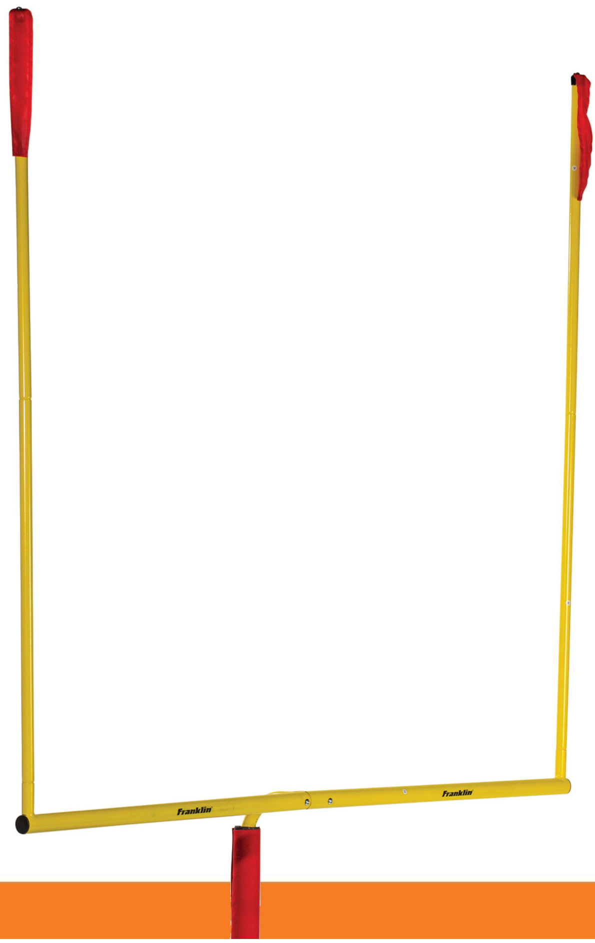
# SQL Injection

Develop
Intelligence

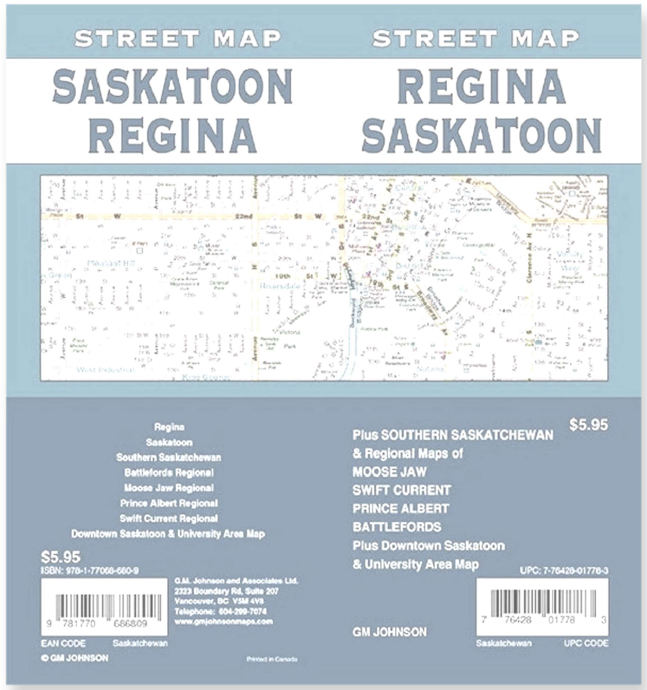
## Goals

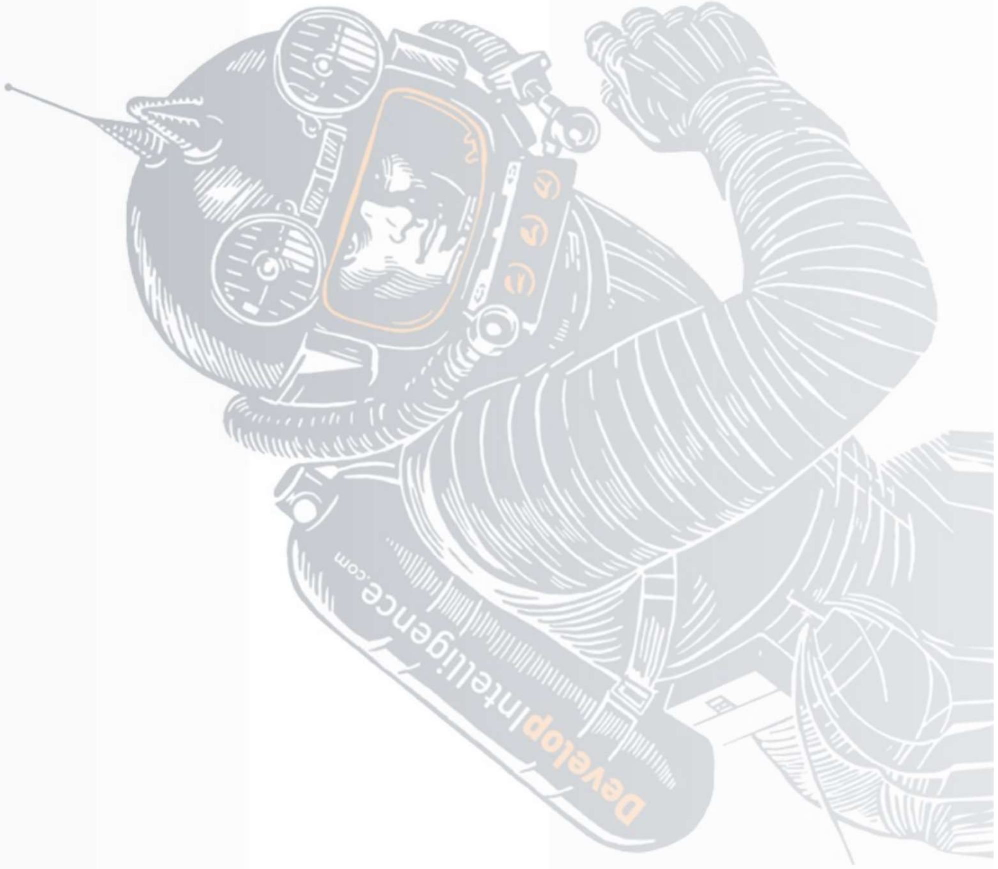
- Explain how SQL injection attacks work
- List 3 techniques to mitigate



## Roadmap

1. Basics
2. Executing SQL
3. Mitigation





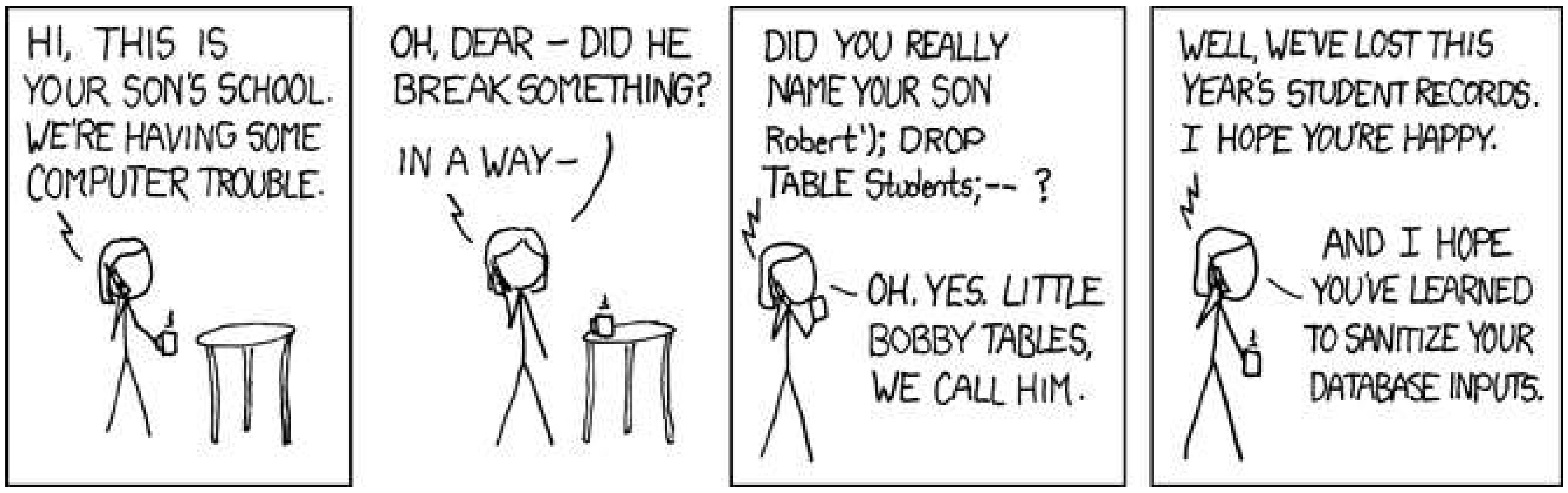
Develop Intelligence

## Basics

## SQL Injection Defined

- Code injection technique
- Attacks data-driven applications
- Run malicious SQL statements through user input

## Example: Exploits of a Mom


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.

## Details

- It's been around forever
- Still happening all the time
- Very Popular

## SQL Injection is Popular

1. Lots of vulnerable applications
2. Databases make attractive targets

## Injection Flavors

1. Scary Input
2. Error messages
3. Blind Attacks

## Recent SQL Injection Victims

- Bulgarian Revenue Agency
- United Nations Internet Governance Forum
- Johns Hopkins University

## Good News Everyone!

- It's easy to avoid
- Never combine user input with Entity SQL command text
- Don't
    - Trust user input
    - Concatenate SQL strings
- Choose an ORM

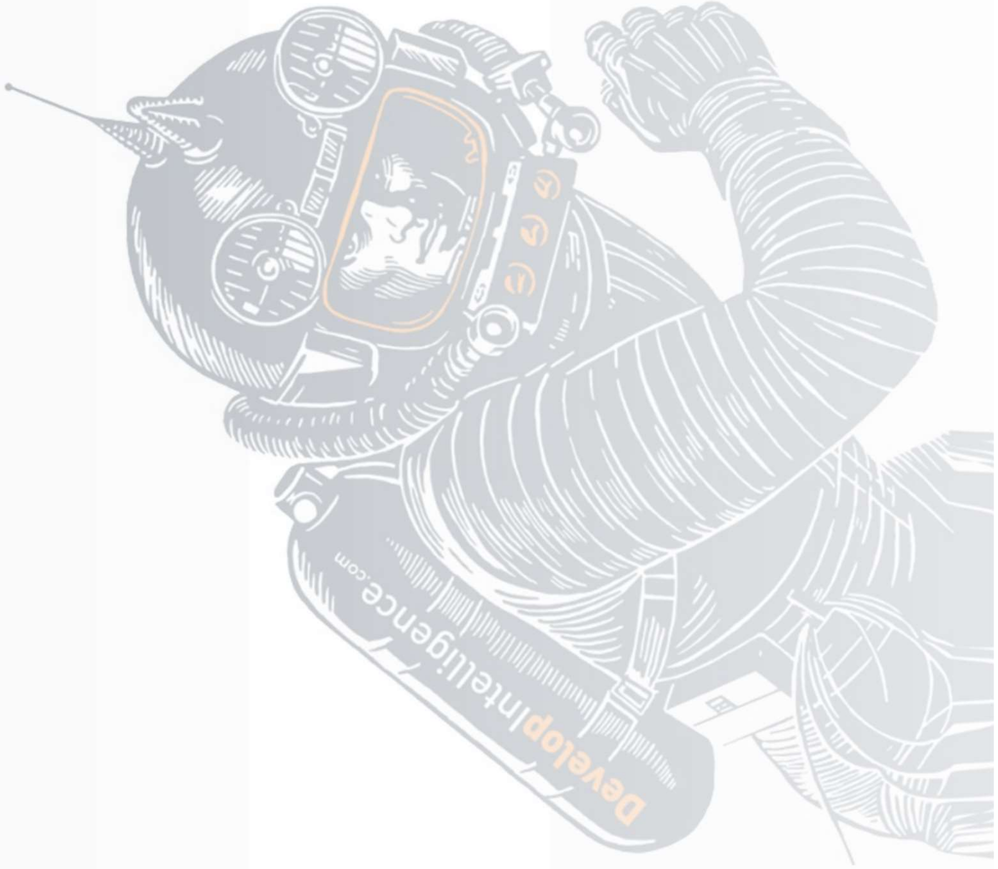
## Dumb Example

Don't do this:

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

Because of this:

```
a';DROP TABLE users; SELECT * FROM userinfo WHERE 't';--
```


DevelopIntelligence.com

## Executing SQL

## EF Core Review

- DbContext
- DbSet
- Entity Framework Core

## No ADO.NET Required

- Even though EF Core exposes familiar classes like:
    - SqlCommand
    - SqlCommandBuilder
    - SqlConnection
- Ideally, everything happens through the DbContext

## But You Can Still Execute SQL

- Which means SQL Injection is still possible
- Two extension methods on `DbSet`:
  - [`FromSqlRaw`]
  - [`FromSqlInterpolated`]

## Example FromSqlRaw

```
var user = getUserName();
var blogs = context.Blogs
    .FromSqlRaw("SELECT * FROM dbo.Blogs where userName = '" + user + "';")
    .ToList();
```

**Bad Idea!**

## Good News About DbSet Methods

- Results are still typed
- Can only be used on query roots
- One statement at a time

## Even Scarier: DatabaseFacade

- Provides database-level functionality
- Exposes `ExecuteSqlRaw`

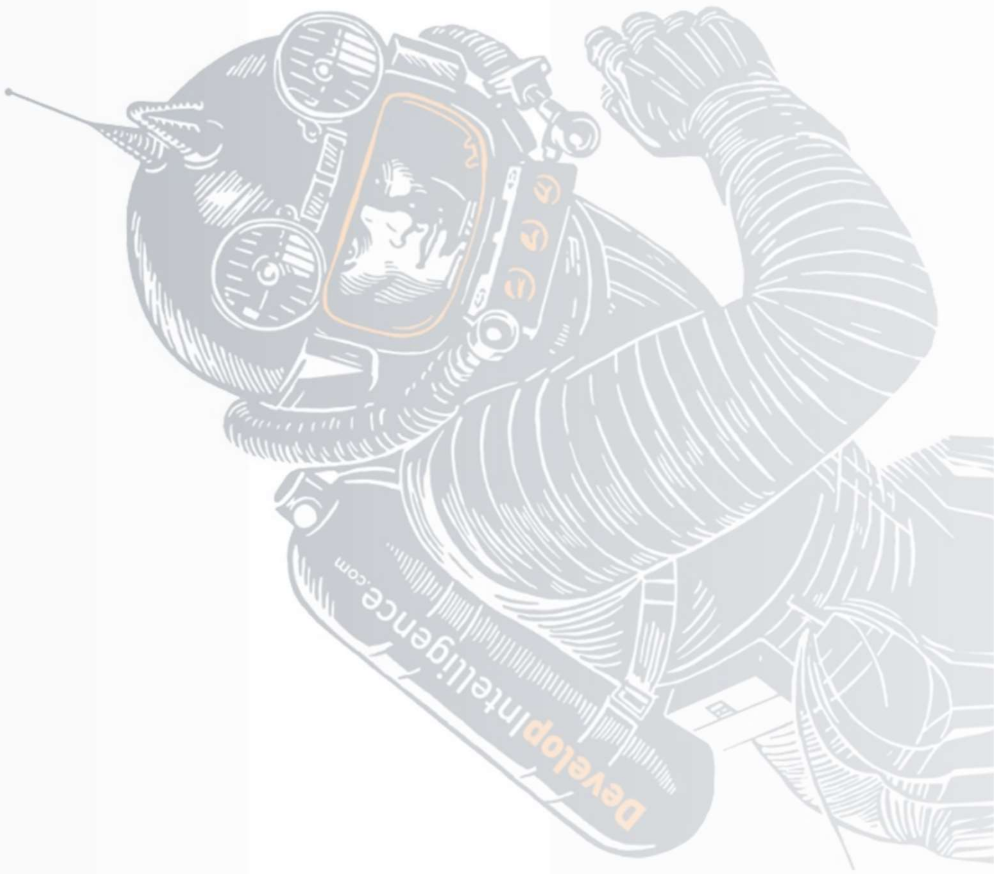
## Example

```
var query = "INSERT dbo.Snake(ID, Name, meannessLevel) VALUES(";
query += "'" + toCreate.ID.ToString() + "', ";
query += "'" + toCreate.Name + "', ";
query += toCreate.MeannessLevel.ToString();
query += ");";

int result = this.context.Database.ExecuteSqlRaw(query);
return this.Snakes.FirstOrDefault(s => s.ID == toCreate.ID);
```

## Why would you use a raw query?

- Performance - LINQ-generated SQL isn't always efficient
- Expressiveness - LINQ and SQL aren't 100% equivalent
- Stupid database tricks
    - Table-valued function


DevelopIntelligence.com

## Mitigation

## Options In Order

1. Do everything with LINQ
2. Paramaterize/Interpolate queries
3. Use stored procedures
4. Sanitize by hand (Yikes!)
5. One weird trick

## Technique #1

**No Direct SQL Execution in C#**

- Do everything with LINQ

## Example

```
var user = getUserName();
var blogs = context.Blogs
    .Where(b=>b.UserName == user)
    .Where(b=>b.Age > 10)
    .ToList();
```

## Technique #2

**Paramaterize your queries**

- `FromSqlRaw` has an overload to pass parameters
- It looks like string interpolation
- Actually gets turned into a DbParameter object

## Example

Develop Intelligence

```
var user = getUserName();
var blogs = context.Blogs
    .FromSqlRaw("SELECT * FROM dbo.Blogs where userName = {0}", user)
    .ToList();
```

## Technique #2 Variation

**Interpolate your queries**

- `FromSqlInterpolated` works like `FromSqlRaw`
- Allows interpolation syntax
- Effectively the same

## Example

```
var user = getUserName();
var blogs = context.Blogs
    .FromSqlInterpolated("SELECT * FROM dbo.Blogs where userName = {user}")
    .ToList();
```

## Technique #3: Stored Procedures

- Not 100% safe
    - Dynamic SQL in the sproc is still vulnerable
- But it forces parameterization

## Example

```
var user = new SqlParameter("user", "johndoe");

var blogs = context.Blogs
    .FromSqlRaw("EXECUTE dbo.GetMostPopularBlogsForUser @user", user)
    .ToList();
```

## Technique #4: Sanitize At Your Peril

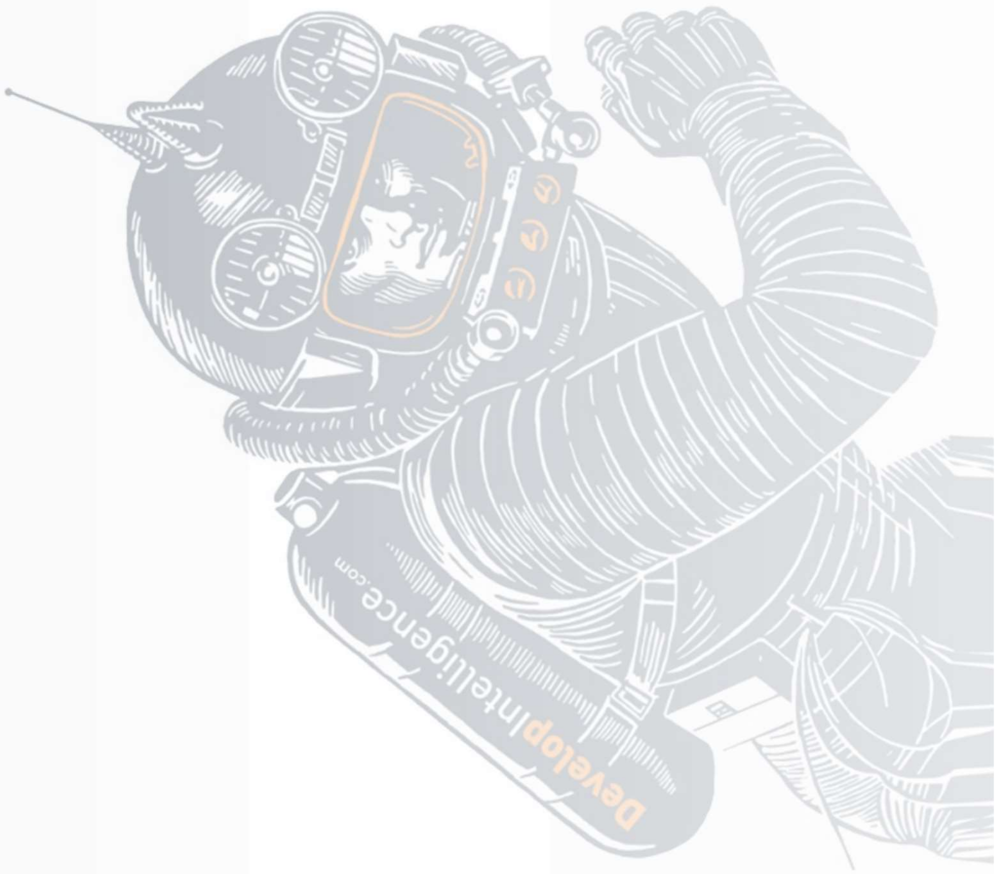
- Escape possibly dangerous strings
- Not an exact science
- Often possible to hack

## Example

```
Encoder oe = new OracleEncoder();
String query = "SELECT user_id FROM user_data WHERE user_name = '"
+ oe.encode( req.getParameter("userID")) + "' and user_password = '"
+ oe.encode( req.getParameter("pwd")) +"'";
```

## One Weird Trick: Table-Valued Parameters

- Enable the passing in of 'arrays'
- Advantages:
  - Single trip to the well
  - No dynamic sql generation


DevelopIntelligence.com

## Review

- Explain how SQL injection attacks work
- List 3 techniques to mitigate