Advances in Programming Languages
APL9: Using SQL from Java

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 26 October 2010
Semester 1 Week 6
This is the first of three lectures on integrating domain-specific languages with general-purpose programming languages. In particular, SQL for database queries.

- Using SQL from Java
- Bridging Query and Programming Languages
- Heterogeneous Metaprogramming
This is the first of three lectures on integrating domain-specific languages with general-purpose programming languages. In particular, SQL for database queries.

- Using SQL from Java
- Bridging Query and Programming Languages
- Heterogeneous Metaprogramming
- SQL as a domain-specific language
- Injection of HTML, Javascript and SQL
- Frameworks for generating SQL code
SQL is a programming language, with a declarative part:

```sql
select isbn, title, price
from books
where price > 100.00
order by title
```

and an imperative part:

```sql
update books set price = 10.00 where price < 10.00
drop table sales
```

as well as numerous extensions, such as procedures and transactions.

SQL is a *domain-specific language*, rather than a general-purpose programming language.
Who Writes SQL?

SQL is one of the world’s most widely used programming languages, but programs in SQL come from many sources. For example:

- Hand-written by a programmer
- Generated by some interactive visual tool
- Generated by an application to fetch an answer for a user
- Generated by one program to request information from another

Most SQL is written by programs, not directly by programmers.

The same is true of HTML, another domain-specific language.

Also XML, Postscript, …
SkyServer Demonstration

http://cas.sdss.org/dr7/en/
Find some stars near a certain spot in the sky

```sql
SELECT top 10
  p.objID,
  p.run, p.rerun, p.camcol, p.field, p.obj,
  p.type, p.ra, p.dec
FROM PhotoTag p, fGetNearbyObjEq(40.433, 0.449, 3) n
WHERE n.objID = p.objID and p.type = 3
```
Sample Queries

— Make these a table with click-through links

SELECT TOP 10
  ’navi.asp?ra=’ + cast(p.ra as varchar(30)) + ‘&dec=’ +
  cast(p.dec as varchar(30)) + ‘”>’ +
  cast(p.objId as varchar(30)) + ’</a>’ as objID,
FROM PhotoTag p, fGetNearbyObjEq(40.433,0.449,3) n
WHERE n.objID=p.objId and p.type=3
Sample Queries

—– Count those stars

```sql
select count(*) from star p
```

—– Planetary query

```sql
select 'Pluto' FROM Planets
```

HTML Injection

The Pluto page is an example of *HTML injection*.

The SkyServer website appears to be serving an incorrect image.

This is used in phishing attacks, and other fraud, where a web server can be cajoled into presenting novel material as its own.

For example, a suitably crafted URL may cause a bank’s web server to present a page that requests account details and then sends them to an attacker’s own site.

Similarly, a comment on a blog may contain code that when read in a web browser causes it to take some unexpected action.

This opportunity to inject HTML or Javascript can arise whenever a web site takes user input and uses that to generate pages. This is known loosely as *cross-site scripting* or *XSS*.
2010-02-09  Google Buzz social communication tool launched
2010-02-09  Google Buzz social communication tool launched
2010-02-16  Cross-site scripting injection attack publicly demonstrated
Google Buzz XSS Hack

2010-02-09  Google Buzz social communication tool launched
2010-02-16  Cross-site scripting injection attack publicly demonstrated
2010-02-17  Google patch bug
Google Buzz XSS Hack

2010-02-09  Google Buzz social communication tool launched
2010-02-16  Cross-site scripting injection attack publicly demonstrated
2010-02-17  Google patch bug

http://www.theregister.co.uk/2010/02/16/google_buzz_security_bug/
http://ha.ckers.org/blog/20100216/google-buzz-security-flaw/
Twitter Javascript injection

2010-09-21 10am UK time

Masato Kinugawa creates "Rainbow Twitter" demonstration.
Magnus Holm creates self-tweeting tweet.
Requires user to mouse over; limits replication to around 60 copies/second, reaching only a few hundred thousand twitter users.
Further worms require no mouse-over, insert further tweets, draw arbitrarily on page, redirect browser...

2010-09-21 3pm UK time

Twitter patch vulnerability (7am their time)

Ian Stark APL9 2010-10-26
2010-09-21 10am UK time

- Masato Kinugawa creates “Rainbow Twitter” demonstration.
2010-09-21 10am UK time

- Masato Kinugawa creates “Rainbow Twitter” demonstration.
- Magnus Holm creates self-tweeting tweet.
Twitter Javascript injection

2010-09-21 10am UK time

- Masato Kinugawa creates “Rainbow Twitter” demonstration.
- Magnus Holm creates self-tweeting tweet.

Requires user to mouse over; limits replication to around 60 copies/second, reaching only a few hundred thousand twitter users.
2010-09-21 10am UK time

- Masato Kinugawa creates “Rainbow Twitter” demonstration.
- Magnus Holm creates self-tweeting tweet.
  Requires user to mouse over; limits replication to around 60 copies/second, reaching only a few hundred thousand twitter users.
- Further worms require no mouse-over, insert further tweets, draw arbitrarily on page, redirect browser...
Twitter Javascript injection

2010-09-21 10am UK time

- Masato Kinugawa creates “Rainbow Twitter” demonstration.
- Magnus Holm creates self-tweeting tweet.
  Requires user to mouse over; limits replication to around 60 copies/second, reaching only a few hundred thousand twitter users.
- Further worms require no mouse-over, insert further tweets, draw arbitrarily on page, redirect browser...

2010-09-21 3pm UK time

- Twitter patch vulnerability (7am their time)
Twitter Javascript injection

2010-09-21 10am UK time

- Masato Kinugawa creates “Rainbow Twitter” demonstration.
- Magnus Holm creates self-tweeting tweet.
  Requires user to mouse over; limits replication to around 60 copies/second, reaching only a few hundred thousand twitter users.
- Further worms require no mouse-over, insert further tweets, draw arbitrarily on page, redirect browser...

2010-09-21 3pm UK time

- Twitter patch vulnerability (7am their time)

http://www.guardian.co.uk/technology/blog/2010/sep/21/twitter-hack-explained-xss-javascript
HTML injection causes a server to deliver a surprising web page. 

\textit{SQL injection} can cause a database server to carry out unexpected actions on the database.
HTML injection causes a server to deliver a surprising web page.

*SQL injection* can cause a database server to carry out unexpected actions on the database. For example, where a server contains code like this:

```sql
select id, email, password
from users
where email = 'bob@example.com'
```
HTML injection causes a server to deliver a surprising web page.

*SQL injection* can cause a database server to carry out unexpected actions on the database. For example, where a server contains code like this:

```sql
select id, email, password
from users
where email = 'bob@example.com'
```

we might supply the unusual email address “x’ or 1=1 --”
SQL Injection

HTML injection causes a server to deliver a surprising web page.

*SQL injection* can cause a database server to carry out unexpected actions on the database. For example, where a server contains code like this:

```sql
select id, email, password
from users
where email = 'bob@example.com'
```

we might supply the unusual email address “‘x’ or 1=1 --” to get

```sql
select id, email, password
from users
where email = 'x' or 1=1 --'
```

which will return a complete list of users.
HTML injection causes a server to deliver a surprising web page.

**SQL injection** can cause a database server to carry out unexpected actions on the database. For example, where a server contains code like this:

```sql
select id, email, password
from users
where email = 'bob@example.com'
```

we might supply the perverse email address “x’; update users set email='bob@example.com' where email='admin@server' --”
HTML injection causes a server to deliver a surprising web page.

SQL injection can cause a database server to carry out unexpected actions on the database. For example, where a server contains code like this:

```sql
select id, email, password
from users
where email = 'bob@example.com'
```

we might supply the perverse email address "'x'; update users set email='bob@example.com' where email='admin@server' --" to get

```sql
select id, email, password
from users
where email = 'x';
update users set email = 'bob@example.com'
where email = 'admin@server'
```

which will redirect all the administrator’s email to Bob.
Dubious Licence Plate
How then do we write programs to generate and manipulate queries?

A common approach is to use some standard framework or application programming interface (API). ODBC, the *Open Database Connectivity* specification, is a well-known framework for managed database access:

- At the back, an ODBC *driver* contains code for a specific database management system (DB2, Oracle, SQL Server, ...).
- At the front, the programmer connects to a fixed procedural API
- In between, ODBC libraries translate between API and driver.

Particular programming languages and environments may place further layers on top of ODBC, or use similar mechanisms. For example: *JDBC* for Java and *ADO.NET* for the Microsoft .NET framework.
JDBC: Java Database Connectivity

JDBC is a Java library, in the java.sql.* and javax.sql.* packages, which provides access to read, write and modify tabular data.

Relational databases, with access via SQL, is the most common application; but JDBC can also operate on other data sources.

The connection to the database itself may be via a driver that bridges through ODBC, speaks a proprietary database protocol, or connects to some further networking component or application.
import java.sql.*; // Obtain the relevant classes

// Install a suitable driver
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");

// Identify the database
String url = "jdbc:derby:Users";

// Prepare login information
String user = "bob"
String password = "secret"

// Open connection to database
Connection con = DriverManager.getConnection(url, user, password);
Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT name, id, score FROM Users");

while (rs.next()) // Loop through each row returned by the query
{
    String n  = rs.getString("name");
    int   i   = rs.getInt("id");
    float s   = rs.getFloat("score");
    System.out.println(n+i+s);
}
float findScoreForUser(Connection con, String name) {

    Statement stmt = con.createStatement();

    String query =
        "SELECT id, score FROM Users WHERE name=" + name;

    ResultSet rs = stmt.executeQuery(query);

    float s = rs.getFloat("score");

    return s;
}
String findUsersInRange(Connection con, float low, float high) {

    String prequery = "SELECT id, name FROM Users WHERE ? < score AND score < ?";

    PreparedStatement stmt = con.prepareStatement(prequery);
    stmt.setFloat(1, low);  // Fill in the two
    stmt.setFloat(2, high);  // missing values

    rs = stmt.executeQuery(query);  // Now run the completed query

    String answer = "";  // Start building our answer
    while (rs.next())  // Cycle through the query responses
        { answer = answer + rs.getInt("id") + ":" + rs.getString("name") + "\n"; }
    return answer;
}
That seems like hard work

These examples use the full generality of the ODBC framework:

- Arbitrary drivers . . .
- . . . possibly to proprietary data sources . . .
- . . . across the network . . .
- . . . with authentication and authorization . . .
- . . . ensuring consistency under multiple transactions.

Appropriate wrappers and frameworks can make things more straightforward in simpler situations.

However, the basic scheme of building queries as strings is ubiquitous.
Can I use something else instead?

There are several other approaches to database access:

- Frameworks like Java **Hibernate** preserve objects over time.
- **Object Relational Mapping (ORM)** translates between programming-language object structures and persistent database storage.
- An **Object-Oriented Database Management System (OODBMS)** works with objects instead of relations, tables and rows.
- Other systems like **CouchDB**, **BigTable**, and **Cassandra** store a variety of structured data for shared access.

These are sometimes known collectively as **NoSQL** architectures.

In general, these offer a different mix of features and performance to conventional relational database management systems (RDBMS).
So why would I want to use SQL anyway?

Sometimes, SQL is inevitable for non-technical reasons:

- External databases
- Legacy databases
- Existing code

Often, though, it’s because of genuine advantages of relational databases:

- Complex queries joining multiple large datasets
- Efficient query optimization and execution
- Transactional consistency

SQL or not, though, any scheme which involves handing over complex queries to a specialized engine will encounter the issue of using one language from another.
SQL is a domain-specific language for programming queries over relational databases. Queries may be complex, with declarative and imperative components, and are often constructed by other programs rather than by hand.

Websites are vulnerable to cross-site scripting or XSS whenever they take user input and use it to generate pages. The analogous problem of SQL injection arises where free text input is used to construct structured queries.

Programs generating SQL code use frameworks like JDBC or ADO.NET; and these do construct queries using unstructured string manipulation. Using prepared strings begins to add back some structure.

SQL queries are programs in a structured high-level language, but we treat them as unstructured text.
Thursday’s lecture will be about the LINQ framework on C#.

- Find an online tutorial about C#
- Read it.
- Post the URL, and your comments on the tutorial, to the blog
To find out more about database access in Java and C#, start with these tutorials:

- Sun’s JDBC tutorial

- The C# Station ADO.NET tutorial