Advances in Programming Languages
APL10: Using SQL from Java

Ian Stark

School of Informatics
The University of Edinburgh

Thursday 12 February 2009
Semester 2 Week 5
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
- LINQ: .NET Language Integrated Query
- Language integration in F#
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
- LINQ: .NET Language Integrated Query
- Language integration in F#
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 as a way to communicate with 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/
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 own web server to present a page that requests account details and then sends them to an attacker’s own site.

The opportunity to inject HTML and even Javascript can arise whenever a web site takes user input and uses that to generate pages. It is even possible to use web search engines to locate vulnerable sites.
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.
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'
```
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 --”
**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.
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 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.
HI, THIS IS YOUR SON’S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.

http://xkcd.com/327
HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR - DID HE BREAK SOMETHING?

IN A WAY-

http://xkcd.com/327
XKCD on SQL Injection

http://xkcd.com/327
XKCD on SQL Injection

HI, THIS IS YOUR SON’S SCHOOL. WE’RE HAVING SOME COMPUTER TROUBLE.

OH, DEAR – DID HE BREAK SOMETHING? IN A WAY-

DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?

OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.

WELL, WE’VE LOST THIS YEAR’S STUDENT RECORDS. I HOPE YOU’RE HAPPY.

AND I HOPE YOU’VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://xkcd.com/327
Secunia Advisory: SA33877
Release Date: 2009-02-11
Critical: Moderately critical
Impact: Security bypass; Manipulation of data
Where: From remote
Solution Status: Vendor Patch
Software: w3b|CMS (Web Content Management System)

Description
Input passed to the `cms_admin` cookie in `admin/index.php` is not properly sanitized before being used in SQL queries. This can be exploited to manipulate SQL queries by injecting arbitrary SQL code.

Similarly for parameters `id`, `action`, `seite`, ...

Affects modules Blog, Downloads, Gallery, ...

http://secunia.com/advisories/33877/
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, core ODBC libraries translate between the API and the driver.

Particular programming languages and environments may place further layers on top of ODBC, or have alternative similar mechanisms. For example: *JDBC* for Java and *ADO.NET* for the Microsoft .NET framework.
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);
Sample JDBC

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;
}
JDBC Prepared Strings

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;
}
Have a look at these two tutorials on database access in Java and C#.

- Sun’s JDBC tutorial

- The C# Station ADO.NET tutorial

You don’t need to work through every detail, but the key is to see how these languages provide control of SQL.
SQL is a domain-specific programming language.
This makes it excellent for abstraction and expressiveness in its domain.
Treating SQL programs as strings ignores all of this.
Lots of programs write other SQL programs, by concatenation.
That can create problems, most notoriously security holes.
Standard frameworks may plug some holes, but that’s about it.

SQL queries are programs in a structured high-level language, but we treat them as unstructured text.