

# THE GOLDEN HAMMER

Confessions of a Recovering Database Abuser



*"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail." - Abraham Maslow (1966)*

# Who am I?

## Shawn Oden

*I started life as a pilot who became a flight instructor who was briefly an airline pilot before I accidentally switched into a ColdFusion developer who, by some fluke, morphed into a database developer who adventurously (and somewhat masochistically) decided to be a DBA. And that's who I am today. Who knows what tomorrow may bring.*

- Microsoft Certified Professional: SQL Server
- CompTIA Security+
- Adobe Certified ColdFusion Specialist
- Adobe Certified ColdFusion MX 7 Developer
- Macromedia Certified ColdFusion MX Developer

**@CodeFuMonkey**  
[codefumonkey@gmail.com](mailto:codefumonkey@gmail.com)  
<https://www.codefumonkey.com>



**Note:** I am **CodeFuMonkey**, **NOT ColdFuMonkeh**, nor am I affiliated with **monkehworks.com**. I know that Mr. Gifford and I are easy to get mixed up, but the similarity of our noms de plume occurred many moons ago and was completely unintentional on my part. Sorry, Matt.

# First things first...

## ■ Rule #1:

### ➤ *DO NOT TEST THINGS IN A PRODUCTION DATABASE!*

- Trust me. Murphy's Law is a real thing.

## ■ Rule #2:

### – *DO NOT REPORT FROM A PRODUCTION DATABASE.*

- Set up a Reporting instance of your database. Your users will appreciate it.

## ■ Rule #3:

### – *BACKUPS ARE WORTHLESS IF THEY AREN'T TESTED.*

- If you have a need to RESTORE a BACKUP, chances are it's a bad day.
- Test them before you need them.
- Script the RESTOREs so you don't have to dig into StackOverflow.

## ■ Rule #4:

### – *WHEN IT COMES TO DATABASES, AS WITH MOST PROGRAMMING THINGS, THERE REALLY ARE FEW HARD AND FAST RULES.*

- It nearly always Depends.

## ■ Rule #5:

### – *EXCEPT WHEN IT COMES TO TESTING THINGS IN PRODUCTION*

### ➤ *DO NOT DO IT!*

# Now that we've reviewed some basic rules, let's talk about ...

- What is SQL?
- Databases Are Complex.
- What Bad Habits Did I Get Into?
  - *SIDEBAR: JOINS*
- Some SQL Gotchas.
- There's Just So Many Things To Cover
- Wrapping Up. For Now.

Let's jump in...

# What is SQL?

According to Webster's Dictionary:

*"SQL - /S Q L/ An industry-standard language for creating, updating and, querying relational database management systems."*

According to Wikipedia:

*"SQL (/ˌɛsˌkjuːˈɛl/ S-Q-L or /ˈsiːkwəl/ "sequel" [Structured Query Language]) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables."*

And the most important thing to take away from that is:

*"Do I pronounce it 'ess-que-ell' or 'sequel'?"*

- Technically, it's 'ess-que-ell', but I use both, depending on the context. In my opinion though, that's kinda like asking a developer how to pronounce ".gif" or "JSON".*

# Clear as mud. So...What Is SQL?

- SQL is a standard language you can use to talk to the server where the data is stored:
- It is a “Declarative” language instead of an “Imperative” language.
  - *This is a pretty significant paradigm shift from most programming languages. With SQL, Don't Tell; Ask.*
- SQL is used to get information from your back-end system to be sent to the front-end for display.
  - *The SQL language (and especially the proprietary implementations) have the ability to do much of your business logic, but that is likely better suited for your application.*
- DBMS engines are pretty smart.
  - *They're pretty good at figuring out what you want based on what you have, but they can't read minds. Yet.*

# Databases Are Complex

- Your DBMS does a lot more than just hold data.
- Your DBMS may have a fairly robust Security scheme for managing users or encrypting data.
- Your DBMS likely has the capability for backing your data, or for providing for redundancy.
- Your DBMS likely has its own dialect of SQL that provides for additional functionality not included in the SQL Standard.
- Most SQL is simple CRUD.
  - *“I can make this SQL thing work.”*
    - My queries worked well for a few hundred rows.
    - They didn’t work so well for a few million rows.

Next up...

doh.sql



# Typing Column Names Is Hard

```
SELECT * FROM .....
```

## ■ What is happening?

- *We're selecting all columns from the table.*
- *This usually selects more data than is needed.*
  - That causes the database to work harder
  - That causes the network to work harder
  - That causes the application to work harder
- *This may cause the proper index to be ignored.*
- *This may cause a bad query plan to be generated.*
- *If the underlying column name or order changes, your application might break*

## ■ What should we do?

- *Only SELECT the columns that are needed.*

```
SELECT col1, col2, col3 FROM myTable
```

# I Only Need The First n Rows

```
<cfquery ..... maxrows="42">  
  SELECT cols FROM myTable  
</cfquery>
```

## ■ What is happening?

- *When ColdFusion sends the query to the database, an instruction from the JDBC driver tells the database to stop processing and return records after MAXROWS records.*
  - On the Application side, nothing will be noticed.
  - On the Database side, since this is an instruction from JDBC, the query may continue to process the total number of records from the SELECT.
    - *This may cause the database to ignore the MAXROWS attribute, and process all rows.*
    - *This may cause an incorrect plan to be used.*

## ■ What should we do?

- *This depends on the DBMS used.*
  - **SQL Server: *SELECT TOP 42 col1 FROM myTable***
  - **MySQL: *SELECT col1 FROM myTable LIMIT 42***

# Who Was In Last?

```
<cfquery name="query1">
  INSERT INTO myTable (name)
  VALUES ('Bob')
</cfquery>
<cfquery name="LastInserted">
  SELECT TOP 1 id FROM myTable ORDER BY InsertDate DESC
</cfloop>
```

## ■ What is happening?

- *This pattern has been used to get the id (autogenerated) of the last record inserted into the table.*

## ■ Why is this bad?

- *This query seems to work fine on a single developer machine when that developer is the only one INSERTing data. However, this query can retrieve the wrong id if another INSERT happens between the two queries.*

## ■ What should we do?

- *There are a number of different database-specific ways to query this id, but ColdFusion has a very simple built-in method:*

```
<cfquery name="query1" result="myResult"> INSERT .....</cfquery>
<cfoutput> #myResult.GENERATEDKEY# </cfoutput>
```

# Looping Over A Query

```
<cfloop index="i" from="1" to="42">
  <cfquery name="query1">
    SELECT name FROM myTable1 WHERE id = #i#
  </cfquery>
  <cfoutput>#query1.name# </cfoutput> <br>
</cfloop>
```

- What is happening?
  - *Every loop of the query is creating a new cfquery to get name from myTable1.*
- What should we do?
  - *The same result can be accomplished with a single query and a proper WHERE filter. Then cfloop over the output rather than the query.*
    - This would reduce expensive connections to the database and make the application faster.
- Looping over a query will create a new database connection on each iteration. This can become very expensive and slow down or halt your application.

# I Need To Use One Query For Data In Another Query

```
<cfquery name="query1">
  SELECT id FROM myTable1
</cfquery>
<cfoutput>#query1.id# </cfoutput><br>
<cfoutput query="query1">
  <cfquery name="query2">
    SELECT col1 FROM myTable2 WHERE q1ID = #id#
  </cfquery>
  <cfoutput>#query2.col1# </cfoutput><br>
</cfoutput>
```

## ■ What is happening?

- *This is just another loop over a query. All negatives still apply.*
- *The first query gets id from myTable1. That id is used to retrieve related records from myTable2. The records from myTable2 are displayed as a subsection of query1's id.*
  - This might be used for a nested list or hierarchal information.

## ■ What should we do?

- *More often than not, this can be accomplished with a proper SQL JOIN.*

# SIDEBAR: SQL JOINS

## ■ What is a JOIN?

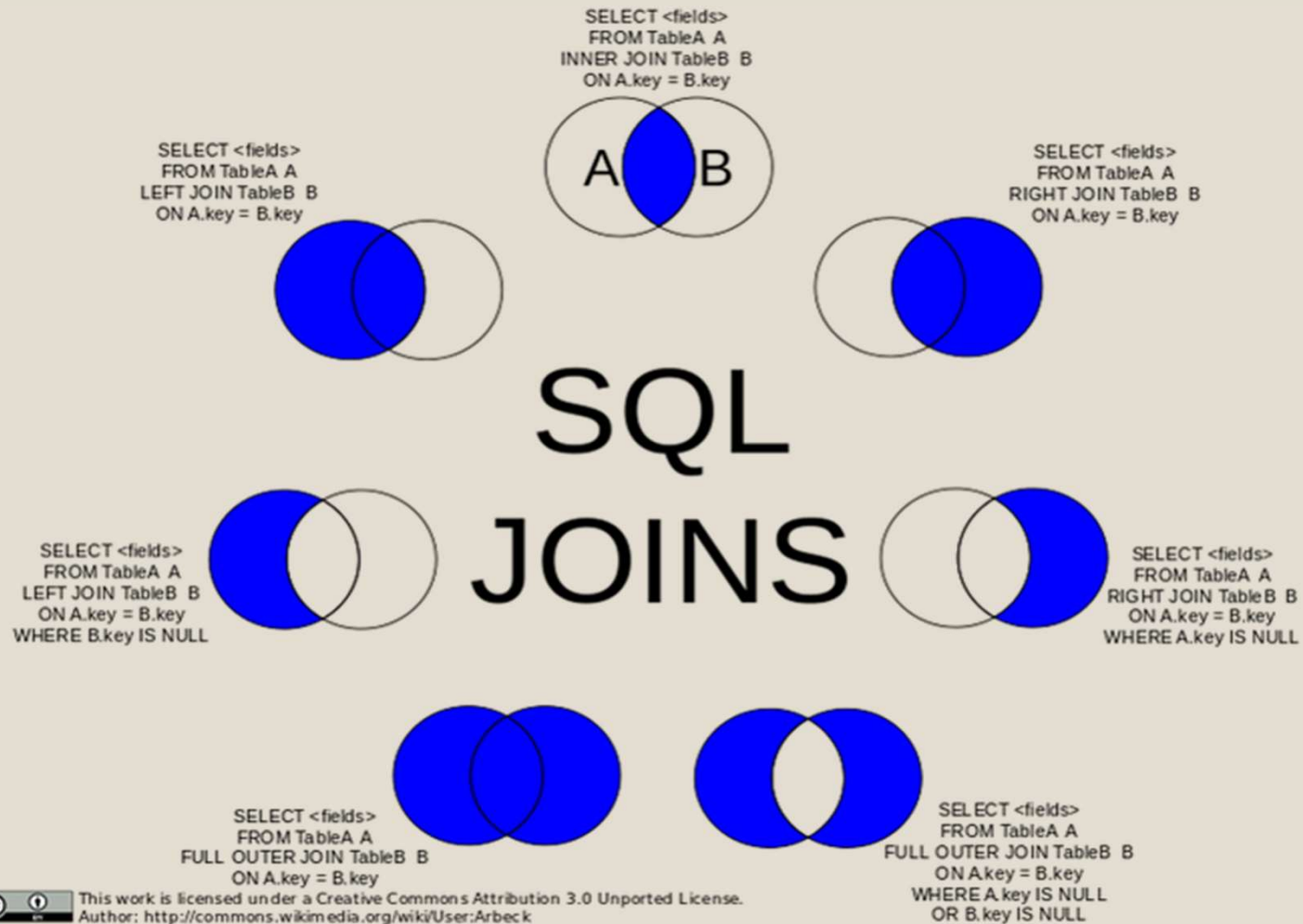
- *It is a way for SQL to relate one or more tables with each other.*
- *The type of JOIN will determine the data available in the final product.*

## ■ There are multiple types of SQL JOINS. A few are:

- *INNER JOIN – This will return only the records that are common between both named tables.*
- *OUTER JOIN – This will return all records from one table and only those that are common from the second table. Unmatched rows will contain NULL values.*
  - *These can be either LEFT OUTER or RIGHT OUTER JOINS.*
- *FULL JOIN – This will return all records from both tables, with matched data for matching rows and NULL data for unmatched rows.*
- *CROSS JOIN – This will return all rows from one table matched to all rows of the second table. This is a CARTESIAN PRODUCT.*

*NOTE: This list is not inclusive, and a ton of information is available online.*

# SIDEBAR: SQL JOINS



# SIDEBAR: SQL JOINS

## ■ My Personal View....

- *The JOIN clause conditions are for linking tables together.*
- *The WHERE clause is for filtering results of the whole query.*

```
SELECT table1.col1 AS t1Col1, table2.col1 AS t2Col1
FROM table1
INNER JOIN table2 ON table1.t2ID = table2.ID
AND table1.t2ProductID = table2.ProductID
AND table2.BuyerID = 42
WHERE table1.CustomerID = 4242
```

## ■ ANSI 89 JOINS

```
SELECT table1.col1 AS t1Col1, table2.col1 AS t2Col1
FROM table1, table2
WHERE table1.t2ID = table2.ID
AND table1.t2ProductID = table2.ProductID
AND table2.BuyerID = 42
AND table1.CustomerID = 4242
```

- **PLEASE DON'T DO THIS.** (ノ益の)ノ≡~~┌┐~~
- *This is an older style of JOINing tables, and creates a Cartesian product before filtering the query results.*
- *It has been deprecated.*



Lookout Below!

SQL has some gotchas!

# My Queries Keep Blocking Each Other

- The **NOLOCK** hint will fix that, right?
  - Yes, *but No.*
  - *NOLOCK probably isn't doing what you think it's doing.*
- The **NOLOCK** hint functions similar to a **READ UNCOMMITTED** transaction lock.
  - *This type of lock will allow for Dirty Reads of data.*
    - This means that a record you just read under a **NOLOCK** hasn't been committed to the database yet, and it may be rolled back. That means your data may not be accurate.
    - This could lead to **Phantom Reads** that give incorrect counts.
    - This should only be used on data that isn't expected to change much.
    - This should also only be used on a **SELECT** statement.
    - **NOLOCK** doesn't keep your query from blocking other queries. It just keeps other queries from blocking this one.
  - *Note that Microsoft SQL Server default is **READ COMMITTED**.*

# Data Type Conversion: 1 vs “1”

- Will those convert to each other?
  - *For these, Yes. For others, It Depends.*
- CFML is a dynamically typed language.
  - *This often allows us to overlook implicit data type conversions.*
- Implicit Conversion happens in SQL, too.
  - *Numbers can be converted to strings. Strings can sometimes be converted to numbers. Dates are sort of stored in the database as number-ish objects, but can't be converted directly to an integer, but they can be converted to a string.*
  - *As with all things, there is some slight overhead, and some things can't be converted to other things.*
  - *When JOINing, filtering with a WHERE, or any other type of comparison, it's important to factor in the data type of the columns or variables you are trying to compare.*

# Data Type Conversion: Can I ...?

| From \ To        | binary | varbinary | char | varchar | nchar | nvarchar | datetime | smalldatetime | date | time | datetimeoffset | datetime2 | decimal | numeric | float | real | bigint | int(INT4) | smallint(INT2) | tinyint(INT1) | money | smallmoney | bit | timestamp | uniqueidentifier | image | ntext | text | sql_variant | xml | CLR UDT | hierarchyid |
|------------------|--------|-----------|------|---------|-------|----------|----------|---------------|------|------|----------------|-----------|---------|---------|-------|------|--------|-----------|----------------|---------------|-------|------------|-----|-----------|------------------|-------|-------|------|-------------|-----|---------|-------------|
| binary           |        | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| varbinary        | ●      |           | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| char             | ●      | ●         |      |         |       |          |          |               |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| varchar          | ●      | ●         | ●    |         |       |          |          |               |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| nchar            | ●      | ●         | ●    | ●       |       |          |          |               |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| nvarchar         | ●      | ●         | ●    | ●       | ●     |          |          |               |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| datetime         | ●      | ●         | ●    | ●       | ●     | ●        |          |               |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| smalldatetime    | ●      | ●         | ●    | ●       | ●     | ●        |          |               |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| date             | ●      | ●         | ●    | ●       | ●     | ●        | ●        |               |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| time             | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             |      |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| datetimeoffset   | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    |      |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| datetime2        | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    |                |           |         |         |       |      |        |           |                |               |       |            |     |           |                  |       |       |      |             |     |         |             |
| decimal          | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| numeric          | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| float            | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| real             | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| bigint           | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| int(INT4)        | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| smallint(INT2)   | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| tinyint(INT1)    | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| money            | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| smallmoney       | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| bit              | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| timestamp        | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| uniqueidentifier | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| image            | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| ntext            | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| text             | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| sql_variant      | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| xml              | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| CLR UDT          | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |
| hierarchyid      | ●      | ●         | ●    | ●       | ●     | ●        | ●        | ●             | ●    | ●    | ●              | ●         | ●       | ●       | ●     | ●    | ●      | ●         | ●              | ●             | ●     | ●          | ●   | ●         | ●                | ●     | ●     | ●    | ●           | ●   | ●       | ●           |

■ Explicit conversion  
● Implicit conversion  
✗ Conversion not allowed  
◆ Requires explicit CAST to prevent the loss of precision or scale that might occur in an implicit conversion.  
○ Implicit conversions between xml data types are supported only if the source or target is untyped xml. Otherwise, the conversion must be explicit.

# Data Types: Character Types

- Likely the most common types of data in your application.
- Also the most easily abused through SQL injection, so pay attention to these.
- Character datatypes are either fixed-width (**char**,**nchar**) or variable-width (**varchar**, **nvarchar**).
- There are also **text** and **ntext**, but these have been deprecated. Use **varchar(max)** or **nvarchar(max)** instead.
- Don't use the Unicode types (**nchar**, **nvarchar**) unless you need them. If you are working with Internationalization, you'll probably need them.

| Data Type           | n      | Actual Storage Size                  | Maximum Value   |
|---------------------|--------|--------------------------------------|---|
| <b>char(n)</b>      | 0-8000 | n bytes                              | Fixed-width. Will pad empty strings to the left.                    |
| <b>nchar(n)</b>     | 0-4000 | n bytes                              | Fixed-width Unicode. A Unicode character requires 2x storage space. |
| <b>varchar(n)</b>   | 0-8000 | n+2 bytes                            | Variable-width. The value won't pad extra characters.               |
| <b>nvarchar(n)</b>  | 0-4000 | (n*2)+2 bytes                        | Variable-width.   |
| <b>varchar(max)</b> |        | Stores up to $2^{31}-1$ bytes (~2GB) | Normal up to 8000bytes, then stored in <b>text</b> .                |

# Data Types: Character Types (caution)

```
CREATE TABLE t1 ( col1 varchar )
```

- In this situation, if you don't specify a length of the `varchar` (i.e. `varchar(50)`), the length will default to **1**.

```
CREATE PROCEDURE mySproc @myvar varchar ...  
DECLARE @val1 varchar ...
```

- These would also default to a length of **1**.

```
SELECT CAST(col1 AS varchar) FROM....
```

- In a Stored Procedure, not declaring a length on a `varchar` will default it to **30** bytes.

*Not specifying a length for a char or varchar may lead to unintentional (bad) or silent (worse) truncation that can be very hard to troubleshoot.*

# Data Types: Character Types (more caution)

```
SELECT ... FROM ... WHERE x = 'foo'
```

- If you are trying to compare a fixed-width column, make sure that you provide the proper width in your comparison.
- In the above example, if *x* is a 5-character `char` data type, then it will never be equal to the 3-character string “*foo*”.

```
SELECT ... FROM ... WHERE x = ' foo'
```

- The above may be what you want.

```
SELECT ... FROM ... WHERE x = ' foo   '
```

- That would also work. Usually.
- The SQL Standard says that when two strings are compared, their lengths must be equal. A string with trailing empty spaces will have those empty spaces ignored.

# Data Types: Exact Numbers - Integers

- Exact numbers will always be represented completely.
- $1 + 2 = 3$
- An integer datatype will be a whole number. When coercing a decimal to an integer, watch out for rounding or truncation.

| Data Type | Storage Size | Minimum Value              | Maximum Value             |
|-----------|--------------|----------------------------|---------------------------|
| tinyint   | 1 byte       | 0                          | 255                       |
| smallint  | 2 bytes      | -32,768                    | 32,767                    |
| int       | 4 bytes      | -2,147,483,648             | 2,147,483,674             |
| bigint    | 8 bytes      | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |



# Data Types: Exact Numbers - Decimals

- A decimal type is still an exact representation of a number.
- $.1 + .2 = .3$
- Decimals are declared like `decimal(P,S)`, where P is Precision and S is Scale.
  - *When declaring a decimal, the default precision is 38 and default scale is 0.*
- Numeric and Decimal datatypes are the same thing.
- Converting from `decimal/numeric` to `float/real` can cause some loss of precision.

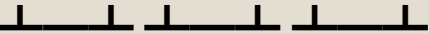
| Data Type      | Storage Size |
|----------------|--------------|
| Decimal(1-9)   | 5 bytes      |
| Decimal(10-19) | 9 bytes      |
| Decimal(20-28) | 13 bytes     |
| Decimal(29-38) | 17 bytes     |

# Data Types: Approximate Numbers

- These are a little more challenging. They aren't always represented completely, and some rounding issues become apparent.
- So  $.1 + .2 = .3$ , right? NOPE. It's  $0.300000000000000004$ .
- (ノ益)ノ ≡ 上上
- *Floats* and *Reals* can store extremely large or small numbers, but don't use them for precise calculations.
- The *real* datatype is essentially a *float(24)*.

| Data Type | n     | Storage Size | Precision | Minimum Value | Maximum Value |
|-----------|-------|--------------|-----------|---------------|---------------|
| Float(n)  | 1-24  | 4 bytes      | 7 digits  | -1.79E + 308  | 1.79E + 308   |
|           | 25-53 | 2 bytes      | 15 digits | -1.79E + 308  | 1.79E + 308   |
| real      | n/a   | 4 bytes      | 7 digits  | -3.40E + 38   | 3.40E + 38    |

# Data Types: Dates. My Favorite.

- Inside a database, a date is not what you see on your screen.
  - *It is usually stored as an integer or other representation of days and/or seconds since the system's epoch, which may be:*
    - 1 JAN 1900 for SQL Server and other Microsoft products
    - 1 JAN 1970 for most Unix-ish systems
    - 31 DEC 1969 for some systems.
    - Sometimes.... (ノ〇益〇)ノ≡ 
- Dates and Date Math will drive you crazy.
  - *I hate dates.*
- However, PLEASE PLEASE PLEASE store a date as a date object and not as a string.
  - *It is much more involved to try to add a day to a string representation of a date.*
  - *It is much more involved to try to find the difference of two dates when they are strings*
  - *It's much more involved to do X when dates are converted to strings.*
    - This holds true for pretty much any language.

# I Thought We Already Played Y2K!

- In the lead-up to the turn of the century, some very smart people worked very hard to ensure that the vast majority of the population had no idea of the scope of the Y2K problem.
  - *We survived, but there are still systems using 2-digit years.*
  - *We dodged one bullet, but we've got another one coming up.*

At **03:14:07 UTC** on the morning of **19 January 2038** we will exceed the number of seconds able to be represented by a signed 32-bit integer.

- *Did you ever think that 2.1 BILLION seconds would run out that fast?*

# Even More Things to Think About

- “RBAR” (pronounced “re bar”): Row By Agonizing Row - Jeff Moden
  - *SQL is designed to be Set-Based.*
    - The engine has to work harder when it does a large operation involving multiple rows if it’s told to do them one at a time.
    - <https://sqlstudies.com/2016/08/17/rbar-vs-batch/>
  - *Functions in the query can also cause RBAR.*
  - ***RBAR IS SLOW***
- CTE: Common Table Expression
  - *Very handy for building up a data set into a temporary table for use within the main query*
  - *Can only be used with SELECT, INSERT, UPDATE, DELETE*
  - *Must be the first statement in a batch. (hint: “;”)*
- Window Functions
  - *Very good way to aggregate data inside of a specific window of data within your query.*

# Even More “More Things to Think About”

## ■ BETWEEN

- *If you need to find results that fall between two points, SQL does have a BETWEEN comparison. However, it's ambiguous.*
- *In SQL Server, BETWEEN is INCLUSIVE of both sides.*
  - ...WHERE x BETWEEN 1 AND 4 will return results that match 1,2,3 and 4.
  - This can be tricky with Dates. How do you return all days in January?
  - It's usually better to be EXPLICIT. Use  $y \geq x$  AND  $y \leq z$  instead.

## ■ The more you say, the better.

- *Be explicit. Don't use shorthand.*
  - What does DATEPART(Y, getDate()) return?
    - *Hint: DATEPART(year, getDate()) might return something different.*

Let Me Explain...

No, There Is Too Much...

Let Me Sum Up...

- SQL IS HARD.
- SQL has become a lot more capable since I started using it.
  - *It gets more complicated every year.*
- SQL is a completely different language, and to do things well requires some deep study of what is happening.
- Be curious.
  
- I have barely scratched the surface of the things I've learned.



# Resources and Other Things to Play With

- There's always RTFM:
  - <https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15>
  - <https://dev.mysql.com/doc/refman/8.0/en/>
  - *Just about every major database system has fairly thorough documentation online.*
- If you want a way to play with different flavors of SQL without installing them:
  - <http://sqlfiddle.com/> < U.S.-Based
    - Has engines for MySQL, Oracle, PostgreSQL, SQLite and MS SQL Server
  - <https://dbfiddle.uk/> < U.K.-Based
    - Has engines for many more databases
      - Db2, Firebird, MariaDB, MySQL, Oracle, PostgreSQL, SQLite and MS SQL Server
    - Because this one is U.K.-based, default date formats are **dd-mm-yyyy**.
- If you want to practice some querying:
  - *SQL Murder Mystery is excellent.*
  - <https://mystery.knightlab.com/>
- Who should you follow:
  - Aaron Bertrand <https://sqlblog.org/bad-habits/>
  - Brent Ozar <https://www.brentozar.com/blog/>
  - Kendra Little <https://littlekendra.com/>
  - Kevin Kline <https://kevinekline.com/>
  - #SQLFamily on Twitter [https://twitter.com/search?q=%23sqlfamily&src=typed\\_query](https://twitter.com/search?q=%23sqlfamily&src=typed_query)
  - *And so many others....*

But If You Take Nothing Else Away  
From This Talk...

➤ DO NOT TEST THINGS IN A  
PRODUCTION DATABASE!

THANK YOU FOR LISTENING  
TO ME JABBER FOR A BIT

Shawn Oden

*@CodeFuMonkey*

*[codefumonkey@gmail.com](mailto:codefumonkey@gmail.com)*

*<https://www.codefumonkey.com>*



I Would Also Like To Extend  
A Big Thank You To Ortus  
Solutions for Into The Box  
and Everything Else That  
They Do!

