ORIE 3120

Lecture 6: SQL #5 [Advanced JOINS]

Announcements

- To get credit for recitations, either
  - Submit answers on Gradescope by 11:59pm Tuesday night
    - Consider using answer sheet
  - Attend recitation, write your netID on the board, and submit answers on Gradescope
    - Answers need not be complete and correct to get full credit, but you do need to try
- To format homework
  - Copy-paste queries into Word or Tex document
  - Add screenshots of output
  - Submit as PDF
  - Screenshots of queries are less likely to get partial credit (b/c they're harder to run)

SQL questions from Piazza

Where should calculated columns go?

- They can be treated like any other column
- They can appear anywhere it is valid for a field name to appear
  - Eg, after a SELECT, after a GROUP BY, after an ORDER BY, ...

Recall: INNER JOIN

```
SELECT Suppliers.SupplierID, ProductName, CompanyName
FROM Products INNER JOIN Suppliers
ON Products.SupplierID = Suppliers.SupplierID
```

For each record in Products:

1. Find all records in Suppliers that match Products.SupplierID = Suppliers.SupplierID
2. Return a joined record for every match

LEFT JOIN

LEFT JOIN

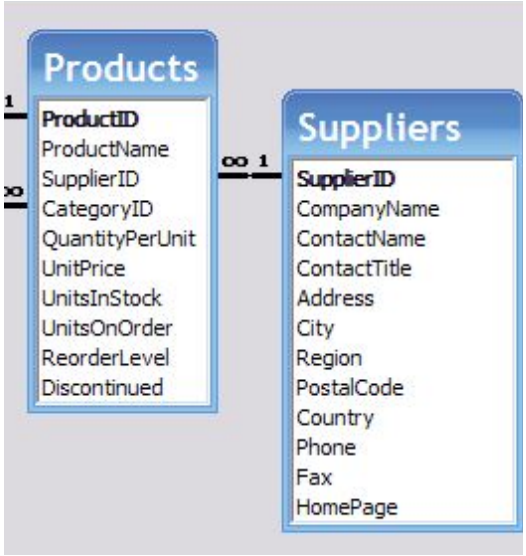
SELECT Products.SupplierID, ProductName, CompanyName

FROM Products **LEFT JOIN** Suppliers

ON Products.SupplierID = Suppliers.SupplierID

For each record in Products:

1. Find all records in Suppliers that match Products.SupplierID = Suppliers.SupplierID
2. Return a joined record for every match
3. **If there are no matches, return a record where the columns from Suppliers are NULL**



Example: LEFT JOIN

Products

ProductName	SupplierID
Aniseed Syrup	1
Chai	1
Chang	1
Chef Anton's Cajun Seasoning	2
Tofu	6

Suppliers

SupplierID	CompanyName
1	Exotic Liquids
2	New Orleans Cajun Delights
3	Grandma Kelly's Homestead

SELECT Products.SupplierID, ProductName, CompanyName

FROM **Products**
LEFT JOIN **Suppliers**

ON Products.SupplierID = Suppliers.SupplierID

SupplierID	ProductName	CompanyName
1	Aniseed Syrup	Exotic Liquids
1	Chai	Exotic Liquids
1	Chang	Exotic Liquids
2	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights
6	Tofu	*NULL*

Example: LEFT JOIN

Products

ProductName	SupplierID
Aniseed Syrup	1
Chai	1
Chang	1
Chef Anton's Cajun Seasoning	2
Tofu	6

Suppliers

SupplierID	CompanyName
1	Exotic Liquids
2	New Orleans Cajun Delights
3	Grandma Kelly's Homestead

SELECT Products.SupplierID, ProductName, CompanyName

FROM **Suppliers**
LEFT JOIN **Products**

ON Products.SupplierID = Suppliers.SupplierID

SupplierID	ProductName	CompanyName
1	Aniseed Syrup	Exotic Liquids
1	Chai	Exotic Liquids
1	Chang	Exotic Liquids
2	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights
NULL	*NULL*	Grandma Kelly's Homestead

RIGHT JOIN

RIGHT JOIN

SELECT Products.SupplierID, ProductName, CompanyName

FROM Products **RIGHT JOIN** Suppliers

ON Products.SupplierID = Suppliers.SupplierID

For each record in **Suppliers** (Suppliers is the table on the right):

1. Find all records in Products that match Products.SupplierID = Suppliers.SupplierID
2. Return a joined record for every match
3. If there are no matches, return a record where the columns from Products are NULL

RIGHT JOIN

- RIGHT JOIN is like left join, except that the role of the tables to the right and left of the “JOIN” keyword are swapped.

- SQLite doesn’t support RIGHT JOIN

- We can achieve the same functionality using LEFT JOIN and swapping the two tables

You can get RIGHT JOIN's functionality in SQLite using a LEFT JOIN

If you want this...

SELECT Products.SupplierID, ProductName, CompanyName

FROM **Products RIGHT JOIN Suppliers**

ON Products.SupplierID = Suppliers.SupplierID

Do this instead…

SELECT Products.SupplierID, ProductName, CompanyName

FROM **Suppliers LEFT JOIN Products**

ON Products.SupplierID = Suppliers.SupplierID

Let’s practice (Q3)

T1

id	a
1	57
2	23
3	9

T2

id	b
1	2
3	11
4	42
5	30
5	56
2	12
1	70

```
SELECT T1.id, T1.a, T2.b
FROM T1
LEFT JOIN T2
ON T1.id = T2.id
```

How many records are returned?

(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

Let’s practice (Q4)

T1

id	a
1	57
2	23
3	9

T2

id	b
1	2
3	11
4	42
5	30
5	56
2	12
1	70

```
SELECT T1.id, T1.a, T2.b
FROM T2
LEFT JOIN T1
ON T1.id = T2.id
```

How many records are returned?

(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

FULL OUTER JOIN

```
SELECT Suppliers.SupplierID, ProductName, CompanyName
FROM Products FULL OUTER JOIN Suppliers
ON Products.SupplierID = Suppliers.SupplierID
```

1. For each record in **Products**:

   Find all records in Suppliers that match Products.SupplierID = Suppliers.SupplierID;

   return a joined record for every match
2. For each unmatched record in **Products**,

   return a record where the columns from Suppliers are NULL
3. For each unmatched record in **Suppliers**:

   return a record where the columns from Products are NULL

Example: OUTER JOIN

Products

ProductName	SupplierID
Aniseed Syrup	1
Chai	1
Chang	1
Chef Anton's Cajun Seasoning	2
Tofu	6

Suppliers

SupplierID	CompanyName
1	Exotic Liquids
2	New Orleans Cajun Delights
3	Grandma Kelly's Homestead

SELECT Products.SupplierID, ProductName, CompanyName

FROM Products

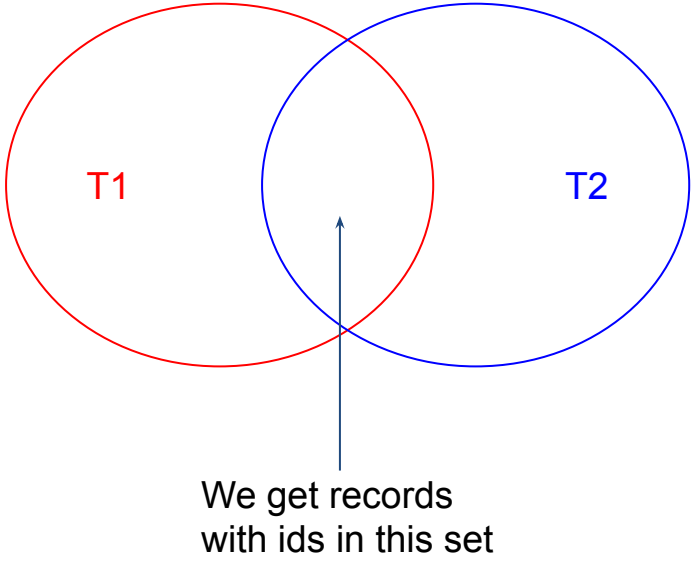
FULL OUTER JOIN Suppliers

ON Products.SupplierID = Suppliers.SupplierID

SupplierID	ProductName	CompanyName
1	Aniseed Syrup	Exotic Liquids
1	Chai	Exotic Liquids
1	Chang	Exotic Liquids
2	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights
NULL	*NULL*	Grandma Kelly's Homestead
6	Tofu	*NULL*

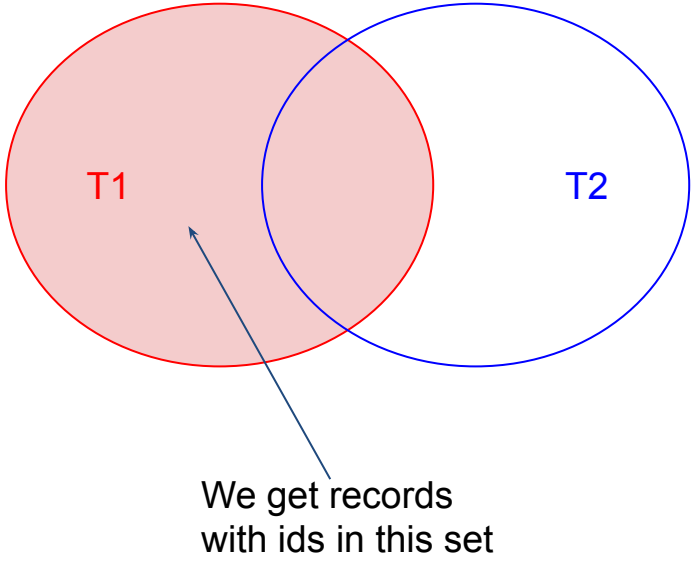
Here’s a way to understand INNER, LEFT, RIGHT, and FULL OUTER JOINS

Suppose we run this query:
SELECT * FROM T1
INNER JOIN T2
ON T1.id = T2.id



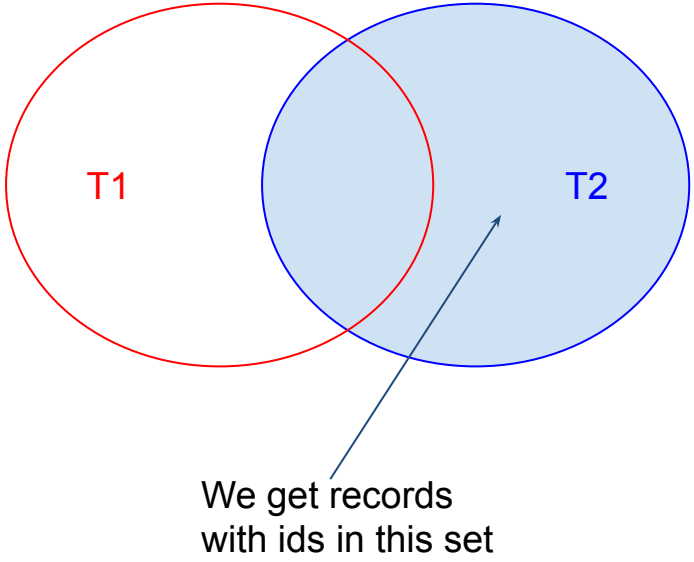
Here’s a way to understand INNER, LEFT, RIGHT, and FULL OUTER JOINS

Suppose we run this query:
SELECT * FROM T1
LEFT JOIN T2
ON T1.id = T2.id



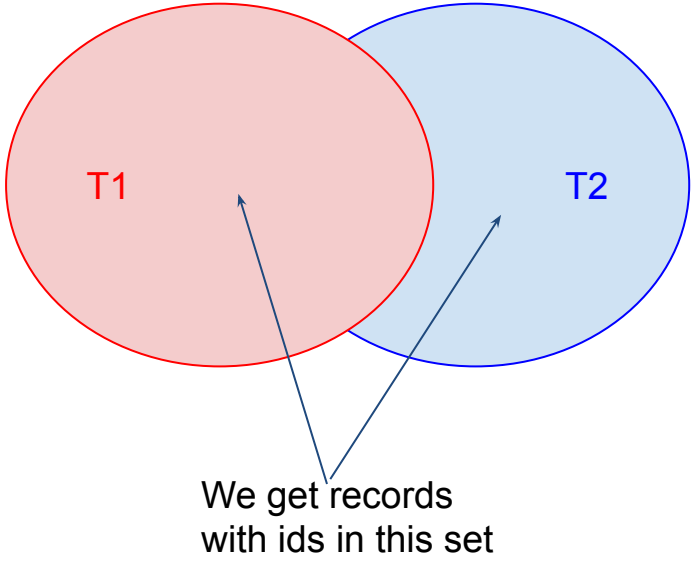
Here’s a way to understand INNER, LEFT, RIGHT, and FULL OUTER JOINS

Suppose we run this query:
SELECT * FROM T1
RIGHT JOIN T2
ON T1.id = T2.id



Here’s a way to understand INNER, LEFT, RIGHT, and FULL OUTER JOINS

Suppose we run this query:
SELECT * FROM T1
FULL OUTER JOIN T2
ON T1.id = T2.id



FULL OUTER JOIN is not implemented in SQLite

That code I just showed you will work in other SQL implementations (MySQL, SQL Server, …), but not in SQLite

But… we can create the same functionality using the UNION keyword

UNION combines the results from two SELECT statements

```
SELECT A,B FROM T
UNION
SELECT C,D FROM S
```

produces a record set with all of the records from the first query, + all of the records from the second, with duplicate records removed.

UNION combines the results from two SELECT statements

SELECT A,B FROM T

UNION **ALL**

SELECT C,D FROM S

produces a record set with all of the records from the first query, + all of the records from the second, **without** removing duplicates.

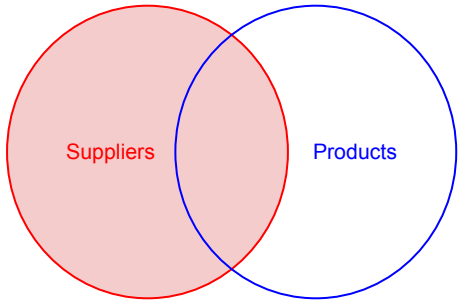
Here’s how to reproduce a FULL OUTER JOIN in SQLite

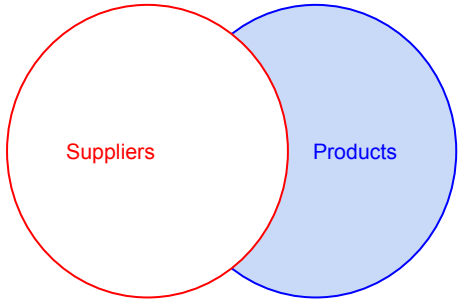
We want this:

```
SELECT ProductName, CompanyName
FROM Products
FULL OUTER JOIN Suppliers
ON Products.SupplierID = Suppliers.SupplierID
```

Write this query instead

```
SELECT ProductName, CompanyName
    FROM Suppliers
    LEFT JOIN Products
    ON Products.SupplierID = Suppliers.SupplierID
UNION ALL
SELECT ProductName, CompanyName
    FROM Products
    LEFT JOIN Suppliers
    ON Products.SupplierID = Suppliers.SupplierID
    WHERE Suppliers.SupplierID IS NULL
```





Let’s practice (Q5)

T1

id	a
1	57
2	23
3	9

T2

id	b
1	2
3	11
4	42
5	30
5	56
2	12
1	70

```
SELECT T1.id, T1.a, T2.b
FROM T2
FULL OUTER JOIN T1
ON T1.id = T2.id
```

How many records are returned?

(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

Let’s practice (Q5)

T1

id	a
1	57
2	23
3	9

T2

id	b
1	2
3	11
4	42
5	30
5	56
2	12
1	70

```
SELECT T1.id, T1.a, T2.b
FROM T2
FULL OUTER JOIN T1
ON T1.id = T2.id
```

How many records are returned?

(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

Let’s practice (Q6)

T1

id	a
1	20
1	57
2	23
3	9

T2

id	b
1	2
3	11
4	42
5	30
5	56
2	12
1	70

```
SELECT T1.id, T1.a, T2.b
FROM T2
FULL OUTER JOIN T1
ON T1.id = T2.id
```

How many records are returned?

(a) 8
(b) 9
(c) 10
(d) 11
(e) 12

Let’s practice (Q6)

T1

id	a
1	20
1	57
2	23
3	9

T2

id	b
1	2
3	11
4	42
5	30
5	56
2	12
1	70

```
SELECT T1.id, T1.a, T2.b
FROM T2
FULL OUTER JOIN T1
ON T1.id = T2.id
```

How many records are returned?

(a) 8
(b) 9
(c) 10
(d) 11
(e) 12

NULLs in comparisons

Keep in mind for WHERE/ON/CASE statements: NULL has tricky behavior in comparisons

- NULL = NULL is false
- NULL <> NULL is false (!= is the same as <>)
- To check whether something is NULL or not, use IS NULL and NOT IS NULL

This has implications for JOINs on fields with NULLs

```
SELECT ProductName, CompanyName
    FROM Suppliers
    LEFT JOIN Products
    ON Products.SupplierID = Suppliers.SupplierID
UNION ALL
SELECT ProductName, CompanyName
    FROM Products
    LEFT JOIN Suppliers
    ON Products.SupplierID = Suppliers.SupplierID
    WHERE Suppliers.SupplierID IS NULL
```

Since NULL = NULL evaluates to false, the records that match this ON clause will never be NULL

As a consequence, we won't duplicate records here

Let’s practice (Q7)

T4

a
4
3
2
1
NULL

T5

b
1
1
1
NULL

```
SELECT * FROM T4
INNER JOIN T5
ON T4.a = T5.b
```

How many records are returned?

(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

Let’s practice (Q7)

T4

a
4
3
2
1
NULL

T5

b
1
1
1
NULL

```
SELECT * FROM T4
INNER JOIN T5
ON T4.a = T5.b
```

How many records are returned?

(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

JOINS on multiple tables

We can join multiple tables

Approach 1: Do it in a sequence of views

Approach 2: Do it in one query

T1

id	a
1	20
1	57
2	23

T2

id	b
1	2
3	11
4	42
5	30
2	12
1	70

T3

id	c
1	-2
2	-11

Approach #1:

```
CREATE VIEW Q1 AS
SELECT T1.id, T1.a, T2.b
FROM T1
INNER JOIN T2
ON T1.id = T2.id

SELECT Q1.id, Q1.a, Q1.b,
T3.c
FROM Q1
INNER JOIN T3
ON Q1.id = T3.id
```

T1

id	a
1	20
1	57
2	23

T2

id	b
1	2
3	11
4	42
5	30
2	12
1	70

T3

id	c
1	-2
2	-11

Approach #2:

```
SELECT T1.id, T1.a, T2.b, T3.c
FROM T1
INNER JOIN T2
ON T1.id = T2.id
INNER JOIN T3
ON T1.id = T3.id
```

id	a	b	c
1	57	2	-2
1	57	70	-2
2	23	12	-11
1	20	2	-2
1	20	70	-2

T1

id	a
1	20
1	57
2	23

T2

id	b
1	2
3	11
4	42
5	30
2	12
1	70

T3

id	c
1	-2
2	-11

Approach #2, alternate syntax:

```
SELECT T1.id, T1.a, T2.b, T3.c
FROM T1, T2, T3
WHERE T1.id = T2.id
AND T1.id = T3.id
```

id	a	b	c
1	57	2	-2
1	57	70	-2
2	23	12	-11
1	20	2	-2
1	20	70	-2

Self JOINS

You can join a table against itself

```
SELECT    M1.Name, M1.MarathonTime,
          M2.MarathonTime AS EqualOrBetterTime
FROM Marathoners AS M1
INNER JOIN Marathoners AS M2
ON M1.MarathonTime >= M2.MarathonTime
```

Marathoners

Name	MarathonTime
Kipsang	2:03:13
Mutai	2:03:13
Kipchoge	2:03:05
Bekele	2:03:03
Kimetto	2:02:57

Name	MarathonTime	EqualOrBetterTime
Kipsang	2:03:13	2:03:13
Kipsang	2:03:13	2:03:13
Kipsang	2:03:13	2:03:05
Kipsang	2:03:13	2:03:03
Kipsang	2:03:13	2:02:57
Mutai	2:03:13	2:03:13
Mutai	2:03:13	2:03:13
Mutai	2:03:13	2:03:05
Mutai	2:03:13	2:03:03
Mutai	2:03:13	2:02:57
Kipchoge	2:03:05	2:03:05
Kipchoge	2:03:05	2:03:03
Kipchoge	2:03:05	2:02:57
Bekele	2:03:03	2:03:03
Bekele	2:03:03	2:02:57
Kimetto	2:02:57	2:02:57

You'll use this trick in the homework to create rankings

Marathoners

Name	MarathonTime
Kipsang	2:03:13
Mutai	2:03:13
Kipchoge	2:03:05
Bekele	2:03:03
Kimetto	2:02:57

Name	MarathonTime	EqualOrBetterTime
Kipsang	2:03:13	2:03:13
Kipsang	2:03:13	2:03:13
Kipsang	2:03:13	2:03:05
Kipsang	2:03:13	2:03:03
Kipsang	2:03:13	2:02:57
Mutai	2:03:13	2:03:13
Mutai	2:03:13	2:03:13
Mutai	2:03:13	2:03:05
Mutai	2:03:13	2:03:03
Mutai	2:03:13	2:02:57
Kipchoge	2:03:05	2:03:05
Kipchoge	2:03:05	2:03:03
Kipchoge	2:03:05	2:02:57
Bekele	2:03:03	2:03:03
Bekele	2:03:03	2:02:57
Kimetto	2:02:57	2:02:57

Name	MarathonTime	Rank
Kimetto	2:02:57	1
Bekele	2:03:03	2
Kipchoge	2:03:05	3
Kipsang	2:03:13	5
Mutai	2:03:13	5

LIMIT

Use the LIMIT keyword to get only the first few results from a query

```
SELECT * FROM Products
ORDER BY UnitsInStock + UnitsOnOrder DESC
LIMIT 5
```

Id	ProductName	SupplierId	CategoryId	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
75	Rhönbräu Klosterbier	12	1	24 - 0.5 l bottles	7.75	125	0	25	0
40	Boston Crab Meat	19	8	24 - 4 oz tins	18.4	123	0	30	0
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25	120	0	25	0
55	Pâté chinois	25	6	24 boxes x 2 pies	24	115	0	20	0
61	Sirop d'érable	29	2	24 - 500 ml bottles	28.5	113	0	25	0

Warning: if the query you LIMIT doesn’t use ORDER BY, the records you get are out of your control

```
SELECT * FROM Products
ORDER BY UnitsInStock + UnitsOnOrder DESC
LIMIT 5
```

Id	ProductName	SupplierId	CategoryId	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
75	Rhönbräu Klosterbier	12	1	24 - 0.5 l bottles	7.75	125	0	25	0
40	Boston Crab Meat	19	8	24 - 4 oz tins	18.4	123	0	30	0
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25	120	0	25	0
55	Pâté chinois	25	6	24 boxes x 2 pies	24	115	0	20	0
61	Sirop d'érable	29	2	24 - 500 ml bottles	28.5	113	0	25	0

CAST

Use the CAST keyword to change the datatype of a field

Table name: Product

	Name	Data type
1	Id	INTEGER
2	ProductName	VARCHAR (8000)
3	SupplierId	INTEGER
4	CategoryId	INTEGER
5	QuantityPerUnit	VARCHAR (8000)
6	UnitPrice	DECIMAL
7	UnitsInStock	INTEGER
8	UnitsOnOrder	INTEGER
9	ReorderLevel	INTEGER
10	Discontinued	INTEGER

```
SELECT    UnitPrice,
          UnitPrice/10,
          CAST(UnitPrice AS Double)/10,
          UnitPrice/10.0,
          1.0*UnitPrice/10
FROM Product
```

UnitPrice	UnitPrice / 10	CAST (UnitPrice AS Double) / 10	UnitPrice / 10.0	1.0 * UnitPrice / 10
18	1	1.8	1.8	1.8
19	1	1.9	1.9	1.9
10	1	1	1	1
22	2	2.2	2.2	2.2
21.35	2.135	2.135	2.135	2.135

Square Brackets

If you have a field or table whose name is the same as a keyword, enclose it in brackets

```
1 SELECT COUNT(*) FROM Order
```

Status

[10:27:45] Error while executing SQL query on database 'Northwind_small (1)': near "Order": syntax error

```
1 SELECT COUNT(*) FROM [Order]
```

[10:28:26] Query finished in 0.000 second(s).

That’s it for SQL