Querying Databases using Transact-SQL

Delegate Guide

QA.COM

Introduction

Introductions

- Name
- Company
- Role
- Data querying experience
- Expectations for the course

ADMIN

- Course timings
- Breaks
- Lunch
- Emergency procedure
- Smoking
- Restrooms
- Phones
- Temperature
- Internet access
- Recycling
- Vending machines
- Diversity

Querying databases using T-SQL

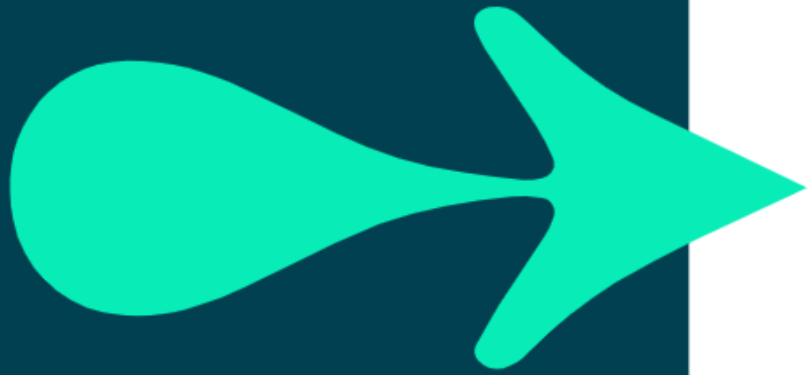
- Introduction to T-SQL
- Retrieving data
- Filtering data
- Sorting returned rows
- Grouping and aggregating data
- Using multiple tables
- Calling views and stored procedures
- Common functions
- Modifying data (appendix)

OBJECTIVES

At the end of this course, you will be able to:

Write an SQL query that:

- Filters unwanted data
- Sorts the results
- Uses built-in functions to calculate some values
- Retrieves data from more than one table



Any Questions?

The Golden Rule:

"There is no such thing as a stupid question"

Corollary to The Golden Rule:

"Even when asked by an instructor"

- Please have a go at answering questions

First amendment to The Golden Rule:

"A question never resides in a single mind"

- By asking questions, you are helping everyone out!

Introduction to Transact SQL

Overview

- Microsoft SQL Server tools
- Introduction to T-SQL
- Hands-on lab

POWERING POTENTIAL

Microsoft SQL Server Tools

MICROSOFT SQL SERVER TOOLS

- Microsoft SQL Server
- SQL Server Management Studio
- Executing queries

- Relational Database Management System (RDBMS)
- Database Engine

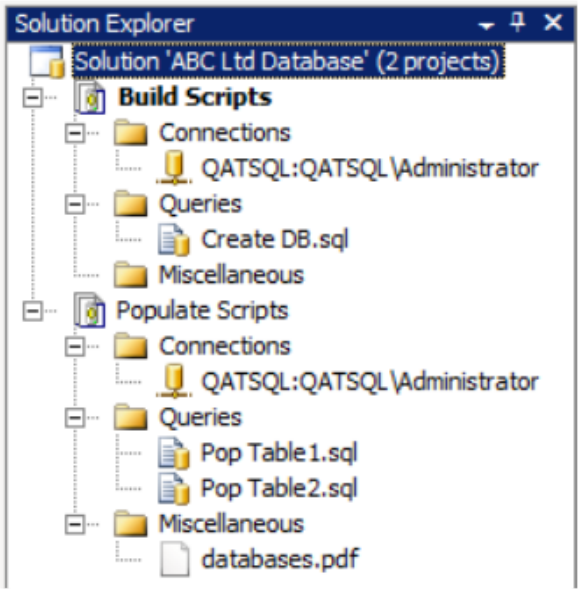
Microsoft SQL Server

SQL SERVER MANAGEMENT STUDIO

- **Introduced with SQL Server 2005**
- **Integrated development and management tool**
  - Previously there were several separate products
- **Central management for SQL Server**
- **Graphical and code management**
  - *Intellisense* editor – code completion
  - Drag and drop from object explorer window
  - Code snippets (introduced in 2012)
- **Solution-based script file management**

SQL Server Solutions

- **A *solution* contains 1 or more *projects***
  - Create an initial project to create the solution
- **A *project* contains files**
- **Add new or existing files to the project**
- **A solution can be added to source control**



EXECUTING QUERIES

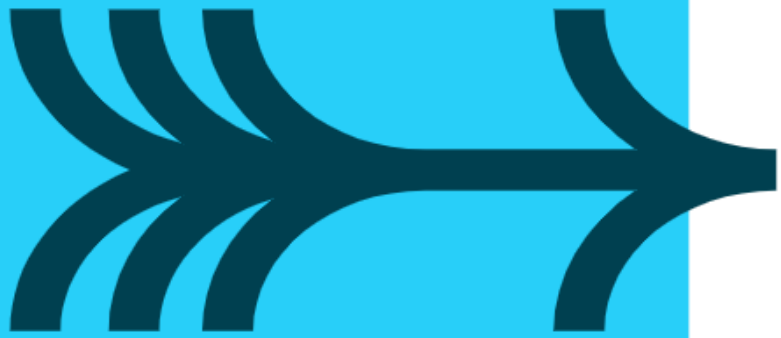
- **Query content**
  - Create queries by interactively entering them in a query window
  - Load a file that contains T-SQL and then execute commands or modify then execute
- **Save queries**
- **Execute queries by:**
  - Selecting the code to run
  - Running the complete script

POWERING POTENTIAL

Introduction to T-SQL

INTRODUCTION TO T-SQL

- What is SQL?
- What is T-SQL?



Hands-on lab

HANDS-ON LAB

- Familiarise yourself with SSMS and the Query Editor

Review

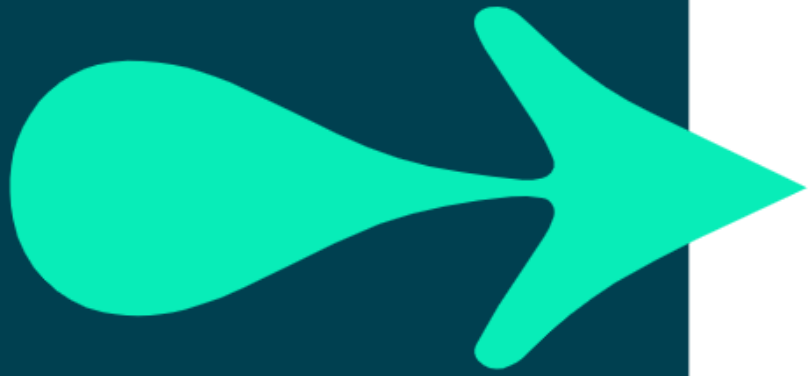
- RDBMSs
- Microsoft SQL Server
- SSMS
- Query Editor

Retrieving Data

Overview

- SELECT statement
- Select lists
- Expressions
- Changing column names
- Best practices
- Hands-on lab

Objectives



At the end of this module you will be able to:

- Write a query that retrieves every column of every row in a table
- Write a query that retrieves only some columns from a table
- Write a query that calculates some of its column values

POWERING POTENTIAL

SELECT statement

SELECT STATEMENT

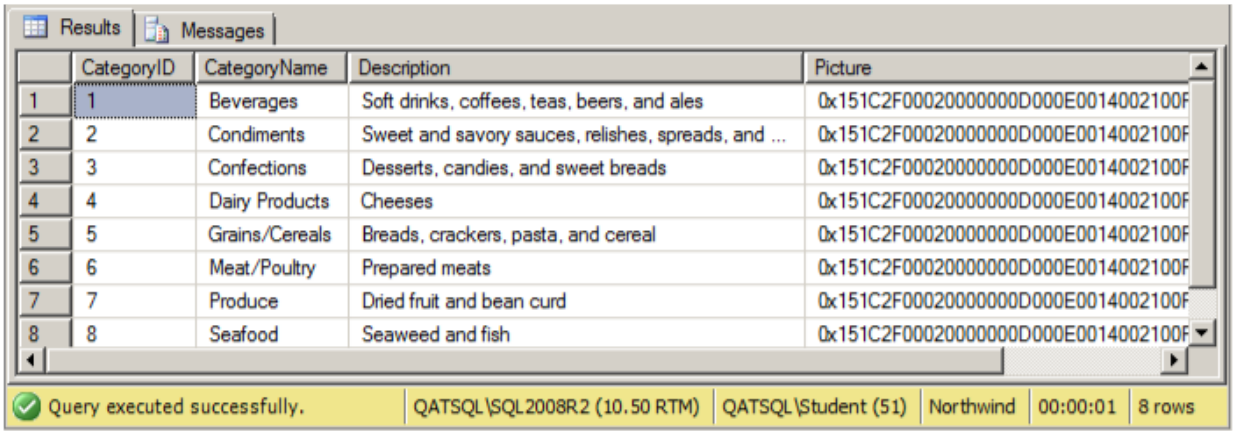
- **SELECT** <<field(s)>>
- **FROM** <<table(s)>>
- WHERE <<condition(s)>>
- GROUP BY <<field(s)>>
- HAVING <<condition(s)>>
- ORDER BY <<field(s)>>

SELECT *

SELECT * FROM *tablename*

Example:

```
SELECT * FROM Categories
```



Results | Messages

	CategoryID	CategoryName	Description	Picture
1	1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x151C2F00020000000D000E0014002100F
2	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x151C2F00020000000D000E0014002100F
3	3	Confections	Desserts, candies, and sweet breads	0x151C2F00020000000D000E0014002100F
4	4	Dairy Products	Cheeses	0x151C2F00020000000D000E0014002100F
5	5	Grains/Cereals	Breads, crackers, pasta, and cereal	0x151C2F00020000000D000E0014002100F
6	6	Meat/Poultry	Prepared meats	0x151C2F00020000000D000E0014002100F
7	7	Produce	Dried fruit and bean curd	0x151C2F00020000000D000E0014002100F
8	8	Seafood	Seaweed and fish	0x151C2F00020000000D000E0014002100F

Query executed successfully. | QATSQL\SQL2008R2 (10.50 RTM) | QATSQL\Student (51) | Northwind | 00:00:01 | 8 rows

POWERING POTENTIAL

Select lists

Select lists

SELECT col*1*, ..., col*n* FROM *tablename*

Example:

```
SELECT CategoryName, Description
FROM Categories
```

Results | Messages

	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and ...
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

QATSQL\SQL2012 (11.0 CTP) | QATSQL\Student (52) | Northwind | 00:00:00 | 8 rows

POWERING
POTENTIAL

Expressions

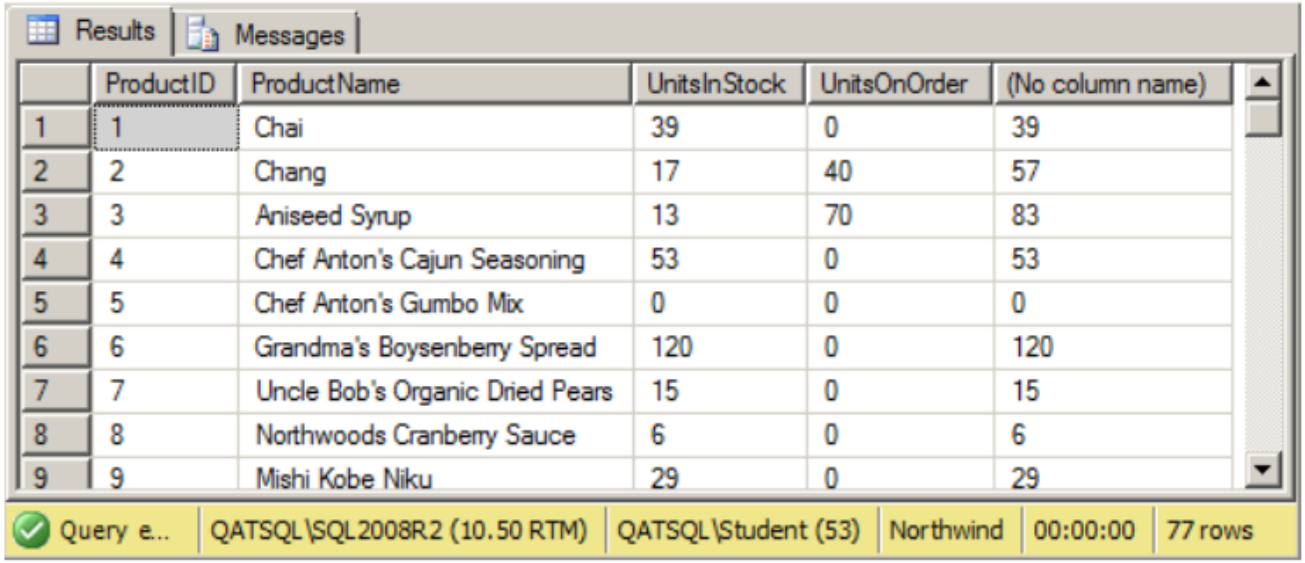
Expressions in select lists

- **“Calculated” values**
- **Expression types:**

Type	Operators
• Arithmetic operators	+, -, *, /, % *e.g. UnitsInStock + UnitsOnOrder*
• String concatenation operator	+ *e.g. FirstName + ' ' + LastName*

Expressions examples (1)

```
SELECT ProductID, ProductName,
       UnitsInStock, UnitsOnOrder,
       UnitsInStock + UnitsOnOrder
FROM Products
```



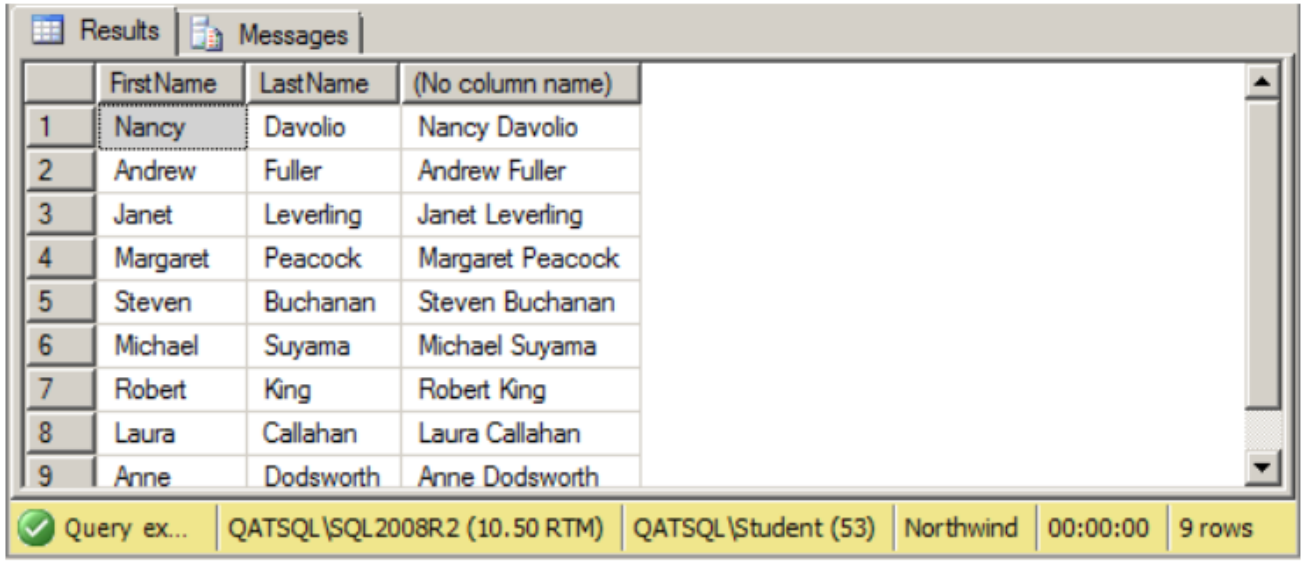
Results | Messages

	ProductID	ProductName	UnitsInStock	UnitsOnOrder	(No column name)
1	1	Chai	39	0	39
2	2	Chang	17	40	57
3	3	Aniseed Syrup	13	70	83
4	4	Chef Anton's Cajun Seasoning	53	0	53
5	5	Chef Anton's Gumbo Mix	0	0	0
6	6	Grandma's Boysenberry Spread	120	0	120
7	7	Uncle Bob's Organic Dried Pears	15	0	15
8	8	Northwoods Cranberry Sauce	6	0	6
9	9	Mishi Kobe Niku	29	0	29

Query e... | QATSQL\SQL2008R2 (10.50 RTM) | QATSQL\Student (53) | Northwind | 00:00:00 | 77 rows

Expressions examples (2)

```
SELECT FirstName, LastName,
       FirstName + ‘ ‘ + LastName
FROM Employees
```



Results | Messages

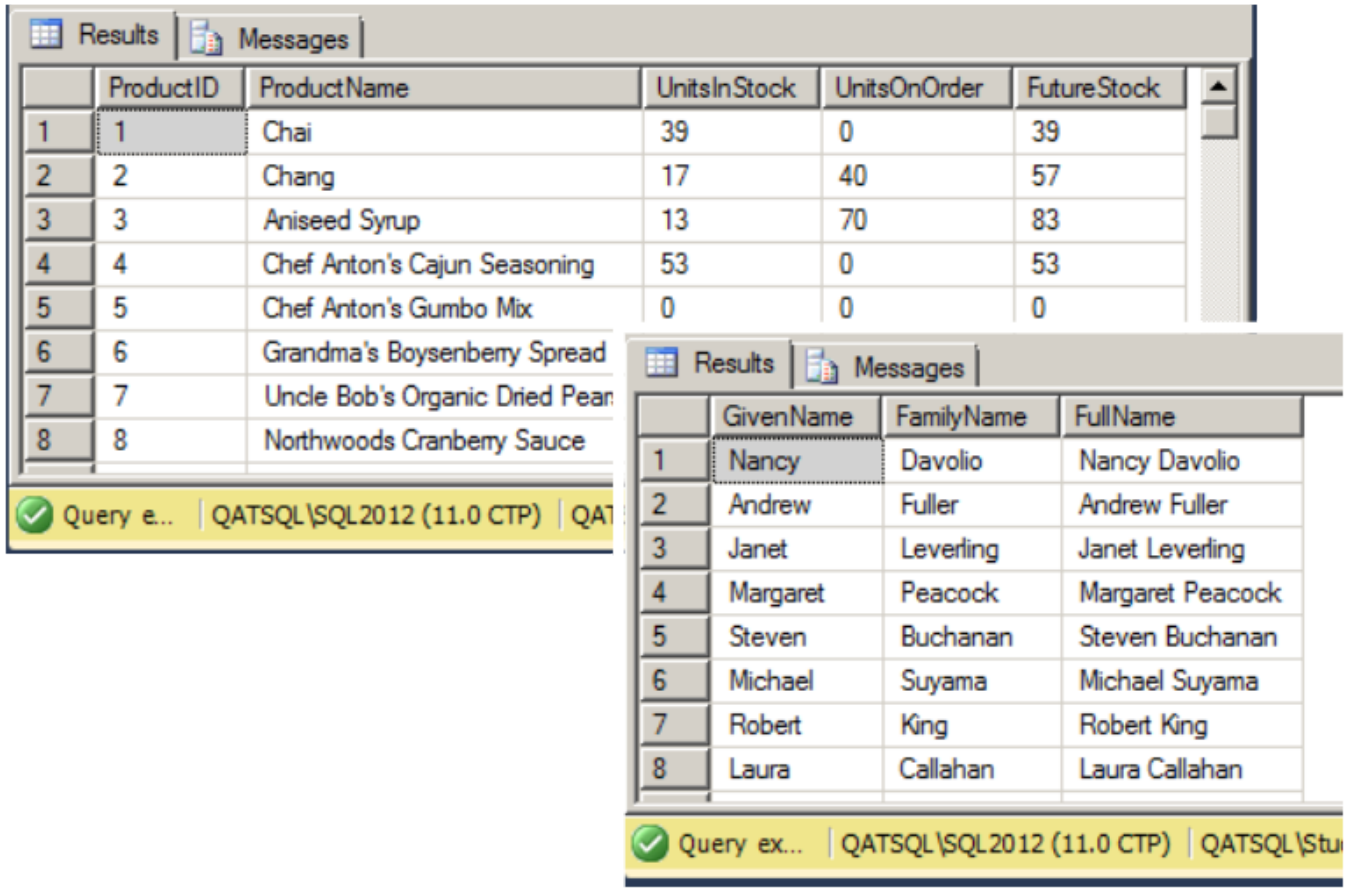
	FirstName	LastName	(No column name)
1	Nancy	Davolio	Nancy Davolio
2	Andrew	Fuller	Andrew Fuller
3	Janet	Leverling	Janet Leverling
4	Margaret	Peacock	Margaret Peacock
5	Steven	Buchanan	Steven Buchanan
6	Michael	Suyama	Michael Suyama
7	Robert	King	Robert King
8	Laura	Callahan	Laura Callahan
9	Anne	Dodsworth	Anne Dodsworth

Query ex... | QATSQL\SQL2008R2 (10.50 RTM) | QATSQL\Student (53) | Northwind | 00:00:00 | 9 rows

POWERING POTENTIAL

Changing column names

Changing column names



Results | Messages

	ProductID	ProductName	UnitsInStock	UnitsOnOrder	FutureStock
1	1	Chai	39	0	39
2	2	Chang	17	40	57
3	3	Aniseed Syrup	13	70	83
4	4	Chef Anton's Cajun Seasoning	53	0	53
5	5	Chef Anton's Gumbo Mix	0	0	0
6	6	Grandma's Boysenberry Spread			
7	7	Uncle Bob's Organic Dried Pear			
8	8	Northwoods Cranberry Sauce			

Query e... | QATSQL\SQL2012 (11.0 CTP) | QAT

Results | Messages

	GivenName	FamilyName	FullName
1	Nancy	Davolio	Nancy Davolio
2	Andrew	Fuller	Andrew Fuller
3	Janet	Leverling	Janet Leverling
4	Margaret	Peacock	Margaret Peacock
5	Steven	Buchanan	Steven Buchanan
6	Michael	Suyama	Michael Suyama
7	Robert	King	Robert King
8	Laura	Callahan	Laura Callahan

Query ex... | QATSQL\SQL2012 (11.0 CTP) | QATSQL\Stu

POWERING POTENTIAL

Best practices

BEST PRACTICES

- Script formatting
- Comments
- Four-part names

Script formatting

- SQL is not case-sensitive
- SQL ignores whitespace

```
SELECT ProductID, ProductName, UnitsInStock, UnitsOnOrder FROM Products
```

- Best practice: BE CONSISTENT!

```
select productid, productname, unitsinstock, unitsonorder from products
```

```
SELECT
ProductID, ProductName, UnitsInStock, UnitsOnOrder
FROM
Products
```

```
SELECT
ProductID,
ProductName,
UnitsInStock,
UnitsOnOrder
FROM
Products
```

```
SELECT
    ProductID,
    ProductName,
    UnitsInStock,
    UnitsOnOrder
FROM
    Products
```

Comments

- **Ignored by SQL**
  - Add documentation to your scripts
- **Two styles:**
  - ```
    SELECT
      UnitsInStock + UnitsOnOrder AS
    FutureStock -- calculate FS
    FROM
       Products
    ```

    and

  - ```
    /*
    This query gets a list of employees
    with their full names
    */
    SELECT FirstName + ' ' + LastName AS Name
    FROM Employees
    ```
- **Best practice: comment your scripts**

FOUR-PART NAMES

Referring to tables in queries:

- `FROM Employees`
- `FROM dbo.Employees`
- `FROM Northwind.dbo.Employees`
- `FROM QATSQL.Northwind.dbo.Employees`

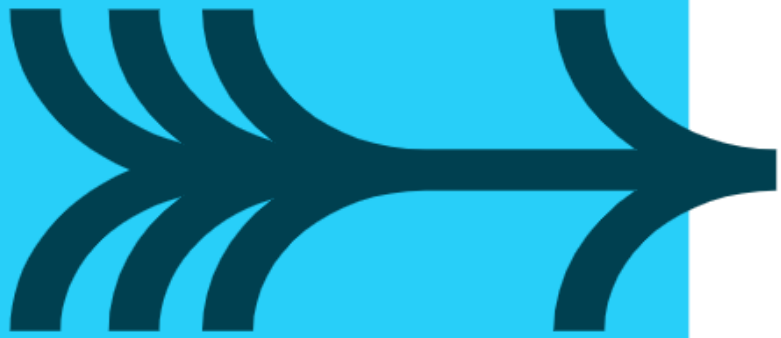
Best Practice: use at least two-part names

POWERING
POTENTIAL

Hands-on lab

HANDS-ON LAB

- Basic SELECT statements
- Expressions in select lists



Review

- SELECT * FROM ...
- SELECT *columns* FROM ...
- SELECT *calculatedColumns* FROM ...
- Comments
- Four-part names

Filtering Rows

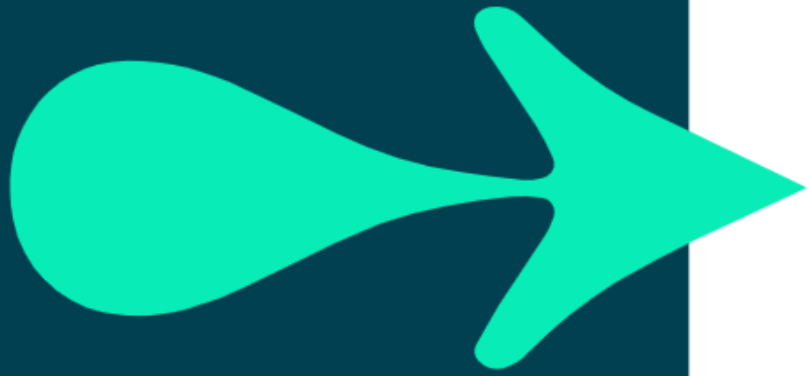
Overview

- WHERE clause
- WHERE expression
- Operators
- Unknown values
- Best practices
- Hands-on lab

Objectives

At the end of this module you will be able to:

- write a query that uses an equality filter
- write a query that use a comparison filter
- write a query that uses the IN and BETWEEN operators
- write a query that looks for text within a column
- write a query that correctly identifies NULL values



POWERING POTENTIAL

WHERE clause

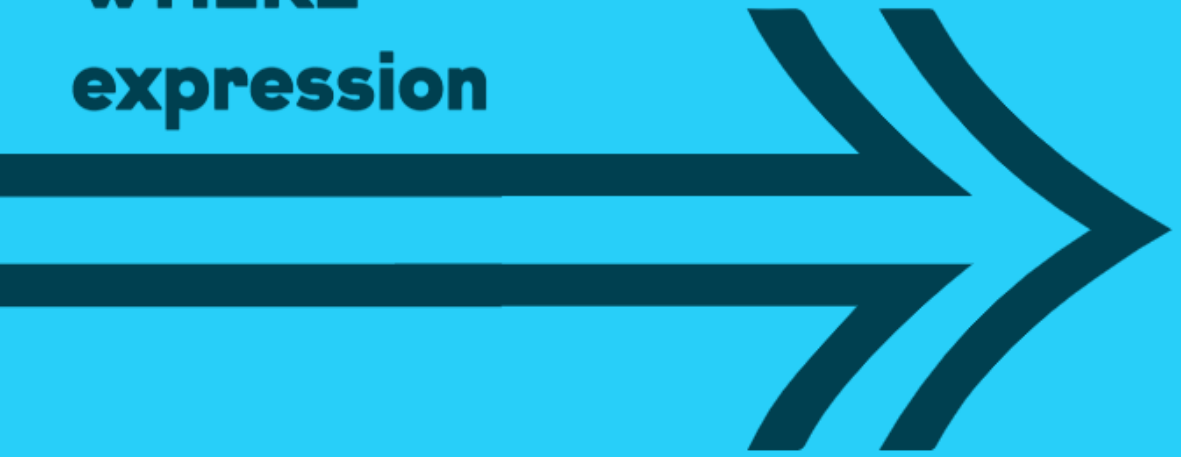
WHERE CLAUSE

```
SELECT <<field(s)>>
FROM  <<table(s)>>
WHERE  <<condition(s)>>
GROUP BY  <<field(s)>>
HAVING  <<condition(s)>>
ORDER BY <<field(s)>>
```

POWERING POTENTIAL

WHERE

expression



WHERE expression

- **Is a predicate**
  - An expression that returns either true or false
- **SQL retrieves all rows that are true**
  - Or possibly none at all
- **Example:**

```
SELECT
  ProductID,
  ProductName,
  UnitsInStock,
  UnitsOnOrder
FROM
  dbo.Products
WHERE
  Discontinued = 0
```

POWERING POTENTIAL

Operators

Operators

Type	Operators
• Comparison operators	• =, <, >, <>, !=, >=, <= *UnitsInStock > 0*
• Logical operators	• AND, OR, NOT *UnitsInStock = 0 AND* *UnitsOnOrder = 0*
• “Special” operators	• BETWEEN, IN, LIKE *OrderDate BETWEEN* *‘2012/01/01’ AND ‘2012/01/31’*

Comparison operators

```
SELECT
  FirstName, LastName,
  City
FROM
  dbo.Employees
WHERE
  Country = 'UK'
```

Results | Messages

	FirstName	LastName	City
1	Steven	Buchanan	London
2	Michael	Suyama	London
3	Robert	King	London
4	Anne	Dodsworth	London

```
SELECT
  ProductID, ProductName,
  UnitPrice
FROM
  dbo.Products
WHERE
  UnitPrice > 100
```

Results | Messages

	ProductID	ProductName	UnitPrice
1	29	Thüringer Rostbratwurst	123.79
2	38	Côte de Blaye	263.50

QATSQL\Student (53) | Northwind | 00:00:00 | 2 rows

Logical operators

```
SELECT
  ProductID, ProductName,
  CategoryID, UnitPrice
FROM
  dbo.Products
WHERE
  CategoryID = 7
  OR CategoryID = 8
  AND UnitPrice > 30
```

Results | Messages

	ProductID	ProductName	CategoryID	UnitPrice
1	7	Uncle Bob's Organic Dried Pears	7	30.00
2	10	Ikura	8	31.00
3	14	Tofu	7	23.25
4	18	Carnarvon Tigers	8	62.50
5	28	Rössle Sauerkraut	7	45.60
6	51	Manjimup Dried Apples	7	53.00
7	74	Longlife Tofu	7	10.00

QL\SQL2012 (11.0 CTP) | QATSQL\Student (54) | Northwind | 00:00:00 | 7 rows

```
...
WHERE
  (CategoryID = 7
  OR CategoryID = 8)
  AND UnitPrice > 30
```

Results | Messages

	ProductID	ProductName	CategoryID	UnitPrice
1	10	Ikura	8	31.00
2	18	Carnarvon Tigers	8	62.50
3	28	Rössle Sauerkraut	7	45.60
4	51	Manjimup Dried Apples	7	53.00

2 (11.0 CTP) | QATSQL\Student (54) | Northwind | 00:00:00 | 4 rows

POWERING POTENTIAL

Unknown values

Specifying a list of values - IN

```
SELECT
  ProductID, ProductName,
  CategoryID, UnitPrice
FROM
  dbo.Products
WHERE
  (CategoryID = 7
  OR CategoryID = 8)
  AND UnitPrice > 30
```

```
SELECT
  ProductID, ProductName,
  CategoryID, UnitPrice
FROM
  dbo.Products
WHERE
  CategoryID IN (7, 8)
  AND UnitPrice > 30
```

Results | Messages

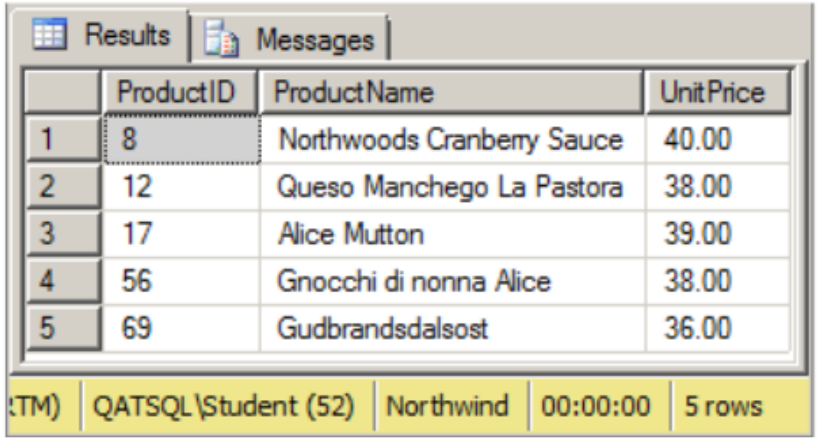
	ProductID	ProductName	CategoryID	UnitPrice
1	10	Ikura	8	31.00
2	18	Carnarvon Tigers	8	62.50
3	28	Rössle Sauerkraut	7	45.60
4	51	Manjimup Dried Apples	7	53.00

2 (11.0 CTP) | QATSQL\Student (54) | Northwind | 00:00:00 | 4 rows

Specifying a range of values - BETWEEN

```
SELECT
  ProductID, ProductName,
  UnitPrice
FROM
  dbo.Products
WHERE
  UnitPrice >= 35
  AND UnitPrice <= 40
```

```
SELECT
  ProductID, ProductName,
  UnitPrice
FROM
  dbo.Products
WHERE
  UnitPrice
    BETWEEN 35 AND 40
```



String comparisons - LIKE

```
SELECT
  FirstName, LastName,
  Title
FROM
  dbo.Employees
WHERE
  Title LIKE 'Sales%'
```

Results | Messages

	FirstName	LastName	Title
1	Nancy	Davolio	Sales Representative
2	Janet	Leverling	Sales Representative
3	Margaret	Peacock	Sales Representative
4	Steven	Buchanan	Sales Manager
5	Michael	Suyama	Sales Representative
6	Robert	King	Sales Representative
7	Anne	Dodsworth	Sales Representative

QATSQL\Student (54) | Northwind | 00:00:00 | 7 rows

```
SELECT
  FirstName, LastName,
  Title
FROM
  dbo.Employees
WHERE
  Title LIKE '%sales%'
```

Results | Messages

	FirstName	LastName	Title
1	Nancy	Davolio	Sales Representative
2	Andrew	Fuller	Vice President, Sales
3	Janet	Leverling	Sales Representative
4	Margaret	Peacock	Sales Representative
5	Steven	Buchanan	Sales Manager
6	Michael	Suyama	Sales Representative
7	Robert	King	Sales Representative
8	Laura	Callahan	Inside Sales Coordinator
9	Anne	Dodswo...	Sales Representative

CTP) | QATSQL\Student (54) | Northwind | 00:00:00 | 9 rows

Filtering on calculated values

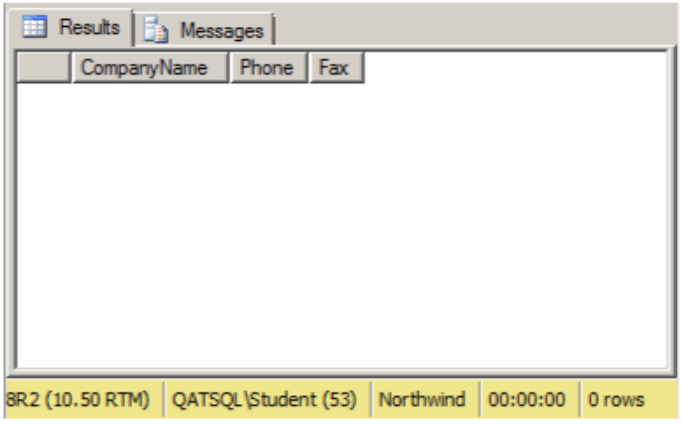
Reuse the expression in the WHERE clause

- Has to be evaluated for every row

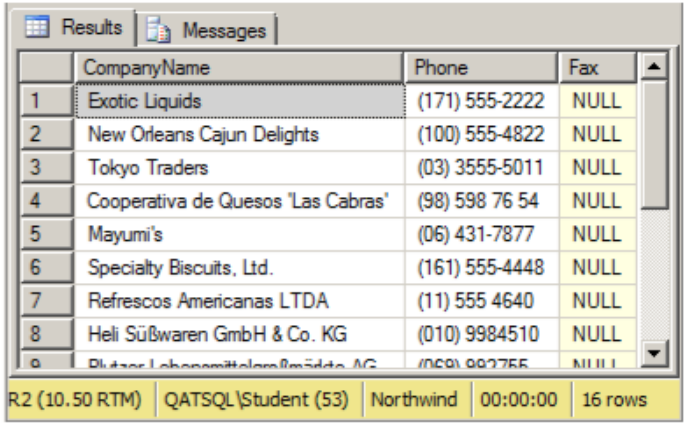
```
SELECT
  ProductID,
  ProductName,
  UnitsInStock + UnitsOnOrder AS FutureStock
FROM
  dbo.Products
WHERE
  --FutureStock < 100 --won't work!
  --use:
  (UnitsInStock + UnitsOnOrder) < 100
```

Finding unknown values (NULLs)

```
SELECT
  CompanyName, Phone,
  Fax
FROM
  dbo.Suppliers
WHERE
  Fax = NULL
```




```
SELECT
  CompanyName, Phone,
  Fax
FROM
  dbo.Suppliers
WHERE
  Fax IS NULL
```



POWERING POTENTIAL

Best practices

Best practices

Avoid NOTs

- Including <>

Use brackets around complex expressions

- This also makes them easier to read

Use IN and BETWEEN

- Rather than lots of ORs and ANDs

Avoid leading wildcards

- Inefficient

Use IS NULL

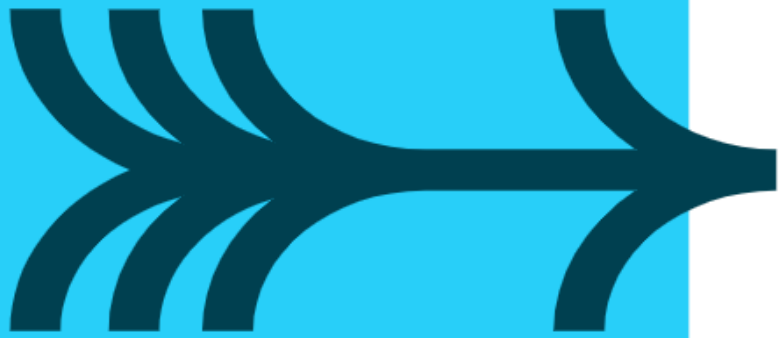
- Rather than = NULL

POWERING
POTENTIAL

Hands-on lab

HANDS-ON LAB

- Basic equality filters
- Basic comparisons
- Logical comparisons
- String comparisons
- NULL comparisons



- WHERE clause
- Comparisons
- IN, BETWEEN, LIKE
- Filtering on calculated expressions
- Working with NULLs

Review

Sorting Rows

Overview

- ORDER BY clause
- Sort order
- Sort on multiple columns
- Sort on expressions
- Eliminate duplicates - DISTINCT
- Limit results - TOP
- Hands-on lab

OBJECTIVES

At the end of this module you will be able to:

- write a query that sorts on a single column
- write a query that sorts on multiple columns
- write a query that sorts on a calculated column
- write a query that selects unique values
- write a query that returns only a specified number of rows

POWERING POTENTIAL

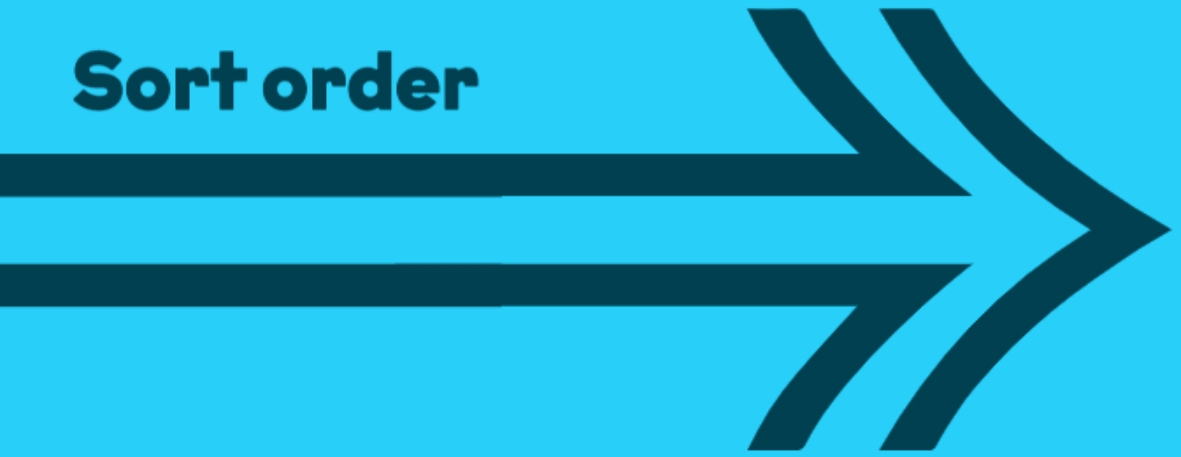
ORDER BY clause

ORDER BY CLAUSE

SELECT <<field(s)>>
FROM <<table(s)>>
WHERE <<condition(s)>>
GROUP BY *<<field(s)>>*
HAVING *<<condition(s)>>*
ORDER BY *<<field(s)>>*

POWERING POTENTIAL

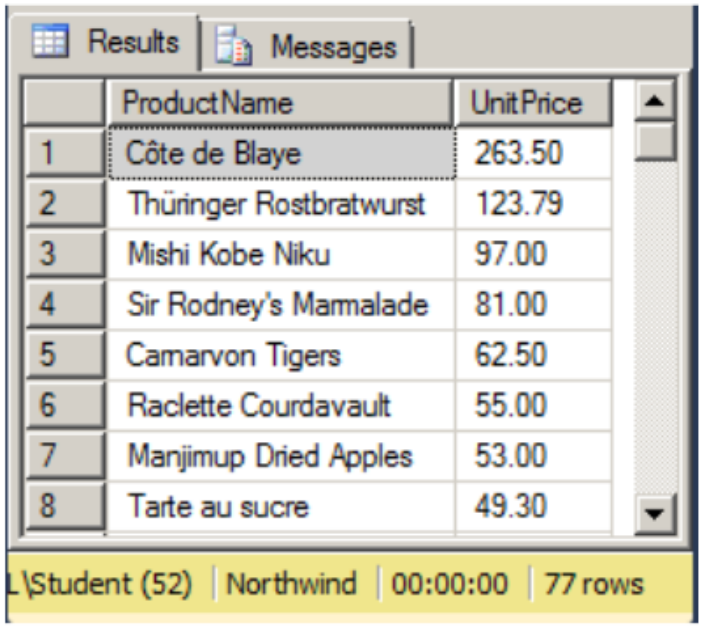
Sort order



Sort order

- **ASC - ascending**
  - Default
- **DESC - descending**

```
SELECT
  ProductName,
  UnitPrice
FROM
  dbo.Products
ORDER BY
  UnitPrice DESC
```



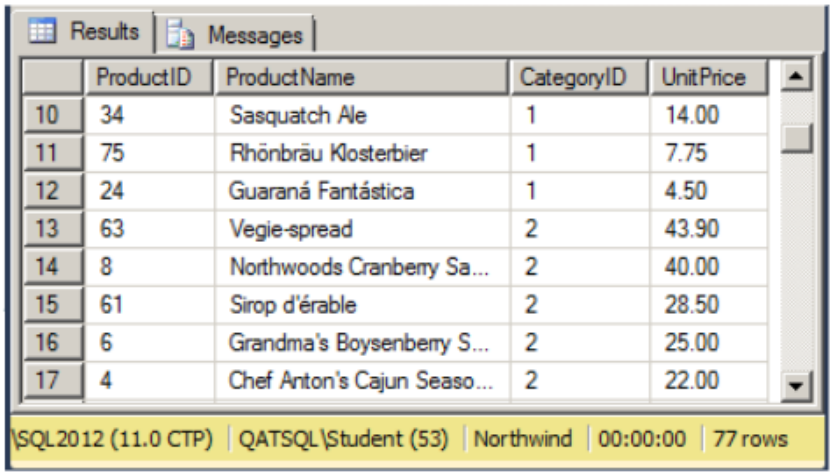
POWERING POTENTIAL

Sort on multiple columns

Sort on multiple columns

- Up to 16 columns in ORDER BY clause
- Sort columns do not have to be included in select list

```
SELECT
  ProductID, ProductName,
  CategoryID, UnitPrice
FROM
  dbo.Products
ORDER BY
  CategoryID,
  UnitPrice DESC
```


Results | Messages

	ProductID	ProductName	CategoryID	UnitPrice
10	34	Sasquatch Ale	1	14.00
11	75	Rhönbräu Klosterbier	1	7.75
12	24	Guaraná Fantástica	1	4.50
13	63	Vegie-spread	2	43.90
14	8	Northwoods Cranberry Sa...	2	40.00
15	61	Sirop d'érable	2	28.50
16	6	Grandma's Boysenberry S...	2	25.00
17	4	Chef Anton's Cajun Seaso...	2	22.00

\SQL2012 (11.0 CTP) | QATSQL\Student (53) | Northwind | 00:00:00 | 77 rows

POWERING POTENTIAL

Sort on expressions

Sort on expressions

ORDER BY

- <<expression>>
- *<<expression-alias>>*
- <<expression-ordinal>>

```
SELECT
  ProductName, UnitsInStock,
  UnitsOnOrder,
  UnitsInStock + UnitsOnOrder
    AS FutureStock
FROM
  dbo.Products
ORDER BY
  FutureStock DESC
```

Results | Messages

	ProductName	UnitsInStock	UnitsOnOrder	FutureStock
1	Rhönbräu Klosterbier	125	0	125
2	Boston Crab Meat	123	0	123
3	Grandma's Boysenberry Spread	120	0	120
4	Pâté chinois	115	0	115
5	Sirop d'érable	113	0	113
6	Geitost	112	0	112
7	Inlagd Sill	112	0	112
8	Sasquatch Ale	111	0	111

QATSQL\SQL2012 (11.0 CTP) | QATSQL\Student (52) | Northwind | 00:00:00 | 77 rows

POWERING POTENTIAL

Eliminate duplicates - DISTINCT

Eliminate duplicates – SELECT DISTINCT

```
SELECT
  City, Country
FROM
  dbo.Customers
```

Results | Messages

	City	Country
1	Berlin	Germany
2	México D.F.	Mexico
3	México D.F.	Mexico
4	London	UK
5	Luleå	Sweden
6	Mannheim	Germany
7	Strasbourg	France
8	Madrid	Spain

4) | Northwind | 00:00:00 | 91 rows

```
SELECT DISTINCT
  City, Country
FROM
  dbo.Customers
```

Results | Messages

	City	Country
1	Aachen	Germany
2	Albuquerque	USA
3	Anchorage	USA
4	Århus	Denmark
5	Barcelona	Spain
6	Barquisimeto	Venezuela
7	Bergamo	Italy
8	Berlin	Germany

(54) | Northwind | 00:00:00 | 69 rows

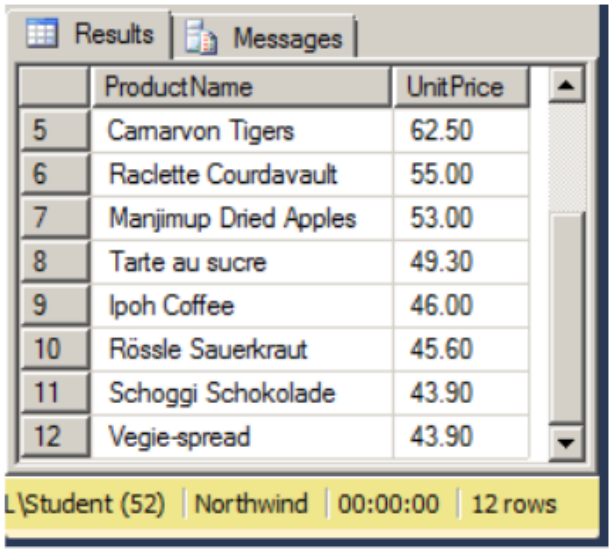
POWERING POTENTIAL

Limit results - TOP

Limit results – SELECT TOP

- SELECT TOP *n*
- SELECT TOP (*n* PERCENT)
- SELECT TOP ... WITH TIES

```
SELECT TOP 11 WITH TIES
  ProductName,
  UnitPrice
FROM
  dbo.Products
ORDER BY
  UnitPrice DESC
```



Hands-on lab

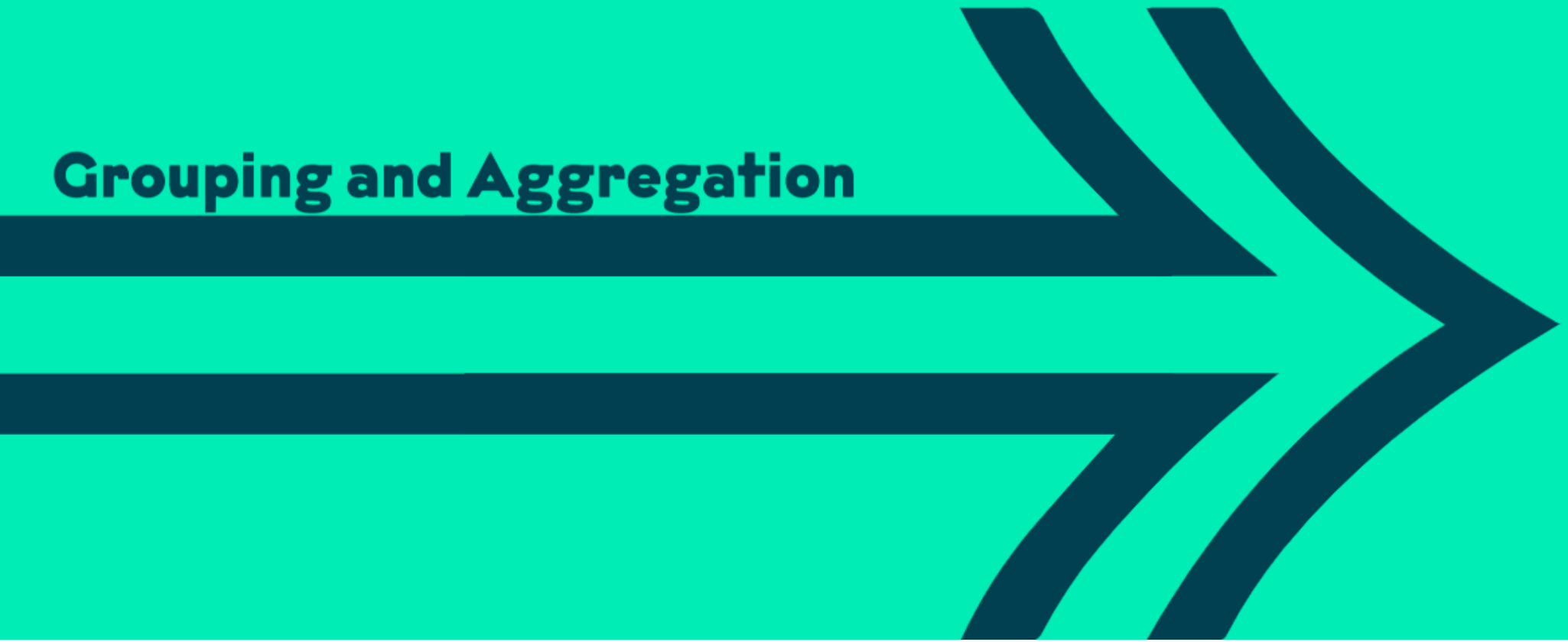
HANDS-ON LAB

- Basic sorting
- Sorting on expressions
- Unique rows
- Limiting results

- ORDER BY clause
- Sorting on columns
- Sort orders
- Selecting unique rows
- Limiting results

Review

Grouping and Aggregation



Overview

- Aggregate functions
- GROUP BY clause
- HAVING clause
- Hands-on lab

OBJECTIVES

At the end of this module you will be able to:

- write a query that calculates single aggregates
- write a query that calculates aggregates for grouped information
- write a query that aggregates, groups and filters on calculated values

POWERING POTENTIAL

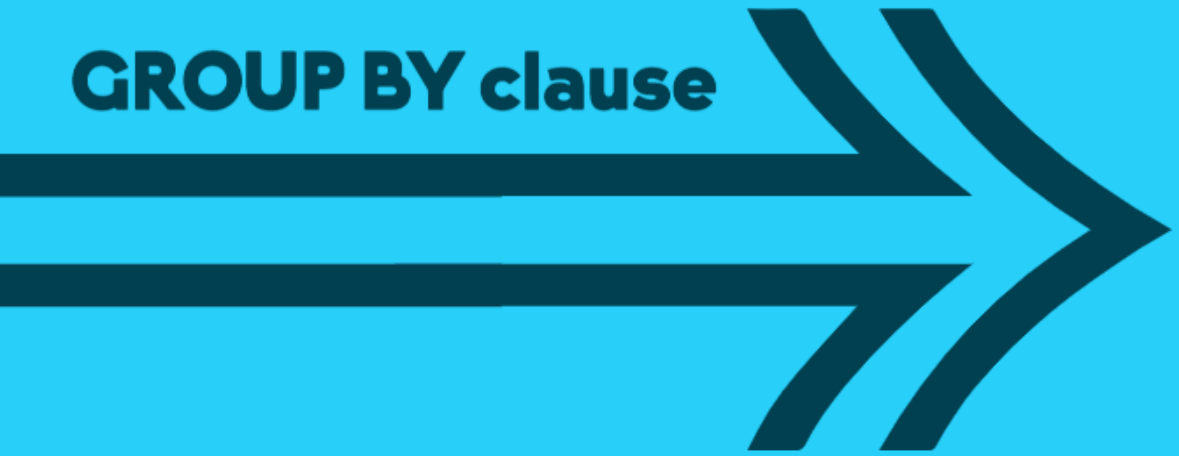
Aggregate functions

AGGREGATE FUNCTIONS

- **COUNT(*)**
- **COUNT(***<<expression>>***)**
- **SUM(***<<expression>>***)**
- **AVG(***<<expression>>***)**
- **MIN(***<<expression>>***) / MAX(***<<expression>>***)**
- **Statistical aggregates**
  - STDEV / STDEVP / VAR / VARP

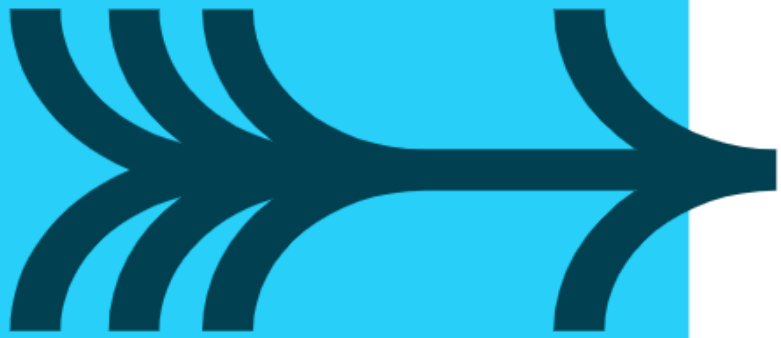
POWERING POTENTIAL

GROUP BY clause



GROUP BY CLAUSE

- SELECT <<field(s)>>
- FROM <<table(s)>>
- WHERE <<condition(s)>>
- **GROUP BY** *<<field(s)>>*
- HAVING *<<condition(s)>>*
- ORDER BY <<field(s)>>



Grouping aggregates

```
SELECT
  CategoryID,
  AVG(UnitPrice)
FROM
  dbo.Products
GROUP BY
  CategoryID
```

Results | Messages

	CategoryID	(No column name)
1	1	37.9791
2	2	23.0625
3	3	25.16
4	4	28.73
5	5	20.25
6	6	54.0066
7	7	32.37
8	8	20.6825

ent (55) | Northwind | 00:00:00 | 8 rows

```
SELECT
  CategoryID,
  AVG(UnitPrice)
FROM
  dbo.Products
GROUP BY
  CategoryID
ORDER BY 2 DESC
```

Results | Messages

	CategoryID	(No column name)
1	6	54.0066
2	1	37.9791
3	7	32.37
4	4	28.73
5	3	25.16
6	2	23.0625
7	8	20.6825
8	5	20.25

dent (55) | Northwind | 00:00:00 | 8 rows

POWERING POTENTIAL

HAVING clause

HAVING CLAUSE

- SELECT <<field(s)>>
- FROM <<table(s)>>
- WHERE *<<row-level filter(s)>>*
- **GROUP BY** *<<field(s)>>*
- **HAVING** *<<group-level filters(s)>>*
- ORDER BY <<field(s)>>

Filtering grouped aggregates

```
SELECT
  CategoryID,
  AVG(UnitPrice)
FROM
  dbo.Products
GROUP BY
  CategoryID
HAVING
  AVG(UnitPrice) > 30
```

Results | Messages

	CategoryID	CategoryAvg
1	1	37.9791
2	6	54.0066
3	7	32.37

(52) | Northwind | 00:00:00 | 3 rows

```
SELECT
  CategoryID,
  AVG(UnitPrice)
FROM
  dbo.Products
GROUP BY
  CategoryID
HAVING
  CategoryID > 5
```

Results | Messages

	CategoryID	CategoryAvg
1	6	54.0066
2	7	32.37
3	8	20.6825

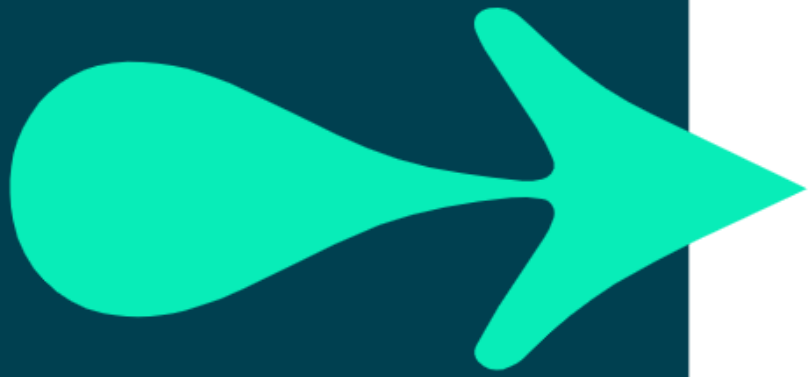
nt (52) | Northwind | 00:00:00 | 3 rows

POWERING POTENTIAL

Hands-on lab

HANDS-ON LAB

- Single aggregates
- GROUP BY
- HAVING



- Aggregate functions
- GROUP BY
- HAVING

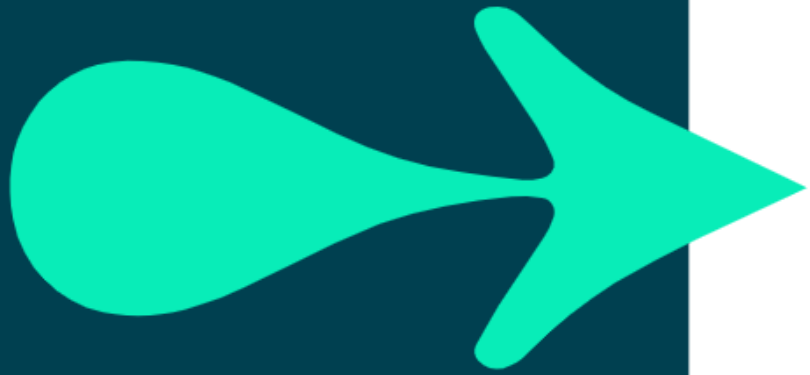
Review

Using Multiple Tables

Overview

- JOINs
- Join types
- UNION, EXCEPT and INTERSECT
- Hands-on lab

OBJECTIVES



At the end of this module you will be able to:

- write a query that joins two tables together
- write a query that joins multiple tables
- write a query that uses an OUTER JOIN
- write a query that uses a UNION operator to join tables

POWERING POTENTIAL

JOINS

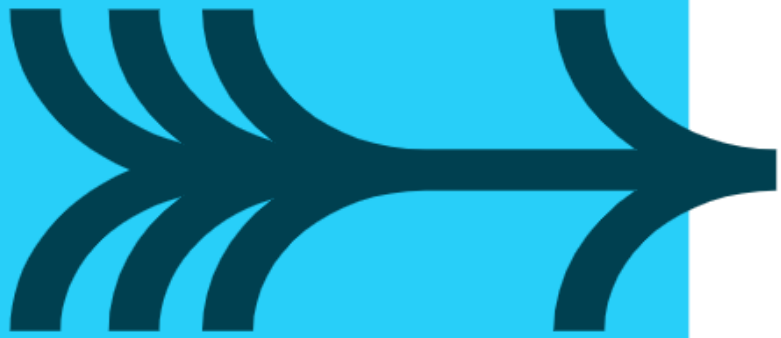
JOINS

SELECT <<field(s)>>
FROM <<table1>>
type-of **JOIN** *<<table2>>* **ON** *<<predicate>>*
WHERE <<condition(s)>>
GROUP BY <<field(s)>>
HAVING <<condition(s)>>
ORDER BY <<field(s)>>

POWERING POTENTIAL

Join Types

JOIN TYPES



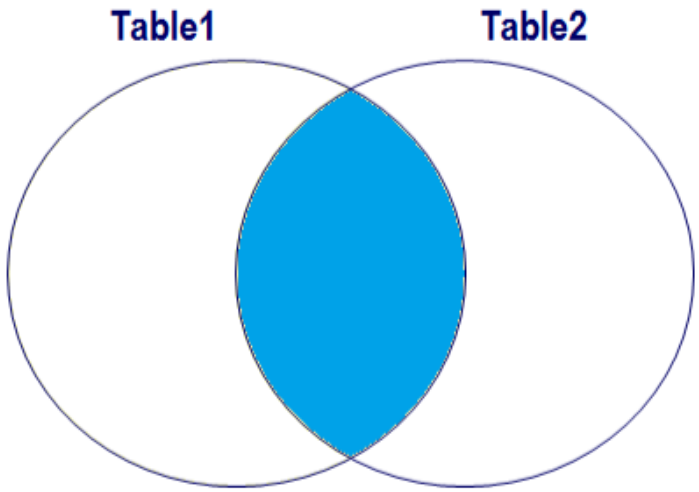
JOINs

- Inner
- Outer (right, left and full)

Inner join

- Returns columns from both tables where the ON clause is true

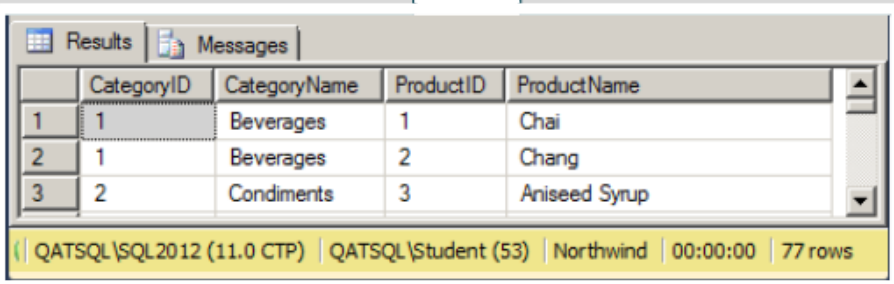
```
SELECT *
        FROM Table1 INNER JOIN Table2
                ON Table1.Col1 = Table2.Col2
```



Inner join

```
SELECT
  Products.CategoryID,
  CategoryName,
  ProductID,
  ProductName
FROM
  dbo.Products
INNER JOIN
  dbo.Categories
ON
  Products.CategoryID =
  Categories.CategoryID
```

```
SELECT
  p.CategoryID,
  CategoryName,
  ProductID,
  ProductName
FROM
  dbo.Products AS p
JOIN
  dbo.Categories AS c
ON
  p.CategoryID =
  c.CategoryID
```

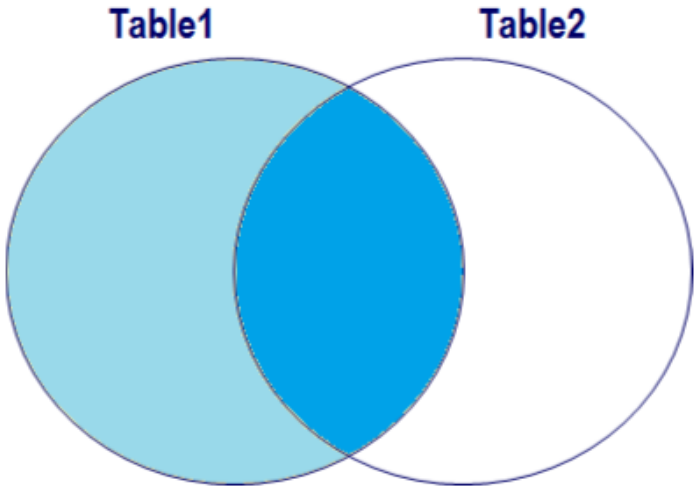

Results | Messages

	CategoryID	CategoryName	ProductID	ProductName
1	1	Beverages	1	Chai
2	1	Beverages	2	Chang
3	2	Condiments	3	Aniseed Syrup

QATSQL\SQL2012 (11.0 CTP) | QATSQL\Student (53) | Northwind | 00:00:00 | 77 rows

Left (outer) Join

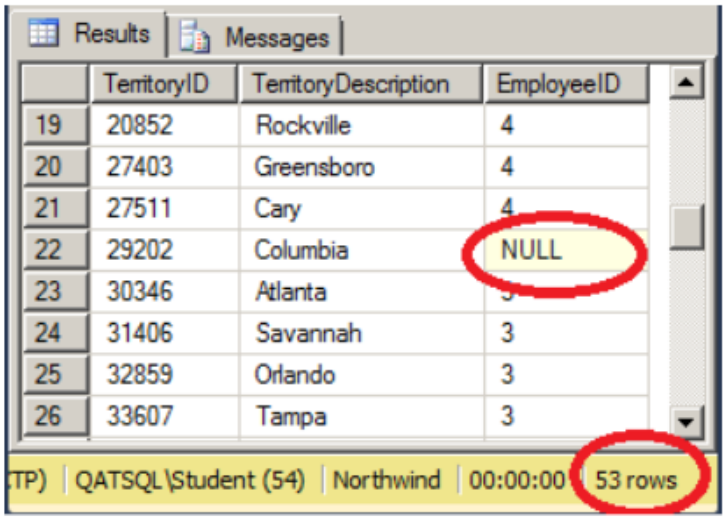
- Returns columns from both tables where the ON clause is true and columns from Table1 in all cases




```
SELECT *
        FROM Table1 LEFT JOIN Table2
                ON Table1.Col1 = Table2.Col2
```

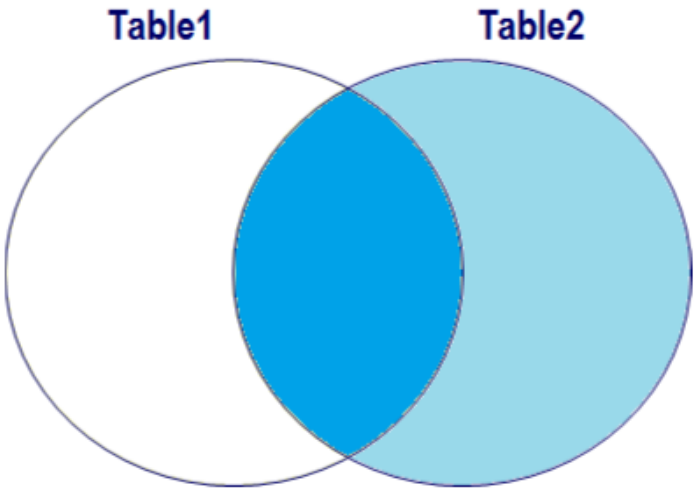
Left join

```
SELECT
  t.TerritoryID,
  t.TerritoryDescription,
  et.EmployeeID
FROM
  dbo.Territories AS t
LEFT JOIN
  dbo.EmployeeTerritories AS et
ON
  t.TerritoryID =
    et.TerritoryID
```



Right Join

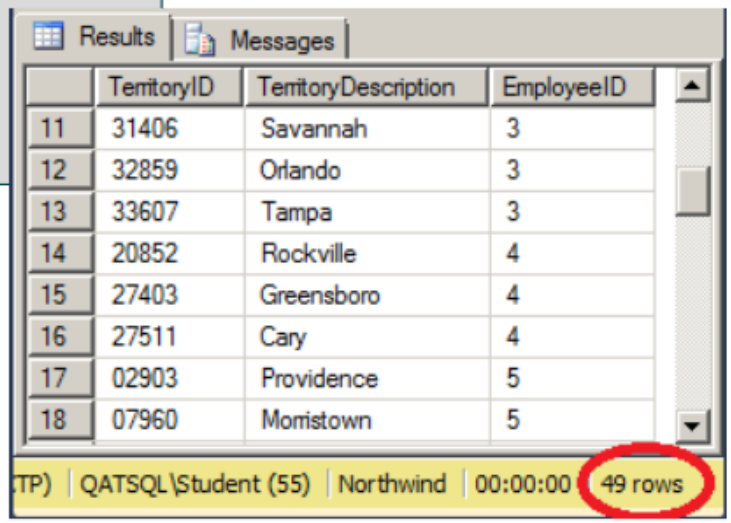
- Returns columns from both tables where the ON clause is true and columns from Table2 in all cases




```
SELECT *
        FROM Table1 RIGHT JOIN Table2
                ON Table1.Col1 = Table2.Col2
```

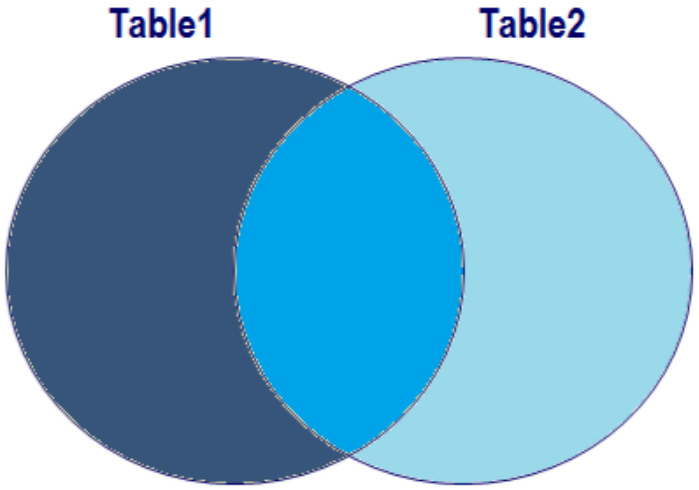
Right join

```
SELECT
  t.TerritoryID,
  t.TerritoryDescription,
  et.EmployeeID
FROM
  dbo.Territories AS t
RIGHT JOIN
  dbo.EmployeeTerritories AS et
ON
  t.TerritoryID =
    et.TerritoryID
```



Full Join

- Returns columns from both tables where the ON clause is true and disjointed rows where matches are not found




```
SELECT *
        FROM Table1 FULL JOIN Table2
                ON Table1.Col1 = Table2.Col2
```

Joining more than two tables

```
SELECT
  t.TerritoryID, t.TerritoryDescription,
  et.EmployeeID, e.FirstName, e.LastName
FROM dbo.Territories AS t
LEFT JOIN dbo.EmployeeTerritories AS et
ON t.TerritoryID = et.TerritoryID
LEFT JOIN dbo.Employees AS e
ON e.EmployeeID = et.EmployeeID
```

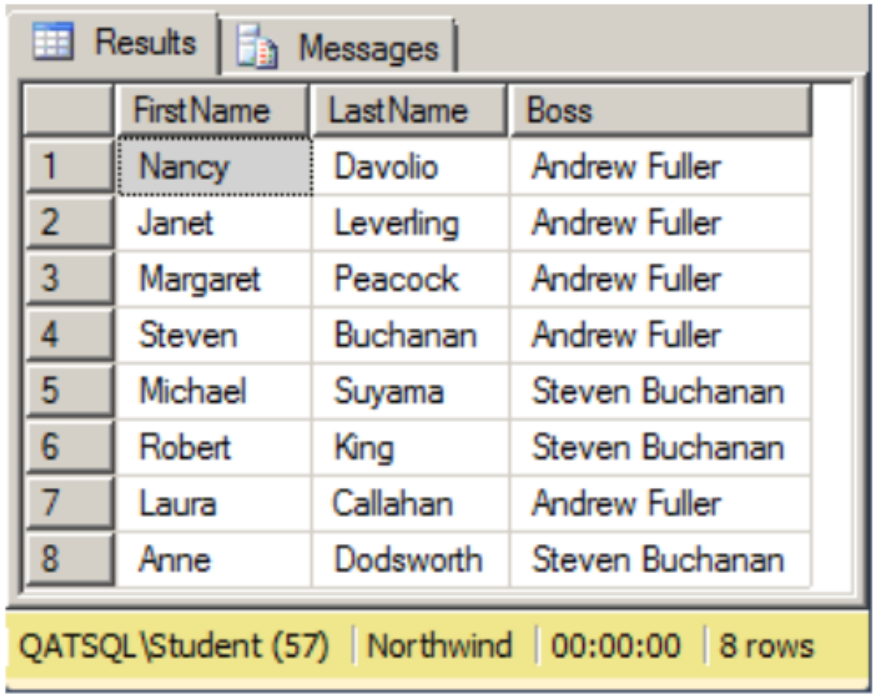
Results | Messages

	TerritoryID	TerritoryDescrip...	EmployeeID	FirstName	LastName
20	27403	Greensboro	4	Margaret	Peacock
21	27511	Cary	4	Margaret	Peacock
22	29202	Columbia	NULL	NULL	NULL
23	30346	Atlanta	3	Janet	Leverling
24	31406	Savannah	3	Janet	Leverling
25	32859	Orlando	3	Janet	Leverling
26	33607	Tampa	3	Janet	Leverling

ATSQL\SQL2012 (11.0 CTP) | QATSQL\Student (56) | Northwind | 00:00:00 | 53 rows

Joining a table to itself

```
SELECT
  emp.FirstName, emp.LastName,
  mgr.FirstName + ' ' + mgr.LastName
    AS Boss
FROM
  dbo.Employees AS emp
JOIN
  dbo.Employees AS mgr
ON
  emp.ReportsTo =
    mgr.EmployeeID
```


Results | Messages

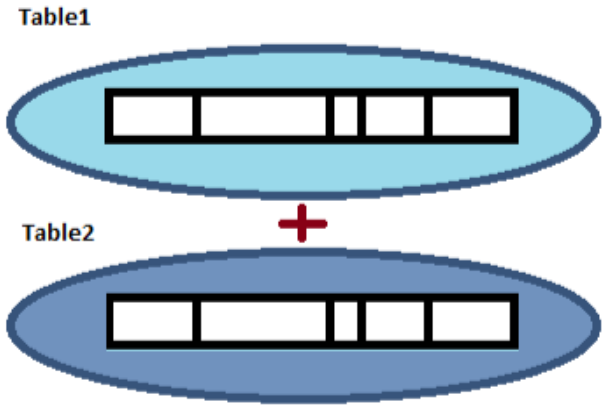
	FirstName	LastName	Boss
1	Nancy	Davolio	Andrew Fuller
2	Janet	Leverling	Andrew Fuller
3	Margaret	Peacock	Andrew Fuller
4	Steven	Buchanan	Andrew Fuller
5	Michael	Suyama	Steven Buchanan
6	Robert	King	Steven Buchanan
7	Laura	Callahan	Andrew Fuller
8	Anne	Dodsworth	Steven Buchanan

QATSQL\Student (57) | Northwind | 00:00:00 | 8 rows

UNION, EXCEPT and INTERSECT

Union [All]

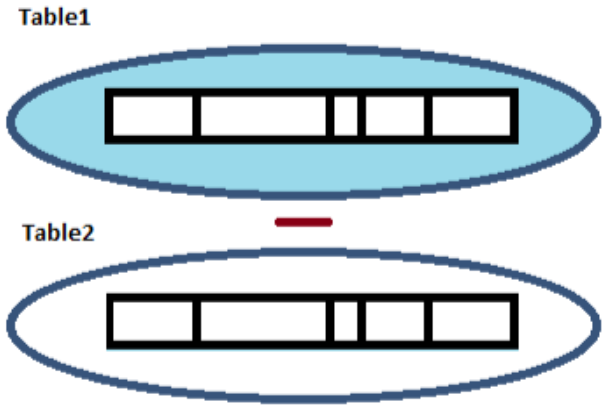
- Accumulation of both sets
- Duplicates may occur with ALL




```
SELECT * FROM Table1
UNION
SELECT * FROM Table2
```

Except

- Rows from Table1 where they do not exist in Table2



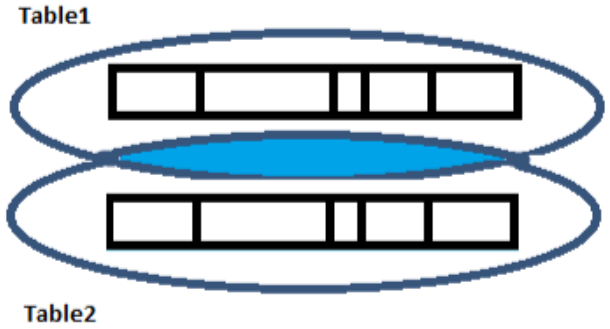

```
SELECT * FROM Table1

EXCEPT

SELECT * FROM Table2
```

Intersect

- Rows that exist in both sets will be returned




```
SELECT * FROM Table1

INTERSECT

SELECT * FROM Table2
```

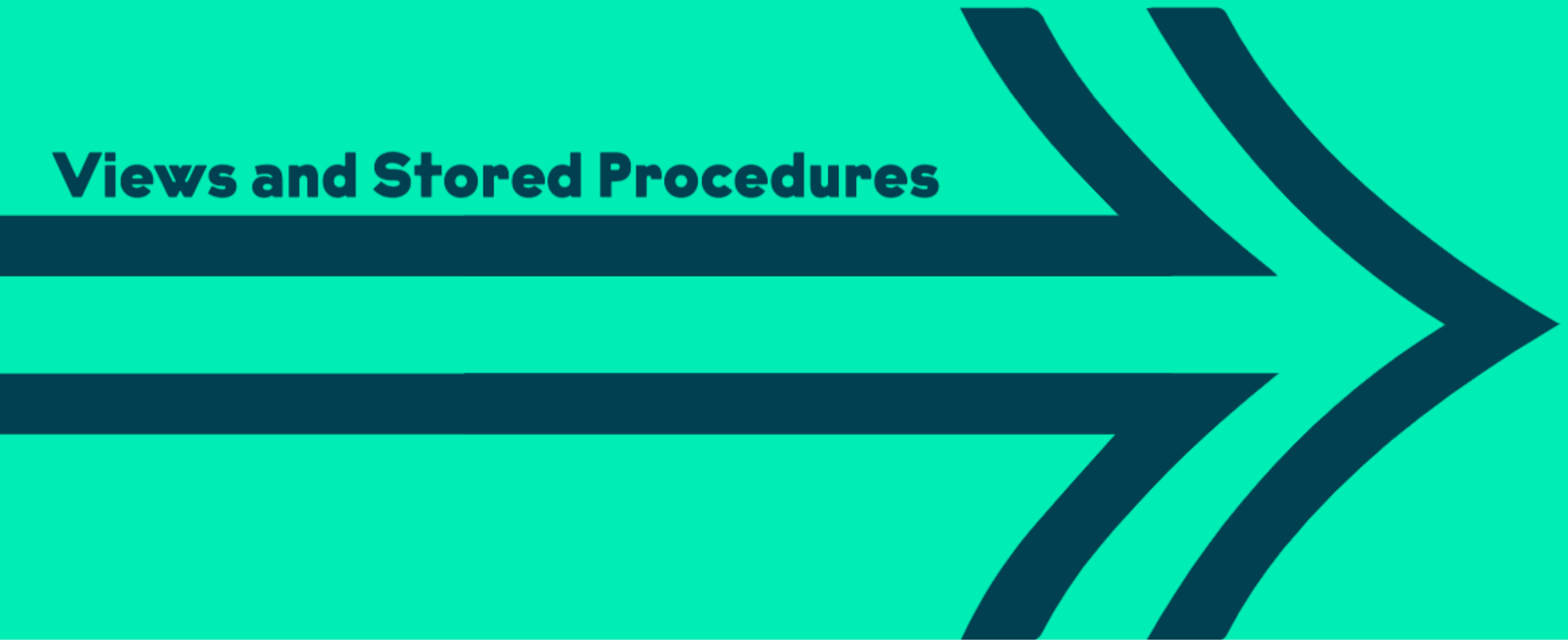
POWERING POTENTIAL

Hands-on lab

HANDS-ON LAB

- Inner joins
- Outer joins
- Unions

Views and Stored Procedures



Overview

- Views
- Using Views
- Stored Procedures
- Using Stored Procedures

OBJECTIVES

At the end of this module you will be able to:

- use a predefined view
- create a simple view
- use a predefined stored procedure

POWERING POTENTIAL

Views

VIEWS

- **Stored select statements**
- **Act just like tables**
- **Benefits:**
  - Reusable
  - Simplify complex queries
  - Limit access to sensitive data

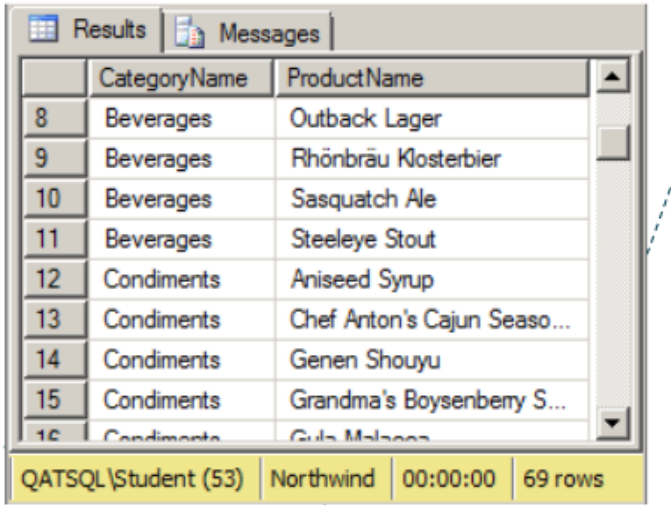
POWERING POTENTIAL

Using Views

Using views

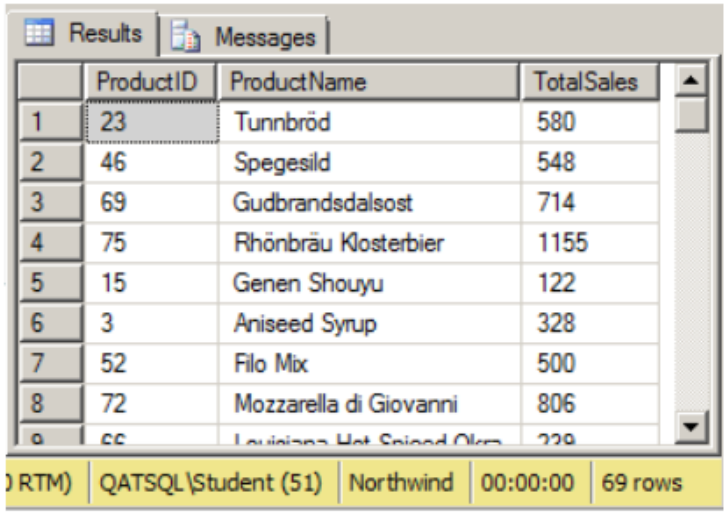
```
SELECT
  CategoryName,
  ProductName
FROM
  dbo.Products AS p
JOIN
  dbo.Categories AS c
ON
  p.CategoryID =
  c.CategoryID
WHERE
  p.Discontinued = 0
ORDER BY
  CategoryName,
  ProductName
```

```
SELECT
  CategoryName,
  ProductName
FROM
  dbo.[Products by Category]
ORDER BY
  CategoryName,
  ProductName
```



Using views (2)

```
SELECT
  p.ProductID,
  p.ProductName,
  SUM(Quantity) AS TotalSales
FROM
  dbo.[Current Product List] AS p
JOIN
  dbo.[Order Details] AS od
ON
  p.ProductID = od.ProductID
GROUP BY
  p.ProductID, p.ProductName
```



Results | Messages

	ProductID	ProductName	TotalSales
1	23	Tunnbröd	580
2	46	Spegesild	548
3	69	Gudbrandsdalsost	714
4	75	Rhönbräu Klosterbier	1155
5	15	Genen Shouyu	122
6	3	Aniseed Syrup	328
7	52	Filo Mix	500
8	72	Mozzarella di Giovanni	806
9	66	Louisiana Hot Spiced Okra	229

) RTM) | QATSQL\Student (51) | Northwind | 00:00:00 | 69 rows

Stored Procedures

STORED PROCEDURES

- Set of SQL statements stored in the database
- Can accept parameters
- Can return records
- Can modify the database
- Can perform complex logic

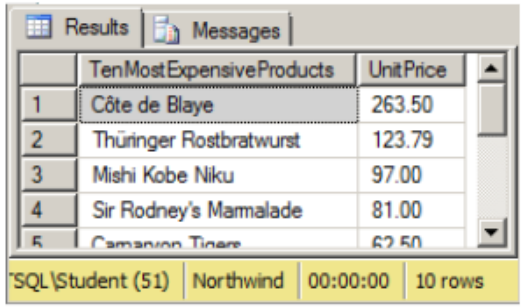
POWERING POTENTIAL

Using Stored Procedures

Using stored procedures

- **EXECUTE** *<<procedure-name>>*
- **EXEC** *<<procedure-name>> param1, ..., paramN*

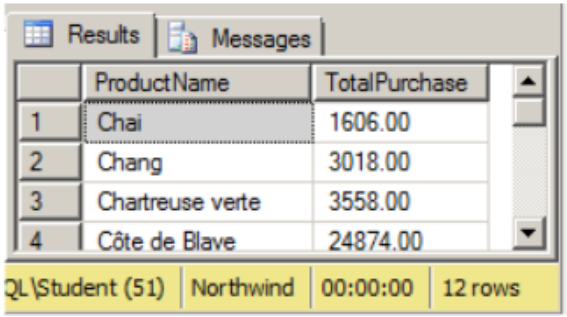
```
EXEC dbo.[Ten Most Expensive Products]
```


Results | Messages

	TenMostExpensiveProducts	UnitPrice
1	Côte de Blaye	263.50
2	Thüringer Rostbratwurst	123.79
3	Mishi Kobe Niku	97.00
4	Sir Rodney's Marmalade	81.00
5	Carnarvon Tigers	62.50

SQL\Student (51) | Northwind | 00:00:00 | 10 rows

```
EXEC dbo.SalesByCategory
  'Beverages', 1996
```

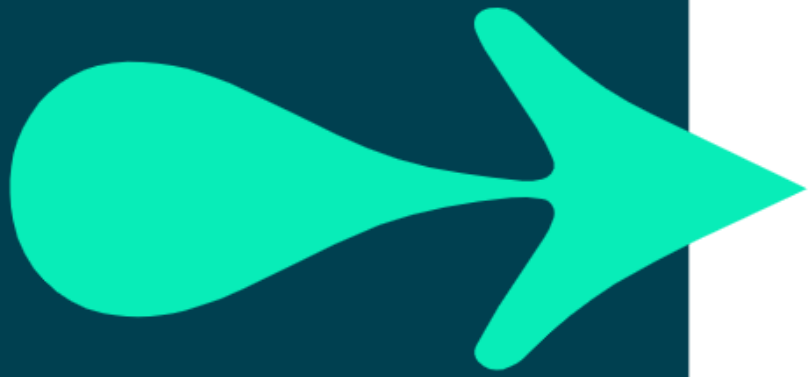

Results | Messages

	ProductName	TotalPurchase
1	Chai	1606.00
2	Chang	3018.00
3	Chartreuse verte	3558.00
4	Côte de Blaye	24874.00

QL\Student (51) | Northwind | 00:00:00 | 12 rows

HANDS-ON LAB

- Using views
- Creating views
- Using stored procedures



Common Functions

Overview

- Functions
- Text functions
- Date functions
- Working with nulls
- Data conversion functions
- Hands-on lab

OBJECTIVES

At the end of this module you will be able to:

- use text manipulation functions
- use date manipulation functions
- use the ISNULL function
- format dates

Functions

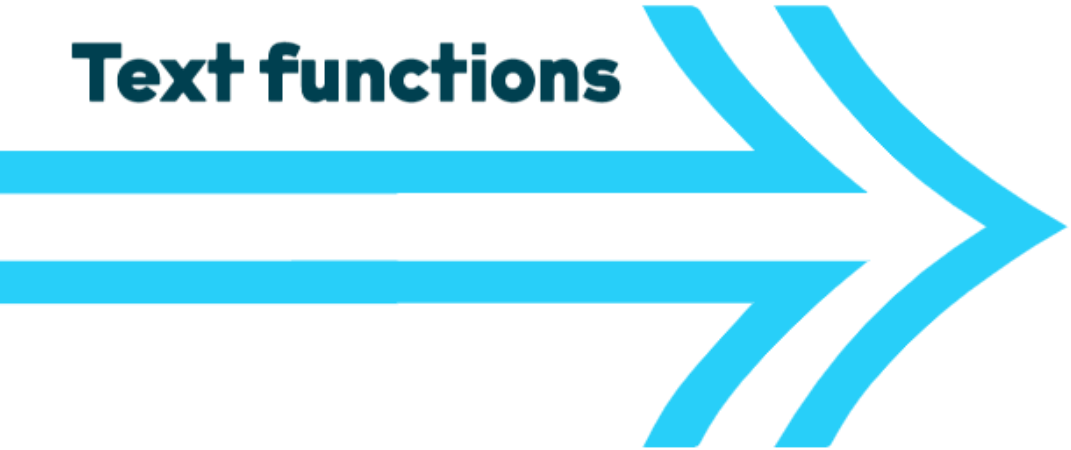
FUNCTIONS

- **Perform a calculation**
- **Can be passed parameters**
- **Return a value**
- **Can be used wherever an expression is expected**
  - Select lists
  - WHERE clauses
  - ORDER BY clauses

POWERING POTENTIAL

Text functions

Text functions



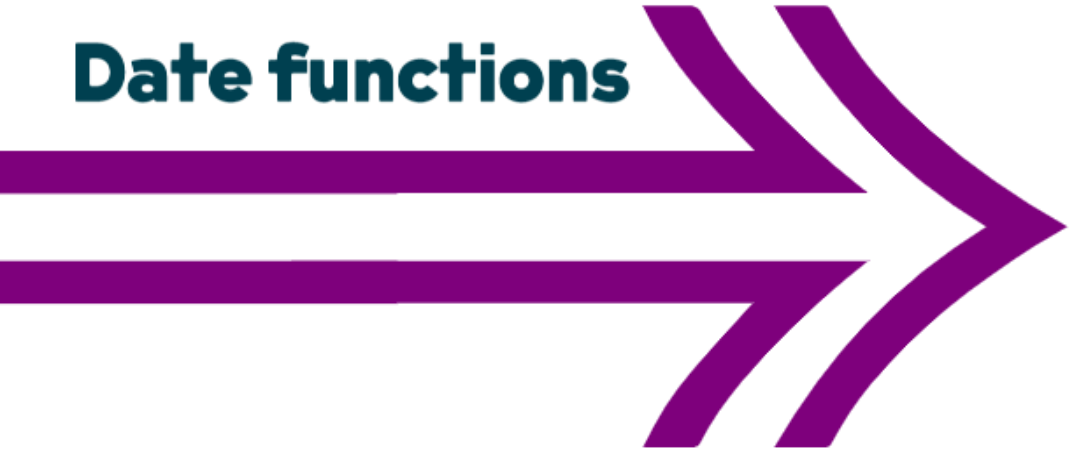
- **Left / Right *(expr, n)***
  - return the left / right -most *n* characters of *expr*
- **Upper / Lower *(expr)***
  - Convert *expr* to all UPPERCASE or all lowercase letters
- **SubString *(expr, start, length)***
  - Take *length* letters from *expr*, starting from *start*
- **CharIndex / PatIndex *(look-for, expr, start)***
  - CharIndex looks for an exact match on *look-for* in *expr*, optionally starting at *start*
  - PatIndex performs pattern matching similar to LIKE

Date functions

WORKING WITH DATES

- **DateTime**
  - Date and time, minimum date 01/01/1753, accurate to 0.00333s
- **SmallDateTime**
  - As datetime but less accurate (no milliseconds)
- **DateTime2** *since SQL Server 2008*
  - date and time, min. 01/01/0001, accurate to 100 nanoseconds
- **DateTimeOffset** *since 2008*
  - As DateTime2 also includes a time zone stamp (+/- hh:mm)
- **Date / Time** *since 2008*
  - Date or Time portion of DateTime2 only

Date functions

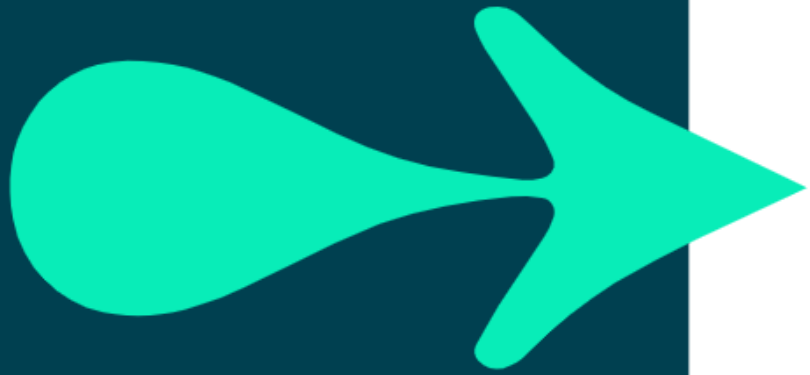


- **GetDate() / GetUTCDate()**
- Retrieve the current system data and time
- **DateAdd*(part, number, expr)***
- Add number of parts (months / days / etc) to expr
- **DateDiff*(part, startDate, endDate)***
- Return the number of parts difference between start and end
- **DatePart*(part, expr)***
- Return the part value of expr
- **Year / Month / Day *(expr)***

POWERING POTENTIAL

Working with nulls

DEALING WITH NULLS



- **ISNULL(expr, null-value)**
  - If expr IS NULL, return null-value
- **NULLIF(expr1, expr2)**
  - If expr1 equals expr2, return NULL
- **COALESCE(expr1, expr2, ..., exprN)**
  - Return the first non-null value in the list of expressions:
    - IF expr1 IS NULL, return expr2 unless that is also null in which case return exprN, otherwise return expr1.

POWERING POTENTIAL

Data conversion functions

CONVERSION FUNCTIONS

- CAST(*expr* AS *type*)
- CONVERT(*type*, *expr*)
- CONVERT(*type*, *expr*, *format*)

New in SQL Server 2012:

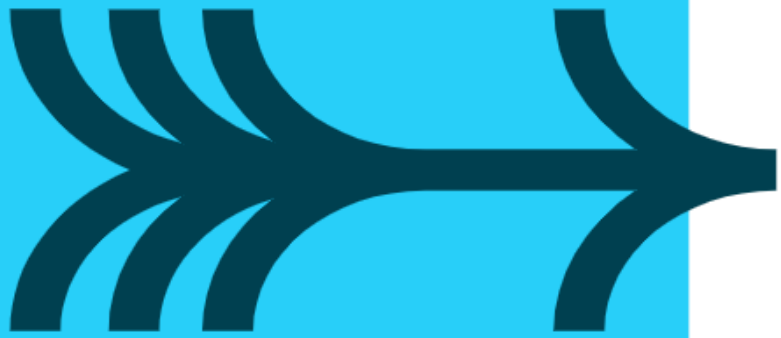
- PARSE
- TRY_PARSE
- TRY_CONVERT

POWERING
POTENTIAL

Hands-on lab

HANDS-ON LAB

- Text functions
- Date functions
- NULL functions
- Formatting dates



- Functions
- Text functions
- Date functions
- Working with nulls
- Data conversion functions

Review

Modifying Data

Overview

- Adding new rows
  - INSERT
- Modifying existing rows
  - UPDATE
- Removing rows
  - DELETE

POWERING
POTENTIAL

INSERT

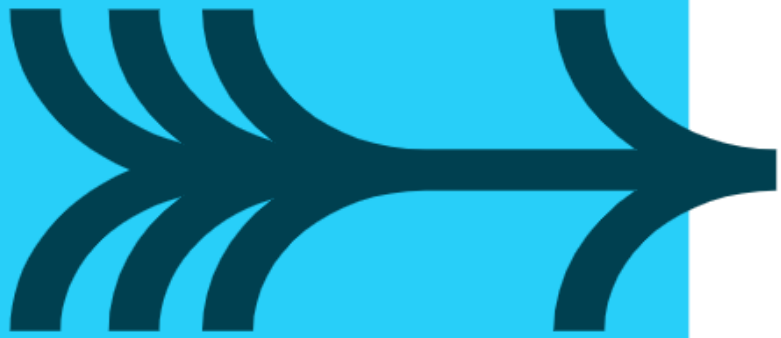
INSERT

- **INSERT [INTO]** *table* **(***col1*, ... *colN***) VALUES (***val1*, ..., *valN***)**
- **INSERT [INTO]** *table* **SELECT ...**

POWERING POTENTIAL

UPDATE

UPDATE



- **UPDATE**
  table
  SET
  col1 **=** *val1*,
  ...,
  colN **=** *valN*
  WHERE
  colX **=** *valX*

POWERING POTENTIAL

DELETE

DELETE

- **DELETE [FROM]** *table*
  WHERE ...

- INSERT
- UPDATE
- DELETE

Review

Working with Tables

Overview

- Data Definition Language (DDL)
  - CREATE
  - ALTER
  - DROP
- Data Manipulation Language (DML)
  - INSERT
  - UPDATE
  - DELETE

POWERING POTENTIAL

CREATE

CREATE

- **Use CREATE to add a table to the database with the columns defined**

- **CREATE TABLE table**
  (
  ColumnName datatype <options>
  ...
)

- **Options:**
  - NULL / NOT NULL
  - DEFAULT
  - CHECK
  - PRIMARY KEY

Data Types

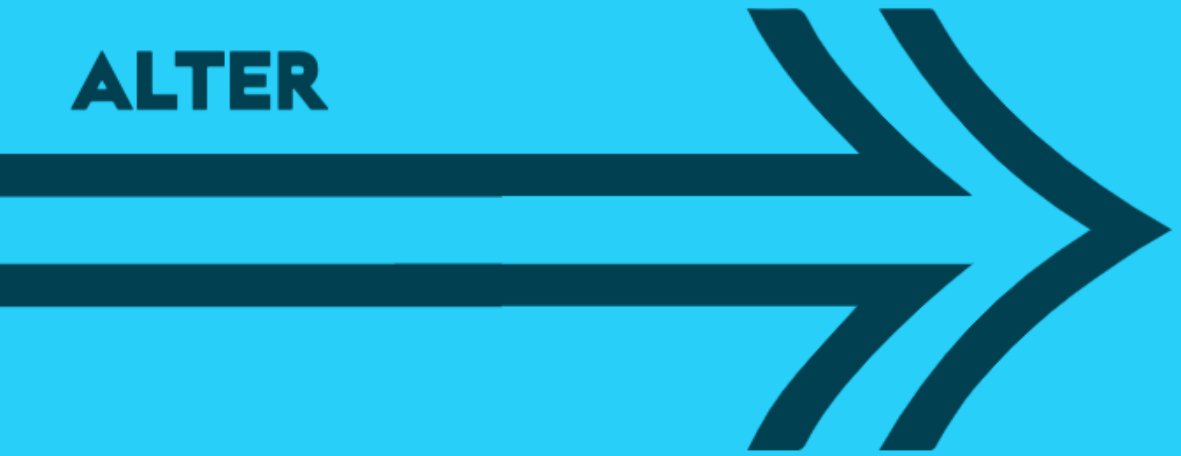
- **Numerics**
  - BigInt, Int, SmallInt, TinyInt, Bit
  - Decimal, Numeric
  - Money, SmallMoney
  - Float, Real
- **Date / Time**
  - DateTime, DateTime2, Date, Time, SmallDateTime, DateTimeOffset
- **Character strings**
  - VarChar, Char, NVarChar, NChar
- **Binary data**
  - Binary, Image, VarBinary
- **Other**
  - XML, Geometry, Geography, HierarchyID, Cursor

Options

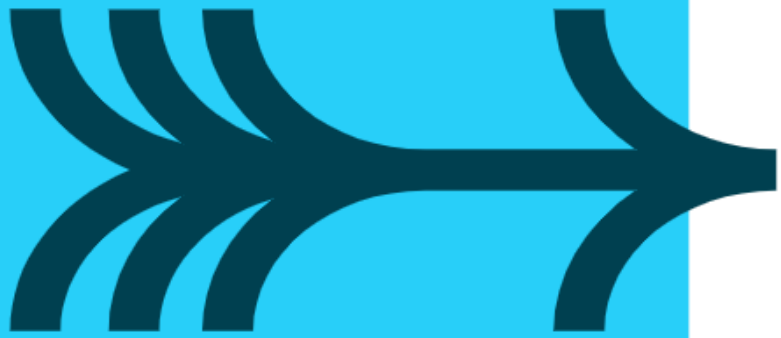
- **NULL / NOT NULL**
  - NULL allows blank / empty values in the column.
  - NOT NULL does not allow blank / empty values.
- **DEFAULT**
  - Allows for a default to be entered if no value is given, such as No in an OrderedFilled column, as the field will be set to Yes only when the products are sent.
- **CHECK**
  - Allows for a rule to be placed on the field such as Age < 130 AND Age>= 0 on a column holding human ages.
- **PRIMARY KEY**
  - This designates the column (or columns combined) as unique within the table, so no duplicates can be held and no nulls are allowed.
  - One primary key is allowed per table.

POWERING POTENTIAL

ALTER



ALTER



- **Use ALTER to change the structure of a table:**
  - Add columns
  - Remove columns

- **ALTER tableName**
  ADD columnName datatype <options>

- **ALTER tableName**
  DROP COLUMN columnName

POWERING
POTENTIAL

DROP

DROP

- Use DROP to remove the table from the database
- DROP tableName

POWERING POTENTIAL

INSERT

INSERT

- **INSERT [INTO]** *table* **(***col1*, ... *colN***) VALUES (***val1*, ..., *valN***)**
- **INSERT [INTO]** *table* **SELECT ...**

POWERING POTENTIAL

UPDATE

UPDATE

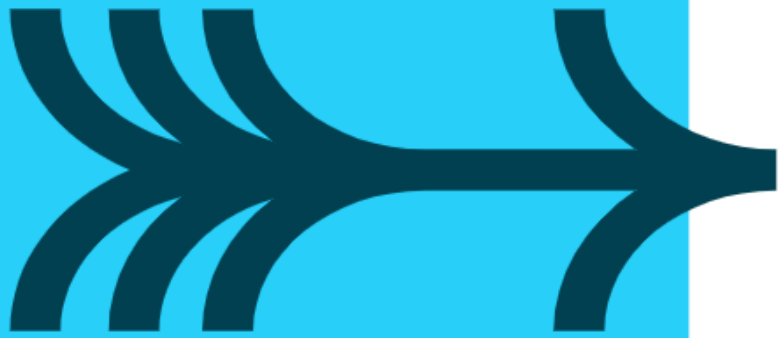
- **UPDATE**
  table
  SET
  col1 **=** *val1*,
  ...,
  colN **=** *valN*
  WHERE
  colX **=** *valX*

POWERING POTENTIAL

DELETE

DELETE

- **DELETE [FROM]** *table*
  WHERE ...



- CREATE
- ALTER
- DROP
- INSERT
- UPDATE
- DELETE

Review

Want to find out more?

QA.COM

Fancy a chat?

0345 757 3888

QALtd QA-Ltd QALtd QALimited

V1.2