



# Set Operations and Functions in SQL

عملي مشترك

محتوى مجاني غير مخصص للبيع التجاري

## قواعد معطيات 1

16/05/2023

13

RBCs Informatics;

تحدثنا في المحاضرة السابقة عن خواص إضافية يمكن استخدامها مع تعليمة SELECT وتحدثنا أيضاً عن الـ Where (Criteria) وبعض الشروط المنطقية التي يمكن استخدامها معها. وسنتحدث في هذه المحاضرة عن كيفية التعامل مع أكثر من Result Set والعمليات التي يمكن تطبيقها عليهم Set Operations, وسنتحدث أيضاً عن الـ Functions وعن أنواع الـ Functions في SQL وبعض خصائصها.

لنبداً على بركة الله

ليكن لدينا المثال الآتي:

state	city
MA	Boston
DC	Washington
CA	Berkeley
IL	Chicago
TX	Dallas
NULL	München
NY	New York
NULL	Paris

state	city
WA	Seattle
CA	Tustin

select state, city from publishers

select state, city from stores

state	city
MA	Boston
DC	Washington
CA	Berkeley
IL	Chicago
TX	Dallas
NULL	München
NY	New York
NULL	Paris

state	city
WA	Seattle
CA	Tustin

select state, city from publishers

"Don't let yesterday take up too much of today."



■ نلاحظ أن النتيجة السابقة كانت عبارة عن 2 Result Sets وذلك لأننا قمنا بتنفيذ 2 Queries منفصلتين كل على حدى. **ولكن** ماذا لو أردنا معرفة أسماء الولايات والمدن التي تحوي ناشرين والتي تحوي متاجر بيع الكتب... ونريد أن يكون ضمن نفس النتيجة؟!

■ ومن أجل ذلك، أي من أجل التعامل مع 2 Sets فأكثر ظهرت الحاجة إلى ما يسمى ب Set Operations وهي عبارة عن ثلاث عمليات:

Union	الاجتماع
Intersect	التقاطع
Except	الفرق

■ فبالعودة لمثالنا فالواضح من الطلب أن العملية المتوجب علينا استخدامها هي Union. ويمكن تنفيذها على الشكل التالي:

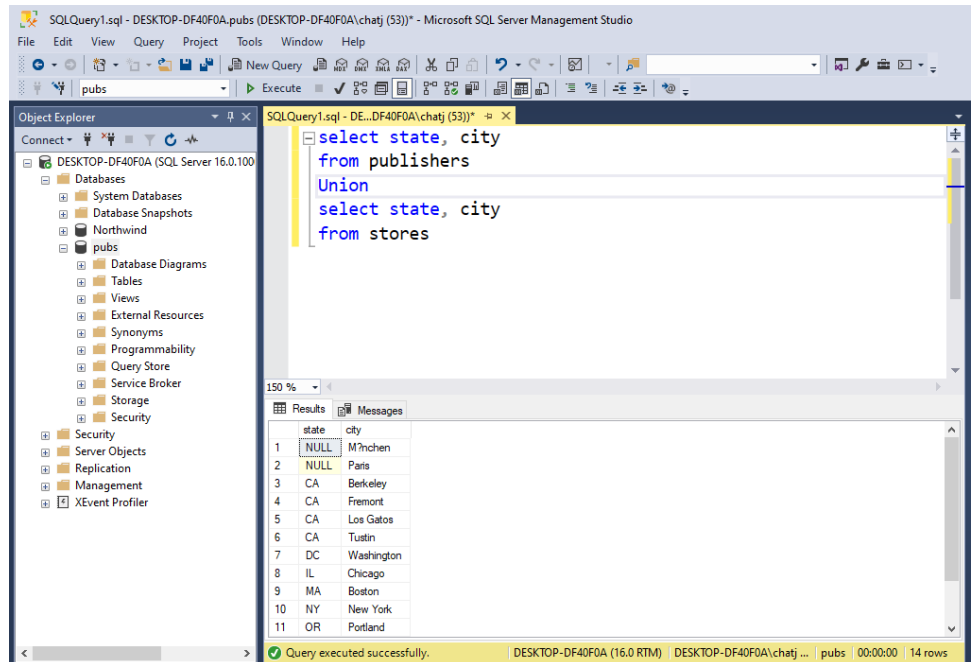
select state, city

from publishers

Union

select state, city

from stores



■ **ملاحظة:** لتطبيق إحدى ال Operations التي تم ذكرها بين 2 Queries فإن هنالك شروط يجب أن تتم مراعاتها:

- أن يكون عدد ال Columns في ال Query الأولى مساوٍ لعدد ال Columns في ال Query الثانية.
- أن يكون ال Datatypes لل Columns في ال Query الأولى نفس ال Datatypes لل Columns في ال Query الثانية.

■ **على سبيل المثال:** إن ال Datatype الخاص بالحقل الأول في ال Query الأولى هو Number, عندئذٍ يجب أن يكون ال Datatype الخاص بالحقل الأول في ال Query الثانية هو Number, وهكذا.....

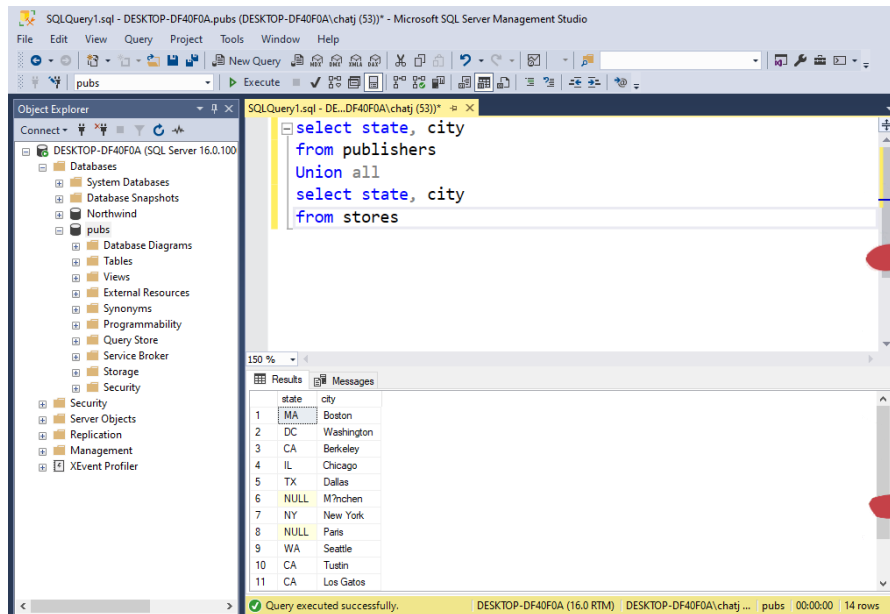
■ **ملاحظة:** ال Columns الناتجة عن أي من هذه ال Operations بين 2 Queries سيكون لها اسم ال Columns الموجودة في ال Query الأولى, وفي حال أردنا تعديل أسماء هذه ال Columns يمكننا استخدام alias وإعطاء اسم لل Column الذي نريد.

■ **مثال:** select state, city as 'common city' from publishers Union select state, city from stores

"If you're going through hell, keep going."

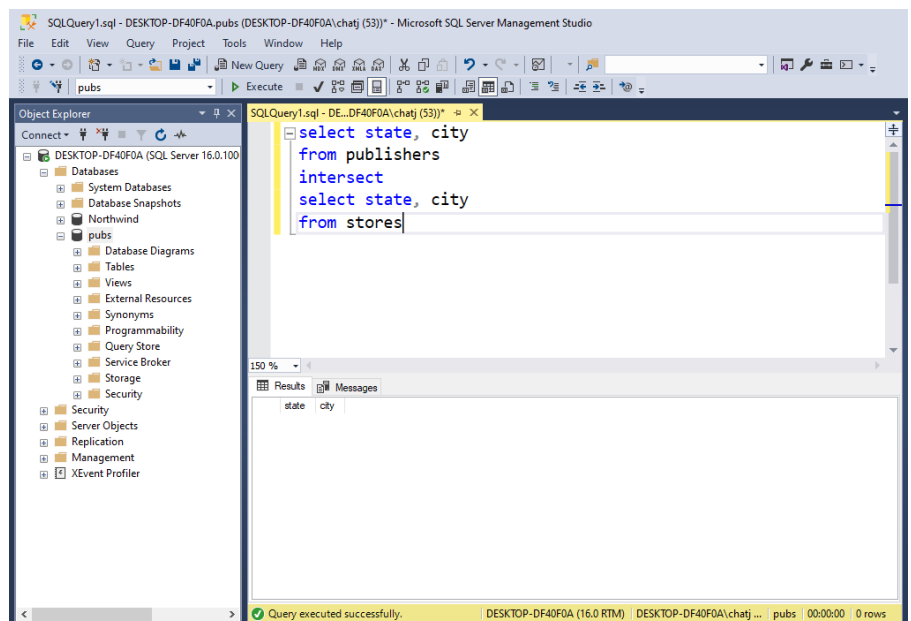
■ **ملاحظة:** إن الـ Union تستخدم الـ distinct في النتيجة التي تعيدها أي أنه إذا كان هنالك تكرارات فعند استخدام الـ Union يتم حذف هذه التكرارات تلقائياً.

■ فلو أردنا على سبيل المثال الحفاظ على هذه التكرارات فإننا في هذه الحالة نستخدم UNION ALL والتي تعيد نتيجة الاجتماع مع الحفاظ على التكرار.



■ لو أردنا على سبيل المثال تنفيذ عملية التقاطع Intersect على هذه الـ 2 Queries:

select state, city  
from publishers  
  
intersect  
  
select state, city  
from stores

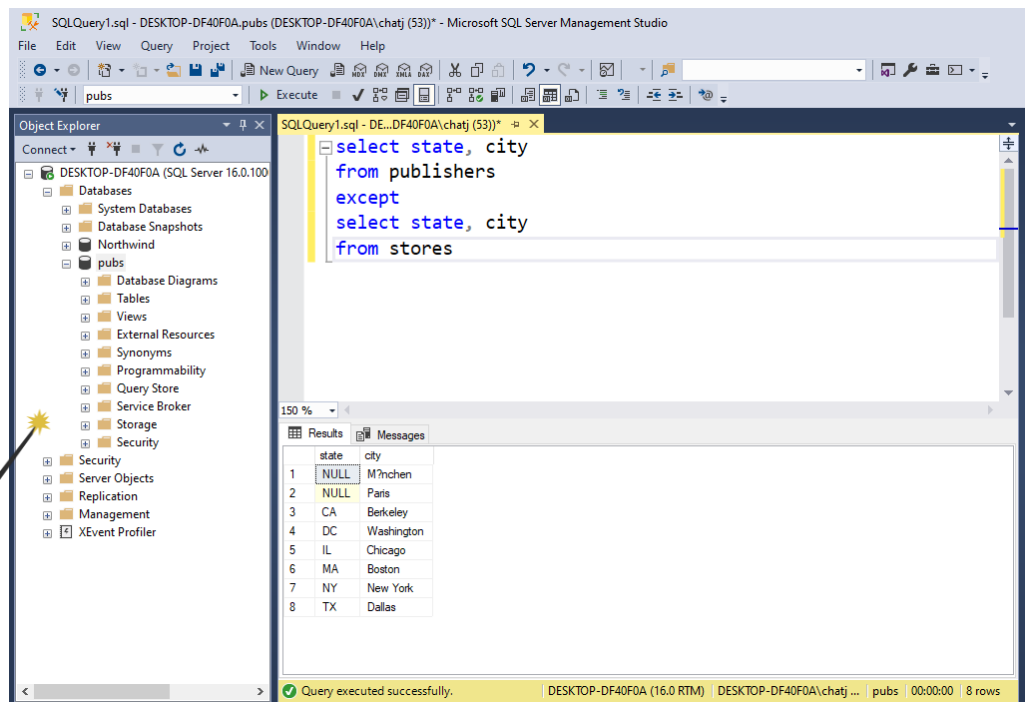


*"Learn as if you will live forever, live like you will die tomorrow."*



- ونلاحظ أن النتيجة فارغة، أي أنه ليس هنالك تقاطع بين هاتين الـ 2 Queries ويجب التنويه أن التقاطع intersect يجب أن يكون على مستوى الـ Query كاملة.
- أي في مثالنا السابق لم يكن هنالك أي record في الـ publishers table مشابه لـ record في الـ stores table بالنسبة للـ (state, city) معاً ولذلك كانت النتيجة فارغة.
- لو أردنا الآن تطبيق الفرق Except:  
 select state, city  
 from publishers  
 except  
 select state, city  
 from stores  
 ومعنى هذه العملية هو أننا نريد الـ records بالـ (state, city) الموجودة في الـ table الأولى وغير موجودة في الـ table الثانية.
- ولو نفذنا التعليمة السابقة تكون النتيجة كالتالي:

select state, city  
 from stores



- **ملاحظة:** إن الـ Records في النتيجة السابقة هم نفس الـ Records في جدول الـ publishers والسبب أنه لا يوجد أي تكرارات لهذه الـ Records في جدول الـ stores. وبالتأكيد حتى نلاحظ آلية عمل هذه الـ operation يجب تنفيذها بين 2 Queries يكون فيها Records مكررة.

## Functions in SQL

بالنسبة للـ Functions فسنحدث عن نوعين للـ Functions والتي تدعمها SQL وهما:

- ❖ Single Row Functions
- ❖ Aggregate Functions

“Life shrinks or expands in proportion to one’s courage.”



## سنبداً الحديث عن ال Single Row Functions:

C1	C2	C3
1	10	Xx
2	15	Yy
3	20	Zz
4	30	Ww
5	40	Ss

■ سنأخذ مثلاً مباشرةً، ليكن لدينا (T1) Table:

■ فلو أردنا على سبيل المثال إحضار النتائج الخاصة بال Column C3 لأول 2 Records ولكن بأحرف كبيرة:

```
select upper(C3)
```

```
from T1
```

```
where C1 in (1, 2)
```

■ فهذا التابع **upper** هو موجود built-in في SQL وظيفته هي تحويل جميع المحارف إلى Capital. وآلية عمله هو أنه يقوم بعمل Fetch لكل Record وينفذ عملية التحويل عليه ولكنه يتم تنفيذه فقط على ال Result Set التي تحقق ال Criteria، وهذا هو مبدأ عمل ال Single Row Functions. وبالتالي سيعيد نتيجة تنفيذ التعليمة السابقة:

XX

YY

■ وبشكل عام فإن ال Functions من نوع Single Row Functions تدعم التعامل مع ثلاث أنواع من ال Datatypes وهم: (Number – Character – Date) ولكل نوع من أنواع ال Datatype هذه له Functions خاصة بالتعامل معه.

**مثال:** ننفذ ال Query التالية:

```
select price, ceiling(price), floor(price), round(price, 1), round(price, -1) from titles
```

وتكون نتيجة تنفيذ هذه ال Query:



فلو لاحظنا ال Result Set في النتيجة نلاحظ أن:

في ال Column الأول أعدد ال price نفسه.

في ال Column الثاني قام بعمل Ceiling لل price ككل وكما نلاحظ أن ال ceiling تقوم بالتقريب للحد الأعلى.

أما في ال Column الثالث قام بعمل Floor وكما نلاحظ أن ال floor معاكسة لل ceiling إذ أنها تقوم بالتقريب للحد الأدنى.

بالنسبة لل Column الرابع والخامس نلاحظ أننا استخدمنا Round وال round تأخذ argument إضافي والذي من خلاله نستطيع تحديد الخانة التي نريد عمل round لها.

**ملاحظة:** طبيعة عمل ال Round أنه إذا كان الجزء المراد تقريبه بين 0. و 4. فإنها تقوم بتقريبه للحد الأدنى، بينما إذا كان بين 5. و 9. فإنها تقوم بتقريبه للحد الأعلى.

ففي ال Column الرابع وعندما أعطينا 1 argument لل Round أي Round(price, 1) فذلك يعني أننا نهتم بتقريب الجزء الذي بعد الفاصلة (نلاحظ ال Record الثامن على سبيل المثال) حيث أن السعر الأساسي كان 22.95 , فالرقم الموجود بعد الفاصلة هو 9 لذلك قام بالتقريب للحد الأعلى فأصبح 23.00 .

بينما عندما كان ال Round(price, -1) والرقم الذي قبل الفاصلة هو 2 لذلك قام بالتقريب للحد الأدنى فأصبح 20.00 .

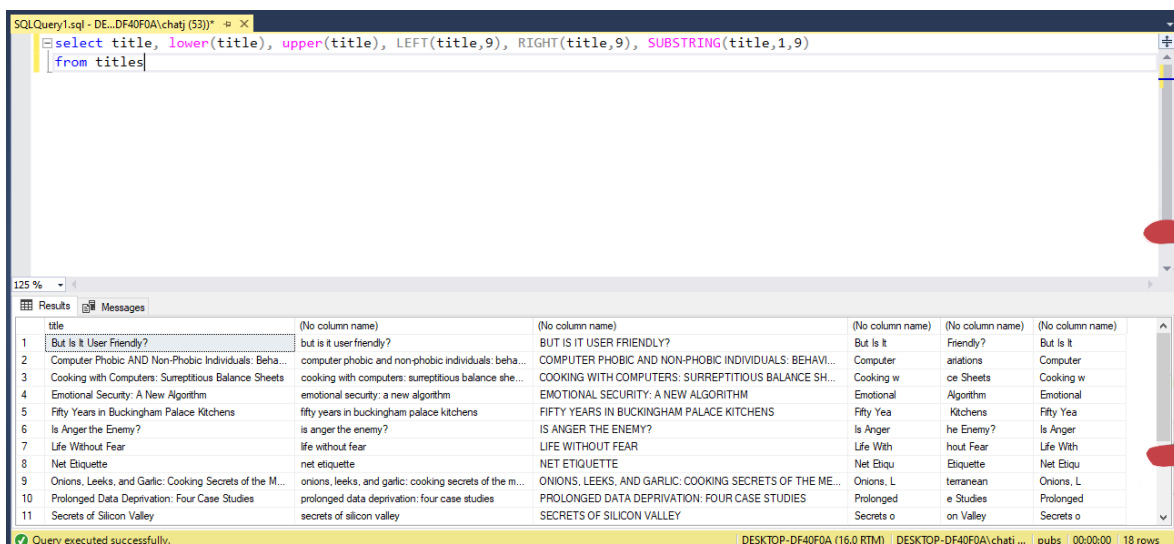
**ملاحظة:** عند استخدام ال Function مع Column ما ضمن تعليمة Select يصبح في النتيجة هذا ال Column ك Calculated Column.

## مثال آخر على ال Functions بالنسبة لل Character:

لنأخذ التعليمة التالية:

```
select title, lower(title), upper(title), LEFT(title, 9), RIGHT(title, 9), SUBSTRING(title, 1, 9)
from titles
```

ويكون نتيجة تنفيذ هذه ال Query:



	title	lower(title)	upper(title)	LEFT(title,9)	RIGHT(title,9)	SUBSTRING(title,1,9)
1	But Is It User Friendly?	but is it user friendly?	BUT IS IT USER FRIENDLY?	But Is It	Friendly?	But Is It
2	Computer Phobic AND Non-Phobic Individuals: Beha...	computer phobic and non-phobic individuals: beha...	COMPUTER PHOBIC AND NON-PHOBIC INDIVIDUALS: BEHA...	Computer	ariations	Computer
3	Cooking with Computers: Surreptitious Balance Sheets	cooking with computers: surreptitious balance she...	COOKING WITH COMPUTERS: SURREPTITIOUS BALANCE SH...	Cooking w	ce Sheets	Cooking w
4	Emotional Security: A New Algorithm	emotional security: a new algorithm	EMOTIONAL SECURITY: A NEW ALGORITHM	Emotional	Algorithm	Emotional
5	Fifty Years in Buckingham Palace Kitchens	fifty years in buckingham palace kitchens	FIFTY YEARS IN BUCKINGHAM PALACE KITCHENS	Fifty Yea	Kitchens	Fifty Yea
6	Is Anger the Enemy?	is anger the enemy?	IS ANGER THE ENEMY?	Is Anger	he Enemy?	Is Anger
7	Life Without Fear	life without fear	LIFE WITHOUT FEAR	Life With	hout Fear	Life With
8	Net Etiquette	net etiquette	NET ETIQUETTE	Net Eliqu	Etiquette	Net Eliqu
9	Onions, Leeks, and Garlic: Cooking Secrets of the M...	onions, leeks, and garlic: cooking secrets of the m...	ONIONS, LEEKS, AND GARLIC: COOKING SECRETS OF THE ME...	Onions, L	terranean	Onions, L
10	Prolonged Data Deprivation: Four Case Studies	prolonged data deprivation: four case studies	PROLONGED DATA DEPRIVATION: FOUR CASE STUDIES	Prolonged	e Studies	Prolonged
11	Secrets of Silicon Valley	secrets of silicon valley	SECRETS OF SILICON VALLEY	Secrets o	on Valley	Secrets o





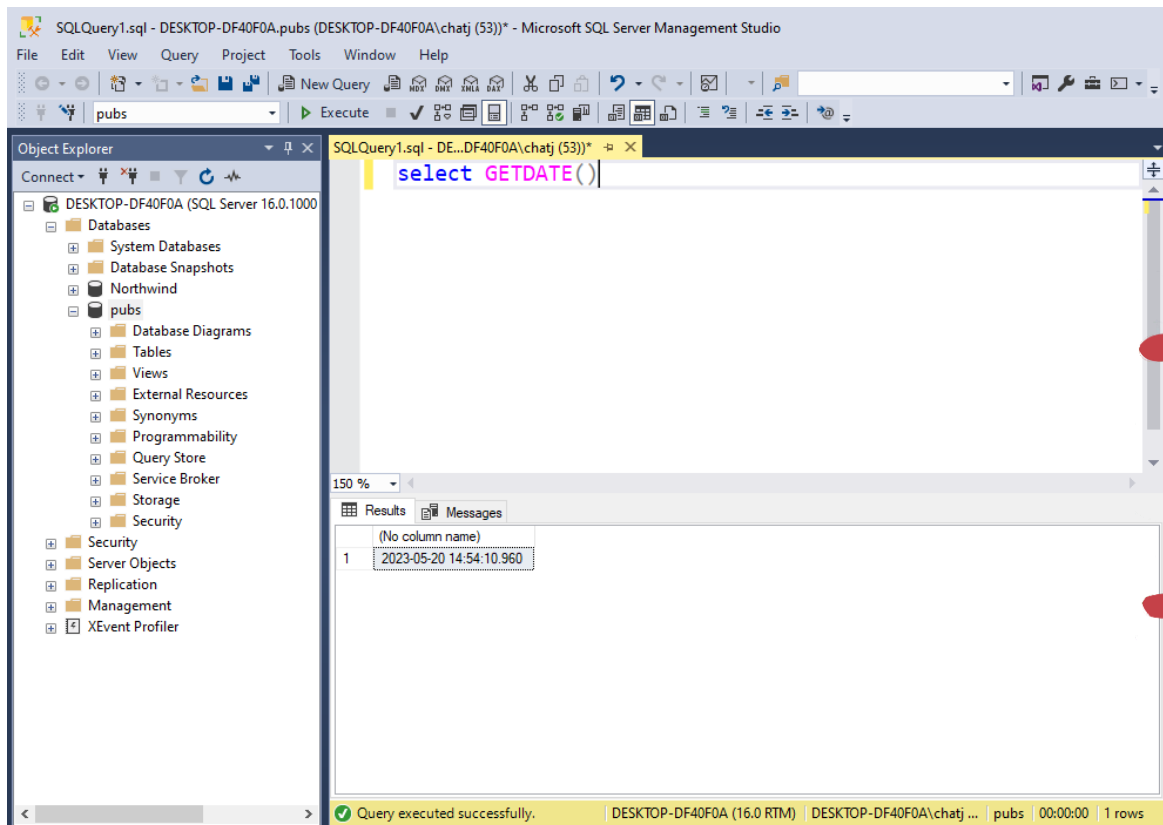
- في الـ Result Set الناتجة نلاحظ أنه تم إعادة الـ title.
- والـ lower قامت بتحويل جميع محارف الـ title إلى أحرف صغيرة Small.
- والـ upper قامت بتحويل جميع محارف الـ title إلى أحرف كبيرة Capital.
- ونلاحظ أن كلاً من الـ LEFT والـ RIGHT والـ SUBSTRING تأخذ parameters إضافية. فبالنسبة لـ LEFT و RIGHT فإن الـ parameter يحدد طول الـ string التي سيعيدها ابتداءً من الـ right أو من الـ left, أما الـ SUBSTRING نستخدم معها 2 parameters لنحدد بداية و طول هذه الـ substring.
- عندما قلنا إن LEFT(title, 9) فإنه سيبدأ من اليسار بطول 9 محارف, وعندما قلنا إن RIGHT(title, 9) فإنه سيبدأ من اليمين وبطول 9 محارف, أما عندما استخدمنا SUBSTRING(title, 1, 9) فنحن حددنا أن يبدأ الـ substring من المحرف الأول وبطول 9 محارف.

■ **ملاحظة:** في الـ LEFT دائماً يكون البدء من الـ position الأول, وفي الـ RIGHT دائماً يكون البدء من الـ position الأخير, أما في الـ SUBSTRING فإننا نستطيع تحديد نقطة البداية التي نريدها.

### مثال آخر على الـ Functions بالنسبة للـ Date:

أول Function وهي الـ GETDATE() والتي يمكننا استخدامها مباشرة ورؤية نتيجتها:

select GETDATE()



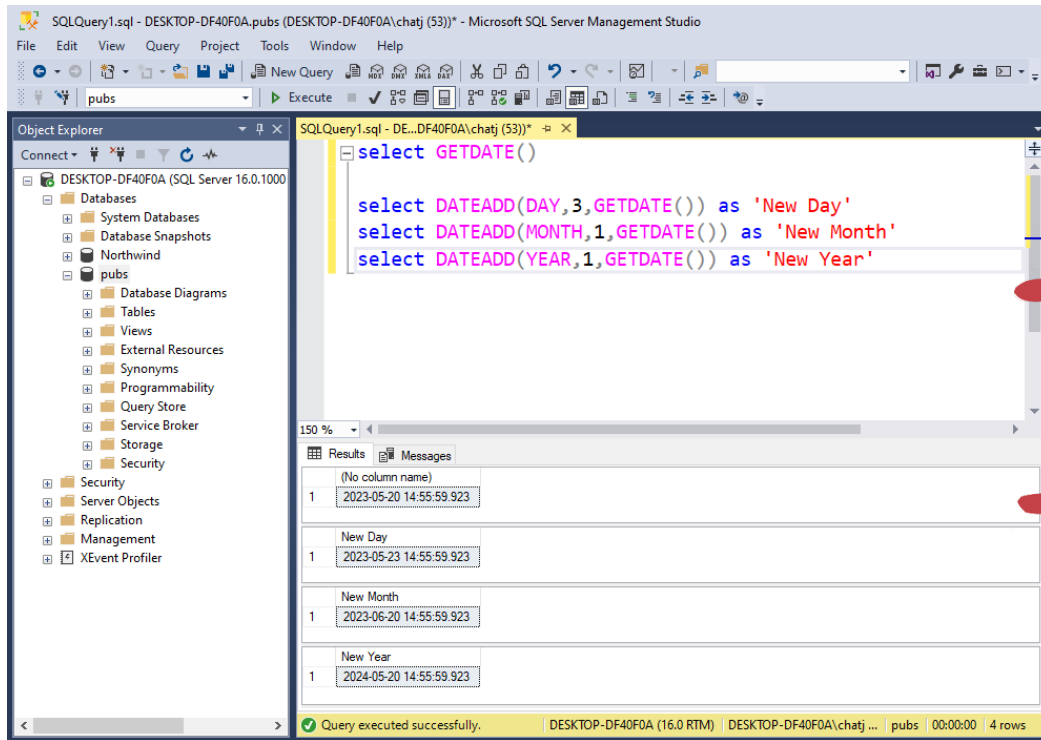
- نلاحظ أن الـ Date مكونة من: (السنة - الشهر - اليوم - الساعة - الدقائق - الثواني).
- سنعرف الآن Function أخرى تدعى الـ DATEADD() والتي يمكننا من خلالها التعديل على نتيجة الـ GETDATE() السابقة والإضافة عليها.

“We need much less than we think we need.”

```
select GETDATE()
select DATEADD(day, 3, GETDATE()) as 'New Day'
select DATEADD(month, 1, GETDATE()) as 'New Month'
select DATEADD(year, 1, GETDATE()) as 'New Year'
```



وتكون النتيجة:



**ملاحظة 1:** إن كل ال Functions التي تحدثنا عنها لحد الآن هي من نوع Single Row Functions.

حيث يتم من خلالها عمل Fetch لكل Record من الوجهة المطلوبة ويتم عند ذلك اختبار فيما إذا كان هذا ال Record يحقق ال Criteria الخاصة بال Query التي نقوم بتنفيذها وعندها يتم تنفيذ هذه ال Function عليه.

**ملاحظة 2:** إن كل ما ذكرناه وسنذكره من أمثلة على ال Functions هي أمثلة بسيطة وقليلة.

إذ أن ال Functions لها طيف واسع جداً وتفصيل كثيرة لا نستطيع أن نذكرها أو نحصيها جميعاً، حتى أنه في كل Version جديدة ل SQL يأتي معها العديد من ال Functions الإضافية والتي لم تكن موجودة مسبقاً، فلو أردنا التوسع فإن هنالك العديد من المصادر والمراجع مثل كتاب SQL يمكننا التوسع فيها. وبالتأكيد فإنه من غير المنطقي أن يتم حفظ ال Functions ، وإنما نقوم بكل بساطة بالبحث عن Function معين تبعاً لل Functionality التي نريد تطبيقها.

*"If things go wrong, don't go with them."*



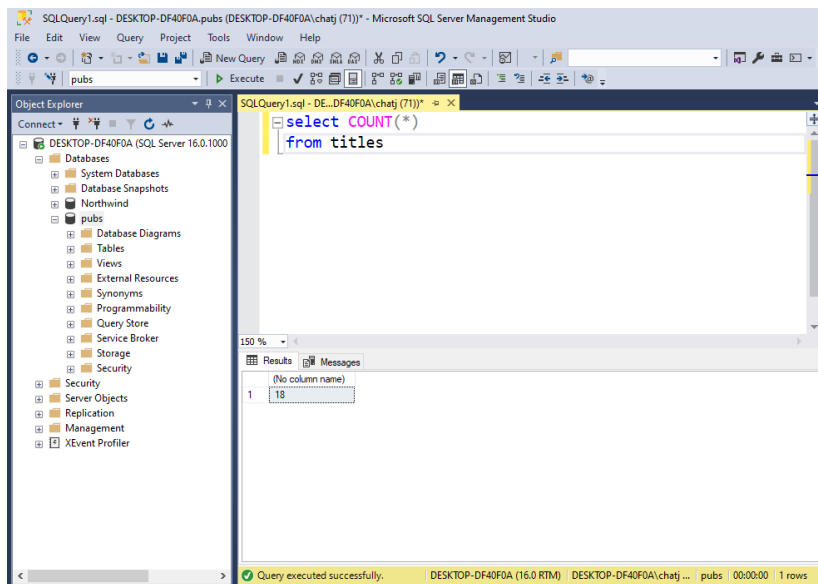
## الآن سنذهب للحديث عن النوع الثاني من الـ Functions وهو Aggregate Functions:

- بالنسبة للـ Aggregate Functions ف فعلياً يوجد هنالك 5 Functions وهم:  
(**COUNT** - **SUM** - **AVERAGE** - **MAX** - **MIN** )
- ولكن ما الفرق في آلية العمل بين الـ Aggregate Function و الـ Single Row Function ؟  
بشكل أساسي فإن الـ Aggregate Function يقوم بالمرور على n record ولكنه يعيد 1 result نتيجة واحدة.

### مثال:

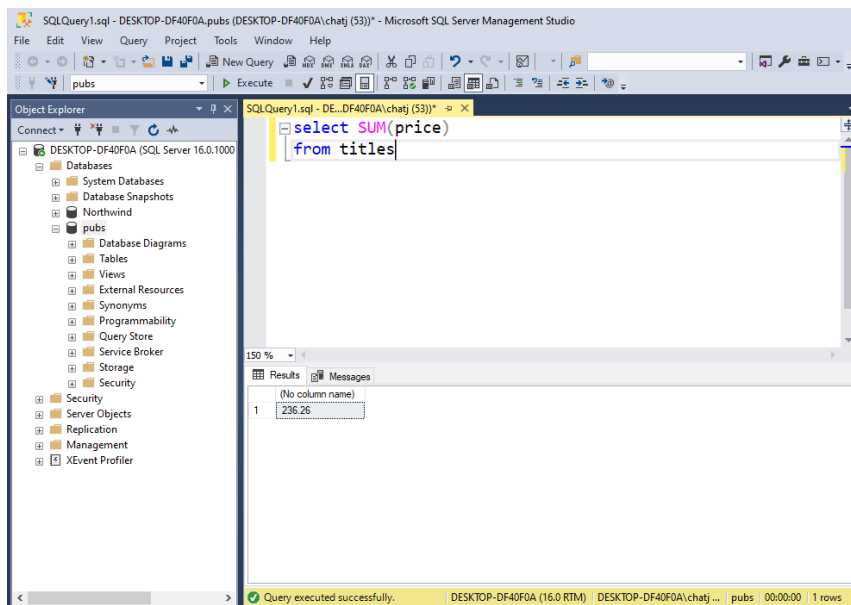
`select COUNT(*) from titles`

فتكون النتيجة:



- إذ أن الـ **COUNT()** هنا قامت بالمرور على جميع الـ Records في الـ titles table وأعادت عددها.

### مثال آخر:



`select SUM(price)`

`from titles`

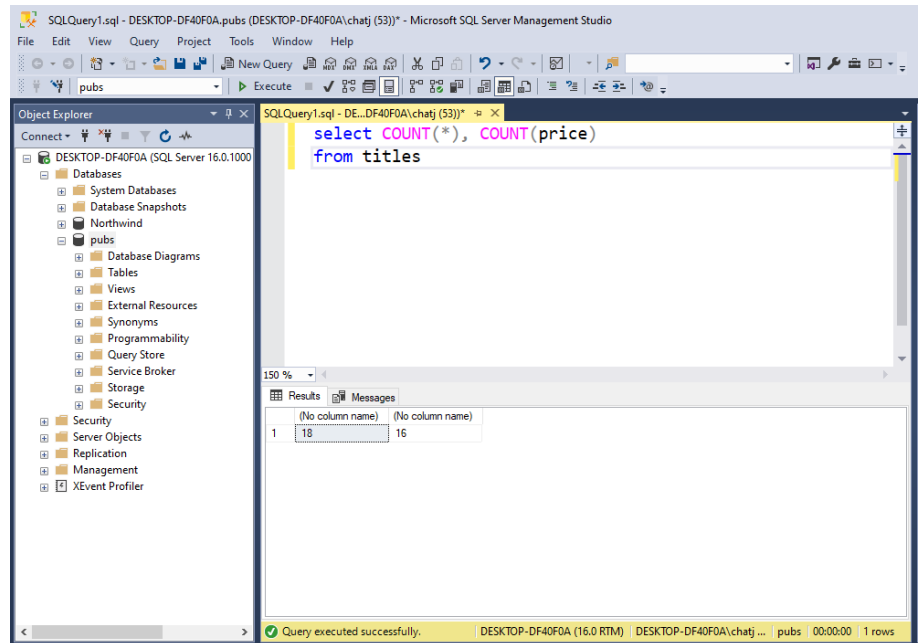
وتكون النتيجة:

■ إذاً أن الـ SUM() هنا قامت بالمرور على كل Record في الـ titles table وجمعت الـ price الخاص به ومن ثم أعادت المجموع النهائي لكل الـ prices.

■ ملاحظة: نستخدم الـ SUM() فقط مع الـ Columns التي لها Data type هو Number.

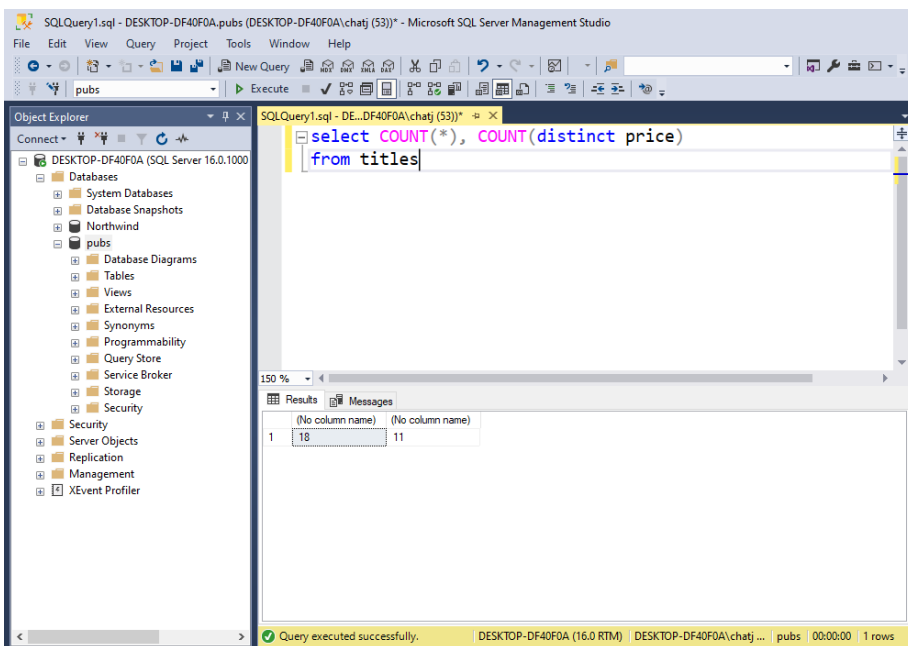
■ لاحظ نتيجة تنفيذ التعليمة التالية:

select COUNT(\*), COUNT(price)  
from titles



ولاحظ الاختلاف في النتيجة حيث أنه عندما نحدد الـ Column عند ذلك يقوم بعد الـ Records التي تحوي value لهذا الـ Column وبالتالي لو عدنا إلى الـ titles table سنجد أن هناك 2 Records, الـ price الخاص بها هو Null.

■ لو استبدلنا التعليمة السابقة بـ:



select COUNT(\*),  
COUNT(distinct price)  
from titles

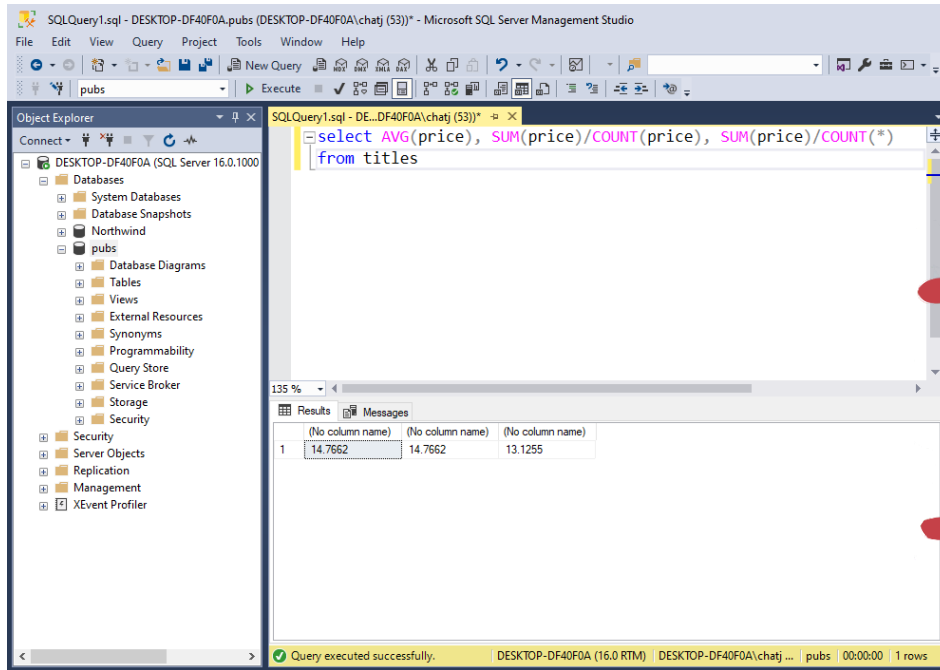
لاحظ أنه عند استعمال distinct قام بعد الـ prices بدون تكرار أي أننا بشكل عام يوجد لدينا 11 سعر مختلف للكتب.

"Everything has beauty, but not everyone sees it."

■ لاحظ التعليمات التالية:

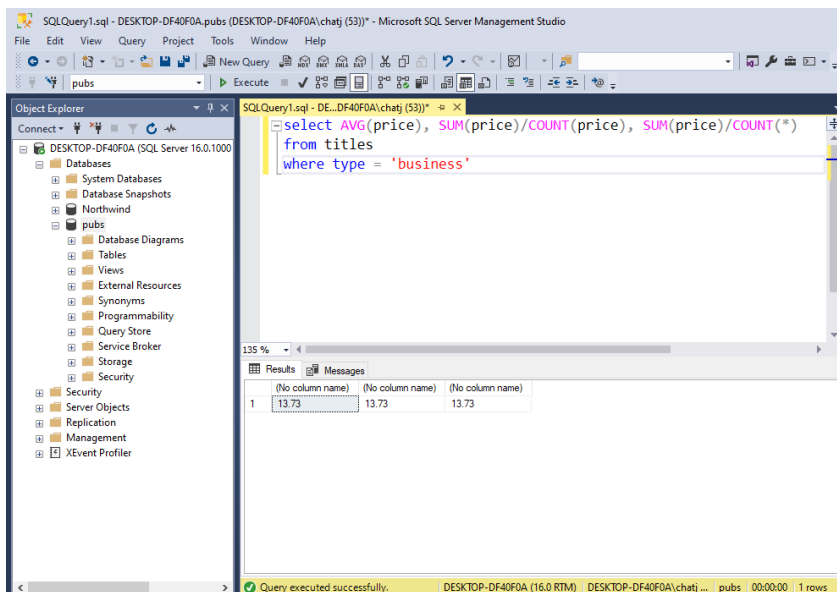
```
select AVG(price), SUM(price)/COUNT(price), SUM(price)/COUNT(*) from titles
```

فإن نتيجتها ستكون كالتالي:



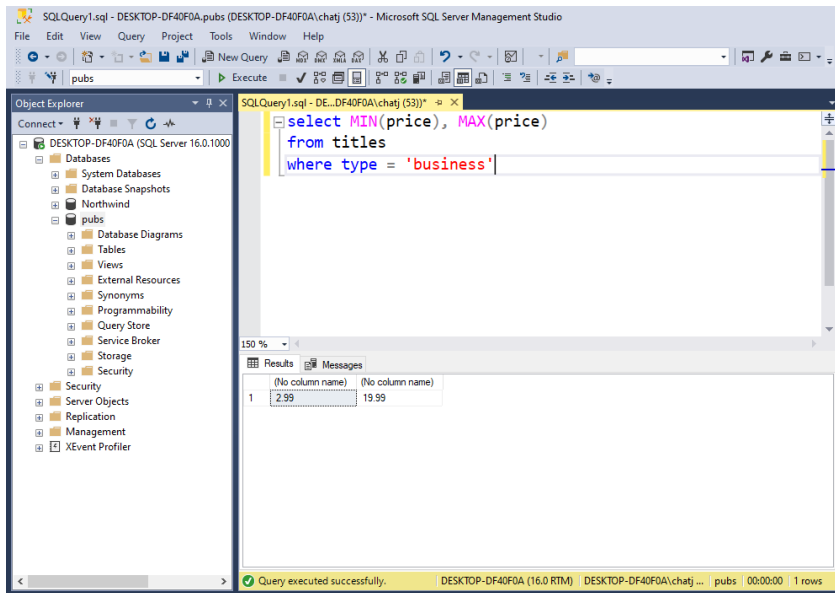
■ نلاحظ أن الـ **AVG()** تقوم بإعادة المتوسط فعند استخدام **AVG(price)** أعادت متوسط سعر الكتب وهي نفس النتيجة التي ستعيدها عندما ننفذ **SUM(price)/COUNT(price)** أي أننا قمنا بتقسيم مجموع أسعار الكتب على عدد الكتب.

■ ولكن من الخطأ أن ننفذ **SUM(price)/COUNT(\*)** لأنه في هذه الحالة سيقوم بالتقسيم على عدد الـ Records بشكل عام ولو كان في هذه الـ Records بعض الـ prices ذي القيمة Null وهذا سيعطي نتيجة خاطئة مختلفة كما لاحظنا.



ويمكننا أن نضيف على التعليمات السابقة مثلاً Criteria:

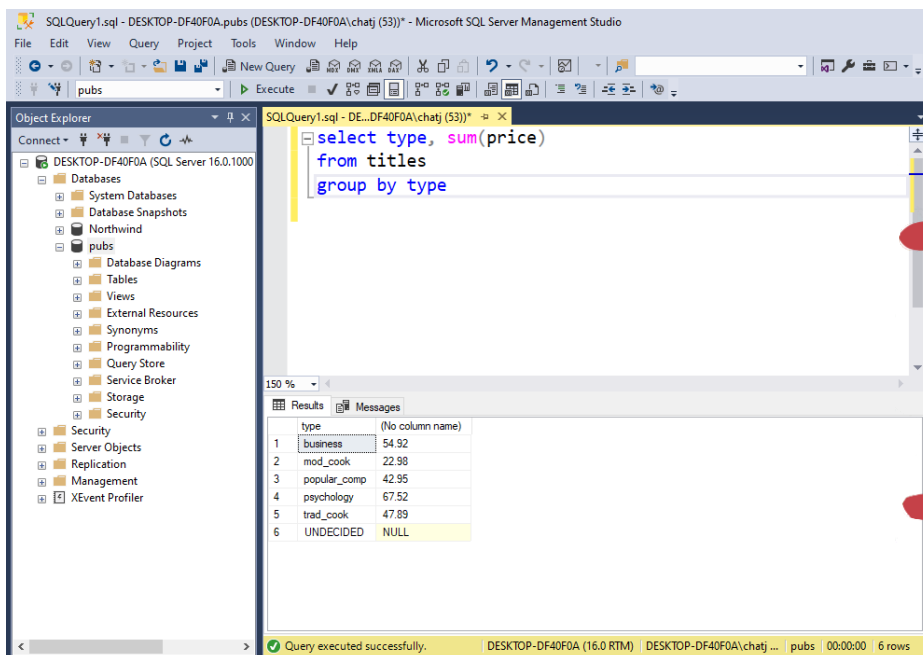
where type = 'business'



يكون نتيجة تنفيذ الكود التالي كالتالي:

```
select max(price), min(price)
from titles
where type = 'business'
```

- لو أردنا على سبيل المثال تصنيف أسعار الكتب ضمن النوع الخاص بها أي في كل Record يكون نوع الكتاب ومجموع أسعار الكتب الموجودة في هذا النوع، ونريد تنفيذ ذلك ضمن Query واحدة. وبالتالي ظهر مفهوم لدعم هذا ال Aggregate Function والذي يدعى GROUP BY.
- فلو نفذنا هذه التعليمات: `select type, SUM(price) from titles GROUP BY type` ستظهر لدينا النتيجة التالية:



- ونلاحظ أنه أعداد ال types ومع كل type قام بتنفيذ ال Aggregate Function على الكتب التي من هذا ال type نفسه.
- فلو قمنا على سبيل المثال بتنفيذ التعليمات السابقة بدون استخدام GROUP BY:

“There is no way to happiness – happiness is the way.”

نلاحظ ظهور خطأ وذلك لأن  
ال type ستعيد 18 records  
بينما ال Aggregate Function  
تعيد قيمة واحدة ولذلك ظهر  
هذا ال Error.

■ ولو أردنا كما في التعليلة السابقة عرض أنواع الكتب مع مجموع أسعار كل نوع **ولكن** نريد عرض فقط الأنواع التي مجموع أسعار كتبها أكبر من 50, **هل يمكننا على سبيل المثال استخدام Where كالتالي؟!**

نلاحظ أنها لم تنفّذ وظهر خطأ وذلك لأن  
ال Criteria أي ال Where يتم تنفيذها عند كل  
Fetch ل Record ففي هذه الحالة نحن نحاول  
اختبار الشرط على ال price ولكن ال SUM() لم  
يتم تنفيذها وحسابها بعد، وهذا هو سبب ظهور  
الخطأ.

■ ولتنفيذ ذلك ظهر مفهوم يدعم ال GROUP BY وهو ال **HAVING** و فعلياً يتم تنفيذها بعد أن تنفّذ ال Query:

## نهاية المحاضرة

