

CC3201-1

BASES DE DATOS

OTOÑO 2018

Clase 8: SQL: Acceso Programático,  
Inyecciones, Seguridad

Aidan Hogan

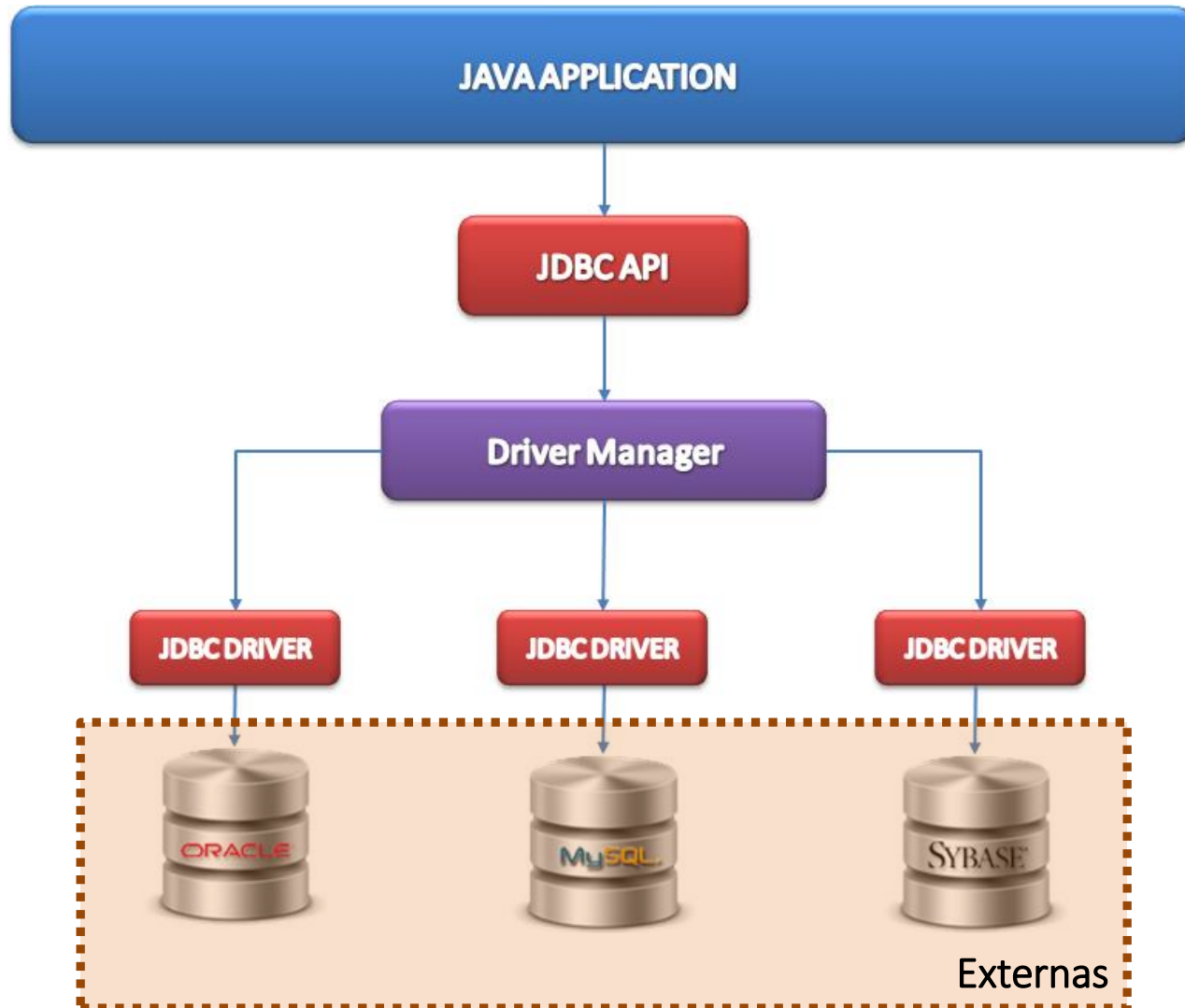
aidhog@gmail.com



# ACCESO PROGRAMÁTICO (JAVA): *JAVA DATABASE CONNECTIVITY* (JDBC)

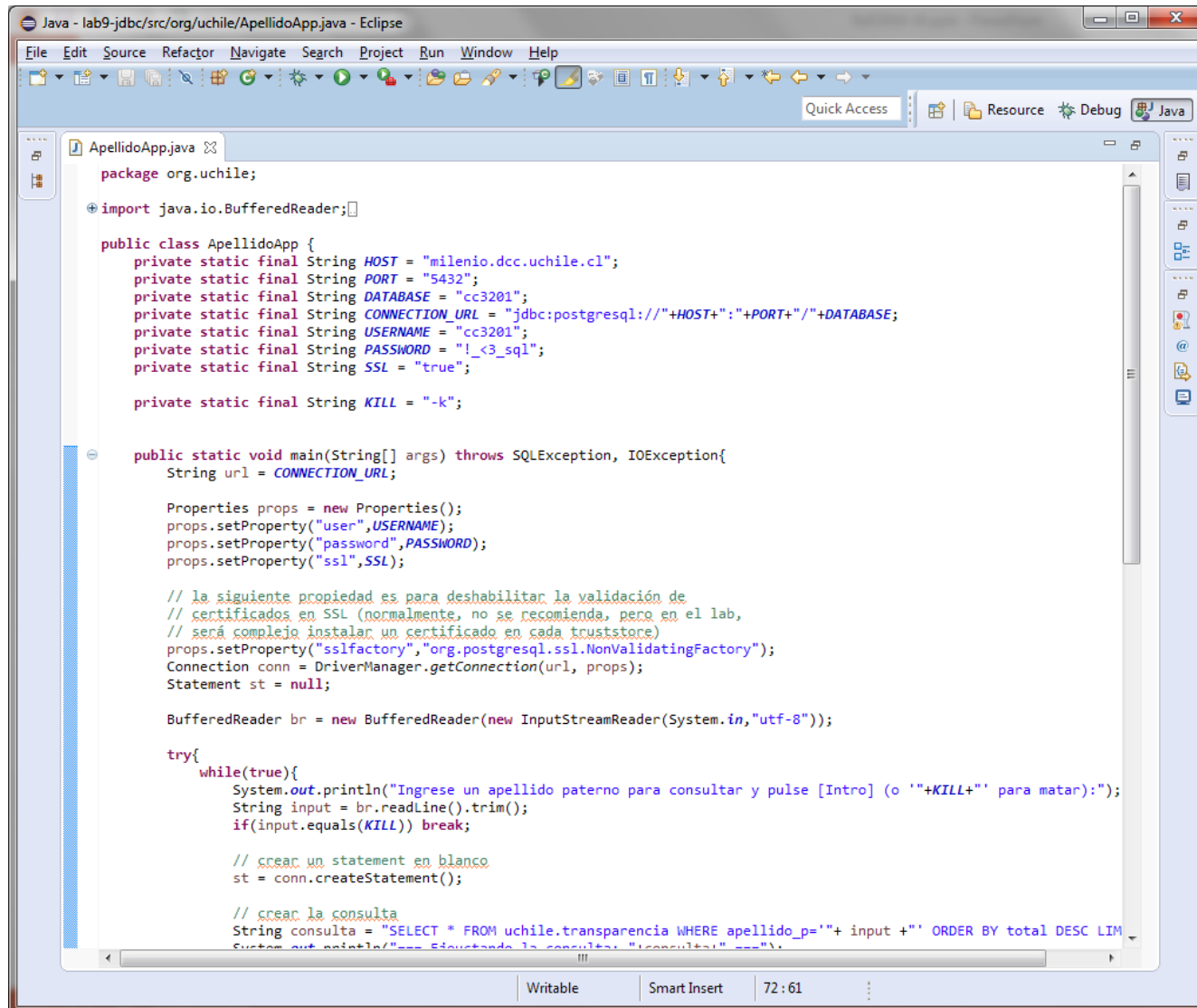


# Java Database Connectivity (JDBC)





# Veremos el ejemplo ApellidoApp.java



```
Java - lab9-jdbc/src/org/uchile/ApellidoApp.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access Resource Debug Java

ApellidoApp.java
package org.uchile;

import java.io.BufferedReader;

public class ApellidoApp {
    private static final String HOST = "milenio.dcc.uchile.cl";
    private static final String PORT = "5432";
    private static final String DATABASE = "cc3201";
    private static final String CONNECTION_URL = "jdbc:postgresql://" + HOST + ":" + PORT + "/" + DATABASE;
    private static final String USERNAME = "cc3201";
    private static final String PASSWORD = "!_<3_sql";
    private static final String SSL = "true";

    private static final String KILL = "-k";

    public static void main(String[] args) throws SQLException, IOException{
        String url = CONNECTION_URL;

        Properties props = new Properties();
        props.setProperty("user", USERNAME);
        props.setProperty("password", PASSWORD);
        props.setProperty("ssl", SSL);

        // la siguiente propiedad es para deshabilitar la validación de
        // certificados en SSL (normalmente, no se recomienda, pero en el lab,
        // será complejo instalar un certificado en cada truststore)
        props.setProperty("sslfactory", "org.postgresql.ssl.NonValidatingFactory");
        Connection conn = DriverManager.getConnection(url, props);
        Statement st = null;

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in, "utf-8"));

        try{
            while(true){
                System.out.println("Ingrese un apellido paterno para consultar y pulse [Intro] (o '"+KILL+"' para matar):");
                String input = br.readLine().trim();
                if(input.equals(KILL)) break;

                // crear un statement en blanco
                st = conn.createStatement();

                // crear la consulta
                String consulta = "SELECT * FROM uchile.transparencia WHERE apellido_p='"+ input + "' ORDER BY total DESC LIM";
                System.out.println("=== Ejecutando la consulta: '"+consulta+"' ===");
            }
        }
    }
}
```

Writable Smart Insert 72:61



# Consulta vs. Actualización

- Para hacer consultas (SELECT):

```
String consulta = "SELECT ...";  
ResultSet rs = statement.executeQuery(consulta);
```

- Para hacer actualizaciones (INSERT; UPDATE, ...)

```
String actualizacion = "UPDATE ...";  
int tuplasAfectadas = statement.executeUpdate(actualizacion);
```



# Un problema ...

```
System.out.println("Ingrese un apellido paterno:");
String input = br.readLine().trim();
if(input.equals(KILL)) break;

// crear un statement en blanco
st = conn.createStatement();

// crear la consulta
String consulta =
    "SELECT * FROM u Chile.transparencia "
    + "WHERE apellido_p='"+input+"' "
    + "ORDER BY total DESC LIMIT 10";
ResultSet rs = st.executeQuery(consulta);

// ...
```

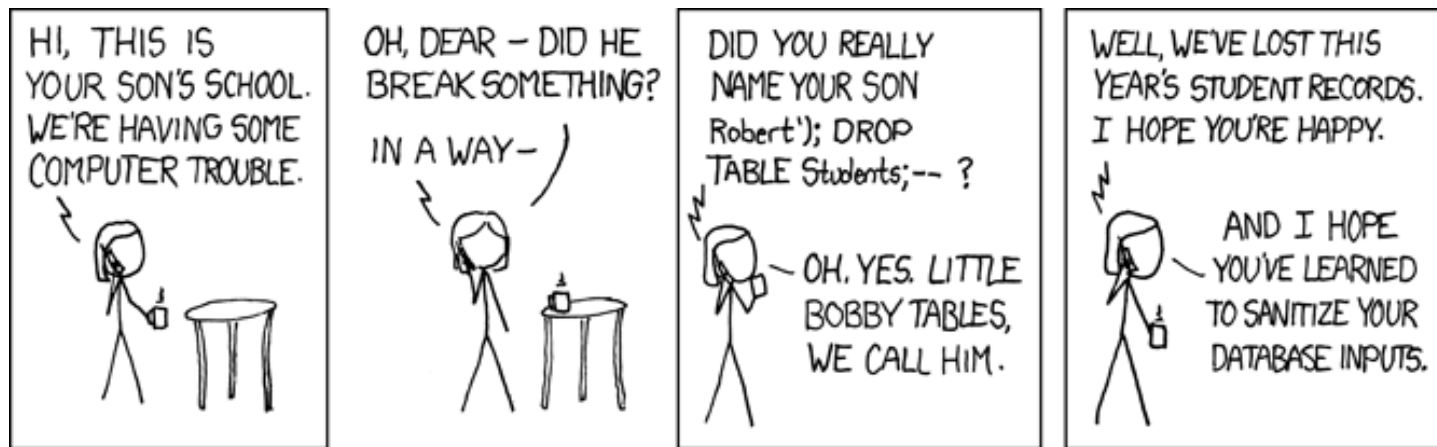
*¿Hay algún problema aquí?*

... no hemos “verificado” el input.



# Inyección SQL

- Un usuario malintencionado puede ingresar un string de entrada para hacer algo inesperado



```
"SELECT nota FROM Students WHERE name='"+input+"'"
```

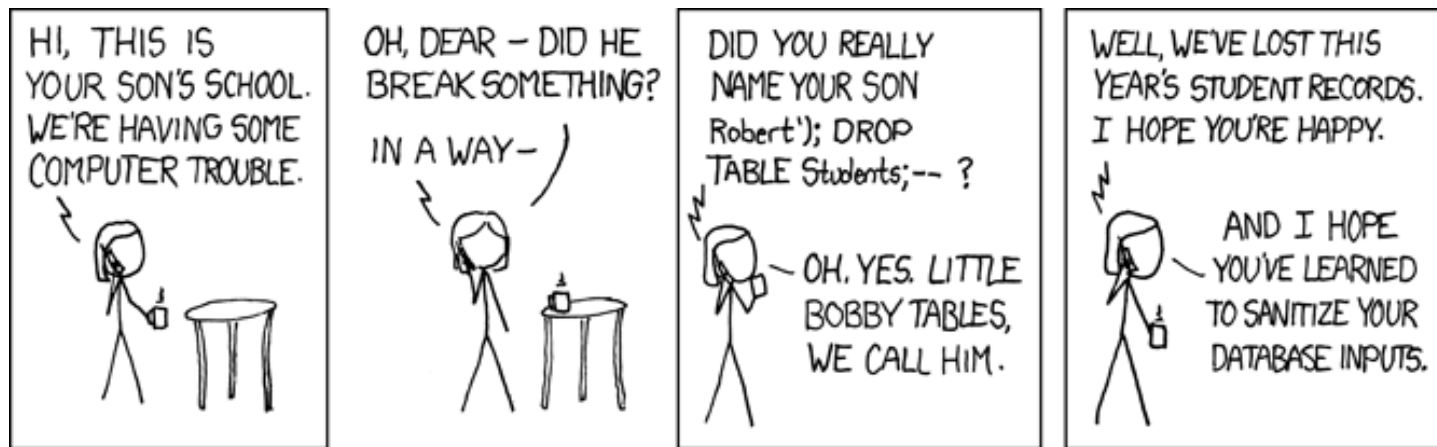
```
"SELECT nota FROM Students WHERE name='Robert'); DROP TABLE Students; -- '"
```

( '-- ' empieza un comentario)



# Inyección SQL (muchas formas)

- Un usuario malintencionado puede ingresar un string de entrada para hacer algo inesperado



```
"SELECT nota FROM Students WHERE name='"+input+"'"
```

```
"SELECT nota FROM Students WHERE name='Robert' OR 1=1;'"
```

¿Qué hace el ejemplo?

¡Devolverá toda la tabla!



# Parece estúpido pero ...

[https://www.owasp.org/images/7/72/OWASP\\_Top\\_10-2017\\_%28en%29.pdf.pdf](https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf)

**T10**

**OWASP Top 10**  
**Application Security Risks – 2017**

6

<b>A1:2017- Injection</b>	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
<b>A2:2017-Broken Authentication</b>	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
<b>A3:2017- Sensitive Data Exposure</b>	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
<b>A4:2017-XML External Entities (XXE)</b>	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
<b>A5:2017-Broken Access Control</b>	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.



# Parece estúpido pero (por ejemplo) ...



Information Commissioner's Office

The UK's independent authority set up to [uphold information rights in the public interest](#), promoting openness by public bodies and data privacy for individuals.

[Home](#) [For the public](#) [For organisations](#) [Report a concern](#) [Action we've taken](#) [About the ICO](#)

[About the ICO](#) / [News and events](#) / [News and blogs](#) /

## TalkTalk gets record £400,000 fine for failing to prevent October 2015 attack

Date **05 October 2016**

Type **News**

Telecoms company TalkTalk has been [issued with a record £400,000 fine](#) by the ICO for security failings that allowed a cyber attacker to access customer data "with ease".

The [ICO's in-depth investigation](#) found that an attack on the company last October could have been prevented if TalkTalk had taken basic steps to protect customers' information.


ICO investigators found that the cyber attack between 15 and 21 October 2015 took advantage of technical weaknesses in TalkTalk's systems. The attacker accessed the personal data of 156,959 customers including their names, addresses, dates of birth, phone numbers and email addresses. In 15,656 cases, the attacker also had access to bank account details and sort codes.



[illegible]

[https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)

## Examples [[edit source](#)]

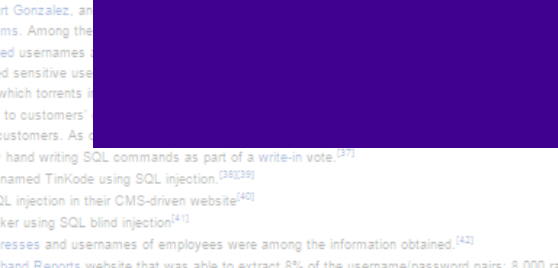
- 
- In February 2002, Jeremiah Jaks discovered that Guess.com was vulnerable to an SQL injection attack, permitting anyone able to construct a properly-crafted URL to pull down 200,000 names, credit card numbers and expiration dates in the site's customer database.<sup>[23]</sup>
- On November 1, 2005, a teenage hacker used SQL injection to break into the site of a Taiwanese information security magazine from the Tech Target group and steal customers' information.<sup>[24]</sup>
  - On January 13, 2006, Russian computer criminals broke into a Rhode Island government website and allegedly stole credit card data from individuals who have done business online with state agencies.<sup>[25]</sup>
  - On March 29, 2006, a hacker discovered an SQL injection flaw in an official Indian government's tourism site.<sup>[26]</sup>
  - On June 29, 2007, a computer criminal defaced the Microsoft UK website using SQL injection.<sup>[27][28]</sup> UK website *The Register* quoted a Microsoft spokesperson acknowledging the problem.
  - In January 2008, tens of thousands of PCs were infected by an automated SQL injection attack that exploited a vulnerability in application code that uses Microsoft SQL Server as the database store.<sup>[29]</sup>
  - In July 2008, Kaspersky's Malaysian site was hacked by a Turkish hacker going by the handle of "m0sted", who said to have used an SQL injection.
  - In February 2013, a group of Maldivian hackers, hacked the website "UN-Maldives" using SQL Injection.
  - In May 28, 2009 *Anti-U.S. Hackers Infiltrate Army Servers* Investigators believe the hackers used a technique called SQL injection to exploit a security vulnerability in Microsoft's SQL Server database to gain entry to the Web servers. "m0sted" is known to have carried out similar attacks on a number of other websites in the past—including against a site maintained by Internet security company Kaspersky Lab.
  - On April 13, 2008, the *Sexual and Violent Offender Registry of Oklahoma* shut down its website for "routine maintenance" after being informed that 10,597 Social Security numbers belonging to sex offenders had been downloaded via an SQL injection attack.<sup>[30]</sup>
  - In May 2008, a server farm inside China used automated queries to Google's search engine to identify SQL server websites which were vulnerable to the attack of an automated SQL injection tool.<sup>[29][31]</sup>
  - In 2008, at least April through August, a sweep of attacks began exploiting the SQL injection vulnerabilities of Microsoft's IIS web server and SQL Server database server. The attack does not require guessing the name of a table or column, and corrupts all text columns in all tables in a single request.<sup>[32]</sup> A HTML string that references a malware JavaScript file is appended to each value. When that database value is later displayed to a website visitor, the script attempts several approaches at gaining control over a visitor's system. The number of exploited web pages is estimated at 500,000.<sup>[33]</sup>
  - On August 17, 2009, the United States Department of Justice charged an American citizen, Albert Gonzalez, and two unnamed Russians with the theft of 130 million credit card numbers using an SQL injection attack. In reportedly "the biggest case of identity theft in American history", the man stole cards from a number of corporate victims after researching their payment processing systems. Among the companies hit were credit card processor Heartland Payment Systems, convenience store chain 7-Eleven, and supermarket chain Hannaford Brothers.<sup>[34]</sup>
  - In December 2009, an attacker breached a RockYou plaintext database containing the unencrypted usernames and passwords of about 32 million users using an SQL injection attack.<sup>[35]</sup>
  - On July 2010, a South American security researcher who goes by the handle "Ch Russo" obtained sensitive user information from popular BitTorrent site The Pirate Bay. He gained access to the site's administrative control panel and exploited a SQL injection vulnerability that enabled him to collect user account information, including IP addresses, MD5 password hashes and records of which torrents individual users have uploaded.<sup>[36]</sup>
  - From July 24 to 26, 2010, attackers from Japan and China used an SQL injection to gain access to customers' credit card data from Neo Beat, an Osaka-based company that runs a large online supermarket site. The attack also affected seven business partners including supermarket chains Izumiya Co, Maruetsu Inc, and Ryukyu Jusco Co. The theft of data affected 100,000 customers. As of August 14, 2010 it was reported that there have been more than 300 cases of credit card information being used by third parties to purchase goods and services in China.
  - On September 19 during the 2010 Swedish general election a voter stole the results of the election by sending writing SQL commands as part of a write-in vote.<sup>[37]</sup>
  - On November 8, 2010 the British Royal Navy website was compromised by a hacker using a SQL injection attack to steal sensitive information. The hacker used TinKode using SQL injection.<sup>[38][39]</sup>
  - On February 5, 2011 HBGary, a technology security firm, was breached by a hacker using an SQL injection in their CMS-driven website.<sup>[40]</sup>
  - On March 27, 2011, mysql.com, the official homepage for MySQL, was hacked by a hacker using SQL blind injection.<sup>[41]</sup>
  - On April 11, 2011, Barracuda Networks was compromised using an SQL injection attack. The attack resulted in the theft of sensitive information, including email addresses and usernames of employees were among the information obtained.<sup>[42]</sup>
  - Over a period of 4 hours on April 27, 2011, an automated SQL injection attack was launched against a Reports website that was able to extract 8% of the username/password pairs: 8,000 random accounts of the 9,000 active and 90,000 old or inactive accounts.<sup>[43][44][45]</sup>
  - On June 1, 2011, "hacktivists" of the group LulzSec were able to exploit a vulnerability in a Sony's website, accessing the personal information of a million users.<sup>[46][47]</sup>
  - In June 2011, PBS was hacked, mostly likely through use of an SQL injection attack. The attack was described in this Imperva® blog.<sup>[48]</sup>
  - In July 2012, the website for Wuhan Online, a massively multiplayer online game, was hacked by a hacker using an SQL injection while the site was being updated.<sup>[49]</sup>
  - In July 2012 a hacker group known as the "LulzSec" group breached Yahoo! Voices. The group breached Yahoo's security by using a "union-based SQL injection technique".<sup>[50][51]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of the University of students, faculty, employees, and alumni from 53 universities including Harvard, Princeton, Stanford, Cornell, Johns Hopkins, and the University of Zurich on pastebin.com. The hackers claimed that they were trying to "hack" the university's website to "expose" the university's financial records, including changing education laws in Europe and increases in tuition in the United States.<sup>[52]</sup>
  - On June 27, 2011, a hacker group known as the "LulzSec" group breached the website of a utility company. They've been able to erase people's debts to water, gas, Internet, electricity, and telephone companies. Additionally, they published admin user name and password for other citizens to log in and clear their debts.<sup>[53]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of the Chinese Chamber of International Commerce. The leaked data was posted publicly in cooperation with Anonymous.<sup>[54]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[55]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[56]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[57]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[58]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[59]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[60]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[61]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[62]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[63]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[64]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[65]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[66]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[67]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[68]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[69]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[70]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[71]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[72]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[73]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[74]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[75]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[76]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[77]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[78]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[79]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[80]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[81]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[82]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested for reporting the security vulnerability. 70,000 user details were exposed over this conflict.<sup>[83]</sup>
  - On October 1, 2012, a hacker group known as the "LulzSec" group breached the website of a security company. The hackers arrested



# Más ejemplos ...

[https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)

Examples [\[edit source\]](#)



- On June 1, 2011, "hacktivists" of the group LulzSec were accused of using SQLi to steal coupons, download keys, and passwords that were stored in plaintext on Sony's website, accessing the personal information of at least 400,000 users.<sup>[32]</sup> A HTML string that references a malware JavaScript file is appended to each value. When that data is processed, the malware is executed.<sup>[33]</sup>
- In June 2011, PBS was hacked.<sup>[34]</sup>
- In May 2012, the website for the British Broadcasting Corporation (BBC) was hacked, and the personal records of 15 account details were leaked.<sup>[35]</sup>
- In July 2012 a hacker group called @delete leaked 71 Chinese government records.<sup>[36]</sup>
- On October 1, 2012, a hacker group called @delete leaked 15 account details from Yahoo!.<sup>[37]</sup>
- On June 27, 2013, hacker group called @delete leaked 15 account details from Yahoo!.<sup>[38]</sup>
- On November 4, 2013, hacker group called @delete leaked 15 account details from Yahoo!.<sup>[39]</sup>
- On February 2, 2014, AVS TV was hacked, and the personal records of 15 account details were leaked.<sup>[40]</sup>
- On February 21, 2014, United Kingdom's National Health Service (NHS) was hacked, and the personal records of 15 account details were leaked.<sup>[41]</sup>
- On February 21, 2014, Hackers leaked 15 account details from Yahoo!.<sup>[42]</sup>
- On March 7, 2014, officials at British telecommunications company Talk Talk's servers, exploiting a vulnerability in a legacy web portal.<sup>[43]</sup>
- In August 2014, Milwaukee-based computer security company Hold Security disclosed that it uncovered a theft of confidential information from nearly 420,000 records.<sup>[44]</sup>
- In October 2015, an SQL injection attack was used to steal the personal details of 156,959 customers from British telecommunications company Talk Talk's servers, exploiting a vulnerability in a legacy web portal.<sup>[45]</sup>



# El Jefe de HBGary ...



**aaronbarr**

Today we taught everyone a lesson. When we actually decide to bite back against those who try to bring us down, we bite back hard. [#gameover](#)

23 minutes ago via web

<http://vocaroo.com/?media=vY7n2sXJaoPZVTHGq> Aaron's new resumé amirite [#hurrhurr](#)

about 1 hour ago via web

Spot the edit: <http://www.linkedin.com/in/tedvera> you Ted Vera, you're not getting away either! Nom nom nom, who's next? Penny? [#hbgary](#)

about 1 hour ago via web

Here's my address: [REDACTED]

about 1 hour ago via web

Here's my social security number: [REDACTED]

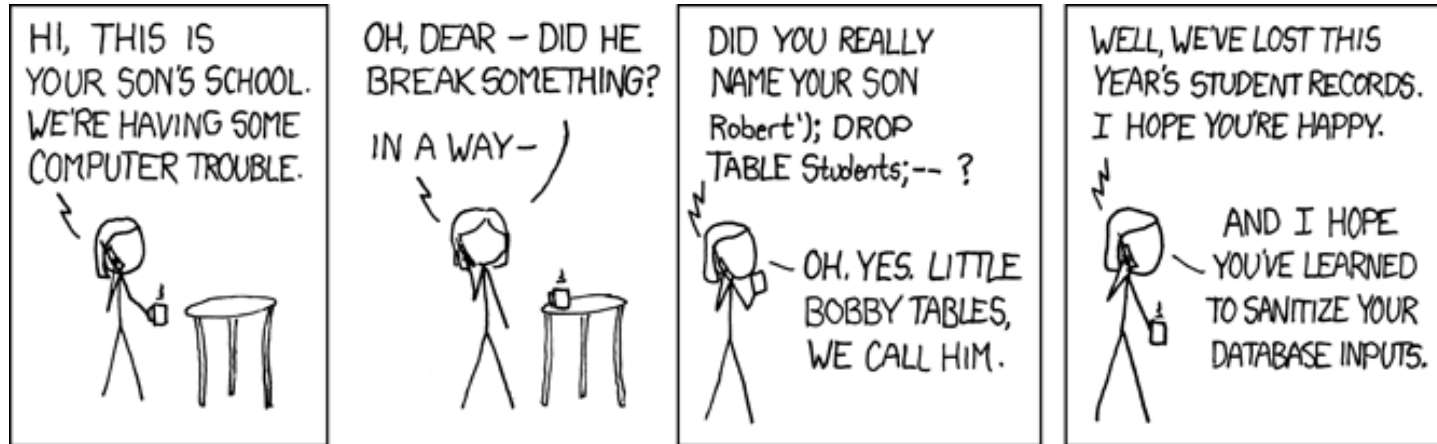
about 1 hour ago via web



**HBGary**  
Detecting Tomorrow's Threats Today  
Part of ManTech International Corporation



# Inyección SQL

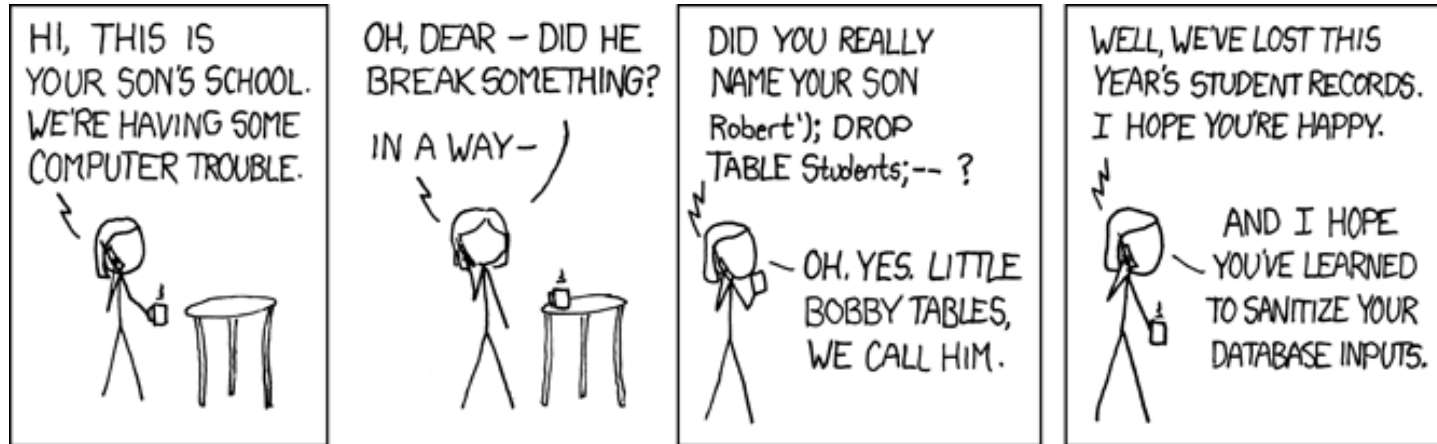


```
String consulta = "SELECT nota FROM Students WHERE name='"+input+"'";  
ResultSet rs = statement.executeQuery(consulta);
```

*¿Cómo podemos resolver el problema?*



# Inyección SQL: ¿escapar los strings?



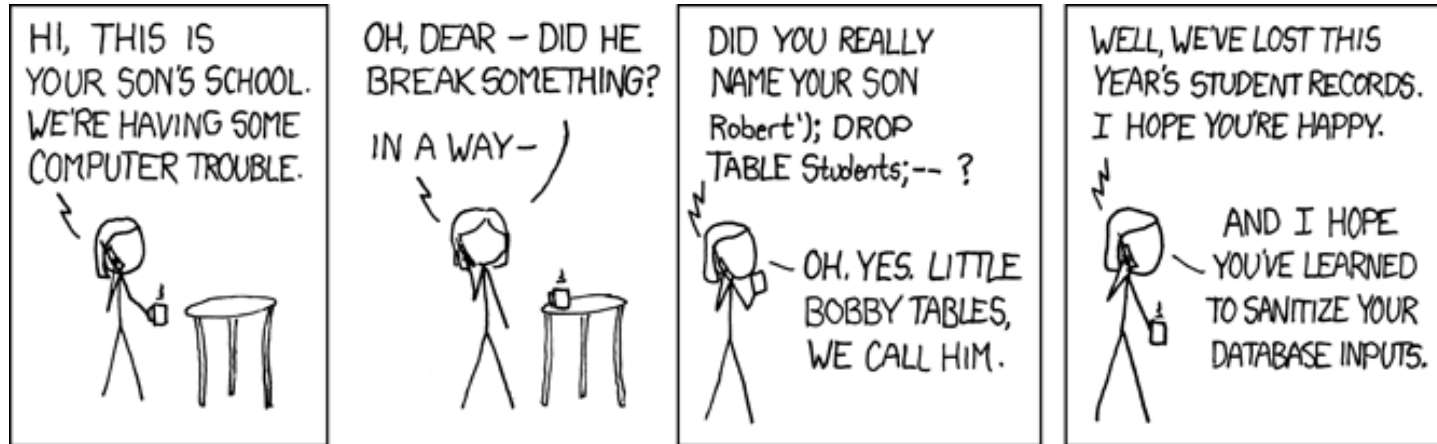
```
String consulta = "SELECT nota FROM Students WHERE name='"+escapar(input)+"'";  
ResultSet rs = statement.executeQuery(consulta);
```

*¿Cómo podemos resolver el problema?*

*Mejor*, pero sería complicado implementar la función *escapar* en un lenguaje de programación general y garantizar que prevenga cada tipo de inyección en cada versión (futura) de cada sistema de Bdd dado cualquier tipo de consulta y entrada!



# Inyección SQL: *¡sentencias pre-compiladas!*



```
String consulta = "SELECT nota FROM Students WHERE name='?'";  
// donde ? es un parámetro que reemplazaremos con la entrada del usuario  
PreparedStatement ps = conn.prepareStatement(consulta);  
ps.setString(1, input);  
ResultSet rs = ps.executeQuery();
```

*Mandamos la consulta al sistema de bases de datos y después se reemplazarán los parámetros con la entrada del usuario*



# Inyección SQL: *¡sentencias pre-compiladas!*

```
String consulta = "SELECT nota FROM Students WHERE name=?";

PreparedStatement ps = conn.prepareStatement(consulta);           // 1
ps.setString(1, input);                                           // 2
ResultSet rs = ps.executeQuery();                                  // 3
```

// 1 : El sistema de bases de datos compila la sentencia

SELECT nota FROM Students WHERE name=?

QUERY PLAN

-----  
Seq Scan on Students (cost=0.00..9654.67 rows=57 width=132)

Filter: ((name)::text = ?::text)

El sistema de base de datos

*La consulta es compilada por el sistema **sin** la entrada*



# Inyección SQL: *¡sentencias pre-compiladas!*

```
String consulta = "SELECT nota FROM Students WHERE name=?";

PreparedStatement ps = conn.prepareStatement(consulta);           // 1
ps.setString(1, input);                                         // 2
ResultSet rs = ps.executeQuery();                               // 3
```

// 2 : El sistema de bases de datos reemplaza el parametro en el plan

```
SELECT nota FROM Students WHERE name=?
```

QUERY PLAN

-----  
Seq Scan on Students (cost=0.00..9654.67 rows=57 width=132)

Filter: ((name)::text = 'Robert'::text)

El sistema de base de datos

*Se reemplaza el parámetro en la sentencia pre-compilada  
(que es un plan en memoria, no un string)*



# Inyección SQL: *¡sentencias pre-compiladas!*

```
String consulta = "SELECT nota FROM Students WHERE name=?";

PreparedStatement ps = conn.prepareStatement(consulta);           // 1
ps.setString(1, input);                                         // 2
ResultSet rs = ps.executeQuery();                               // 3
```

// 3 : El sistema de bases de datos ejecuta el plan

SELECT nota FROM Students WHERE name=?

QUERY PLAN

-----  
Seq Scan on Students (cost=0.00..9654.67 rows=57 width=132)

Filter: ((name)::text = 'Robert'::text)

El sistema de base de datos

nota
3,7



# Sentencias pre-compiladas

```
String consulta = "SELECT nota FROM Students WHERE name=? AND year=?";
```

```
PreparedStatement ps = conn.prepareStatement(consulta);  
for(String[] input:inputs){  
    ps.setString(1, input[1]);  
    ps.setInt(2, Integer.parseInt(input[2]));  
    ResultSet rs = ps.executeQuery();  
    ...  
}
```

Se puede reutilizar el PreparedStatement varias veces  
(es más eficiente también: se compila la sentencia sólo una vez)

Se puede tener varios parámetros con varios tipos



Preguntas?



CATS : ALL YOUR BASE ARE BELONG  
TO US.