && mentor.major == other.major
  select new {a = mentor, b = other};
```

SQL Equivalent

```
SELECT mentor.ID as a, other.ID as b
FROM Student mentor, Student other
WHERE mentor.Year < other.Year
  AND mentor.Major = other.Major;
```

Object-Relational Mapping

Java Database Connectivity (JDBC)

```
List mentors = new ArrayList();
Statement statement =
connection.createStatement();
String query =
  "SELECT mentor.ID as a, other.ID as b " +
  "FROM Student mentor, Student other " +
  "WHERE mentor.Year < other.Year " +
  "  AND mentor.Major = other.Major;";

ResultSet results =
statement.executeQuery(query);
while (results.next()) {
  String mentorID = results.getString("a");
  String otherID = results.getString("b");
  mentors.add(new MIDPair(mentorID,
otherID));
}
results.close();
```

Java Persistence API (JPA)

```
@Entity
@Table(name="Student")
public class Student {
  @Id @Column(name="ID")
  private String id; // Primary key

  @Column(name="Year")
  private int year;

  @ManyToOne @JoinColumn(name="Major")
  private Department major; // Foreign key

  // usual constructors, accessors, etc. go
here
}
```

Java Persistence Query Language (JPQL)

```
List mentors = new ArrayList();
String queryString =
  "select mentor, other " +
  "from Student mentor, Student other " +
  "where mentor.year < other.year " +
  "  and mentor.major = other.major";

Query query =
entityMgr.createQuery(queryString);
for (Object result : query.getResultList()) {
  Object[] pair = (Object[]) result;
  Student mentor = (Student) pair[0];
  Student other = (Student) pair[1];
  mentors.add(new MPair(mentor, other));
}
```

C# LINQ to Entities

```
var context = ...;
var query =
  from mentor in context.students
  from other in context.students
  where mentor.year < other.year
     && mentor.major == other.major
  select new {a = mentor, b = other};
var mentors = query.ToList();
```

Transactional Memory

```
class Fork { val inUse = Ref(false) }

def meal(left: Fork, right: Fork) {
  // thinking

  atomic { implicit txn =>
    if (left.inUse() || right.inUse())
      retry // forks are not both ready, wait
    left.inUse() = true
    right.inUse() = true
  }

  // eating

  atomic { implicit txn =>
    left.inUse() = false
    right.inUse() = false
  }
}
```

Example from ScalaSTM library documentation

Document-Oriented Databases

JavaScript Object Notation (JSON)

```
{
  "ID": "12-34567",
  "Name": "Ann O'Nemus",
  "Year": 2015,
  "Major": "Computer Science",
  "Home Address": {
    "Street": "123 Main",
    "City": "Springfield",
    "State": "AK",
    "ZIP": 98765
  },
  "Phones": [
    {"Type": "Home", "Number":
"555-555-1234"},
    {"Type": "Cell", "Number":
"555-555-5678"}
  ]
}
```

MapReduce

Example in MongoDB: count number of students per major/year

```
var map = function() {
  emit({"Major": this.Major,
        "Year": this.Year}, 1)
}

var reduce = function(key, values) {
  var total = 0;
  for (index in values) total +=
values[index];
  return total;
}

db.runCommand({
  "mapreduce": "students", // source
collection
  "map": map,
  "reduce": reduce,
  "out": "graduates" // output collection
})
```