Aggregate Queries

SUM, AVG, MIN, MAX …

Example database – Employees & Departments

Employees

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	75	46	Toulouse

Departments

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Jose

Example database – Employees & Departments

```
CREATE TABLE Departments(
 DeptName varchar(50)  PRIMARY KEY ,
 Address varchar(50) NULL,
 City varchar(50) NULL
);
```

```
INSERT INTO Departments VALUES
('Administration', 'Bond Street', 'London'),
('Distribution', 'Pond Road', 'Brighton'),
('Planning','Bond Street', 'London'),
('Production', 'Rue Victor Hugo', 'Toulouse'),
('Research', 'Sunset Street', 'San Jose');
```



```
CREATE TABLE dbo.Employees(
FirstName varchar(50) NOT NULL,
Surname varchar(50) NOT NULL,
Dept varchar(50) REFERENCES  Departments(DeptName),
Office varchar(50) NULL,
Salary int NULL,
City varchar(50) NULL,
PRIMARY KEY (FirstName, Surname)
);
```



```
INSERT INTO Employees VALUES
('Mary', 'Brown', 'Administration', '10', 45, 'London'),
('Charles', 'White', 'Production', '20', 36, 'Toulouse'),
('Gus', 'Green', 'Administration', '20', 40, 'Oxford'),
('Jackson', 'Neri', 'Distribution', '16', 45, 'Dover'),
('Charles', 'Brown', 'Planning', '14', 80,          'London'),
('Laurence', 'Chen', 'Planning', '7', 73, 'Worthing'),
('Pauline', 'Bradshaw', 'Administration', '75', 40, 'Brighton'),
('Alice', 'Jackson', 'Production', '75', 46, 'Toulouse');
```

Departments(DeptName, Address, City)

Employee(FirstName, Surname, Dept, Office, Salary, City)

FK(Dept) -> Departments(DeptName)

SQL – Aggregate Queries

- Aggregate queries utilise SQL functions to count, sum and calculate the maximum, minimum and average values over numerical fields in a Database
- Example questions these functions answer:
  - COUNT: How to count the number of employees, the number of distinct salary values?
  - SUM: What is the salary expenditure of each department?
  - MIN: What is minimum salary among all employees?
  - MAX: Who receives the highest salary in the department?
  - AVG: What is the average salary of all employees of a department?

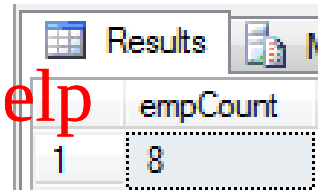
  - With the exception of COUNT, all aggregate operators apply to a <u>single attribute</u>

SQL – Aggregate Queries – COUNT

- Standard syntax of the count function:
  - count(<* |[distinct|all]AttributeList>)

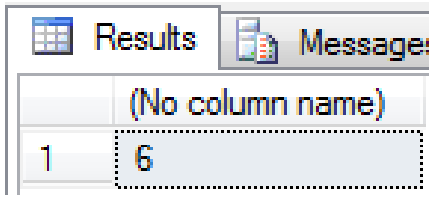
- Find the number of employees

```
SELECT COUNT(*) AS empCount FROM Employees;
```



- Find the number of different values of Salary

```
SELECT COUNT (DISTINCT Salary) FROM Employees
```



Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

SQL – Aggregate Queries – COUNT

- Returns the number of rows or DISTINCT values:
  - **COUNT**(<*|[DISTINCT|ALL]AttributeList>)
  - **COUNT** applies to duplicates, unless otherwise stated

- Find the number of employees
  - SELECT COUNT(*) FROM Employees

- Find the number of different values of Salary
  - SELECT COUNT(DISTINCT Salary) FROM Employees
    - Note: DISTINCT will return only the 1st of duplicate values
      - If two people earn 42k, it will only be counted once not twice

- Find the number of rows having a not null value for Salary
  - SELECT COUNT(Salary) FROM Employees; or alternatively:
  - SELECT COUNT(ALL Salary) FROM Employees

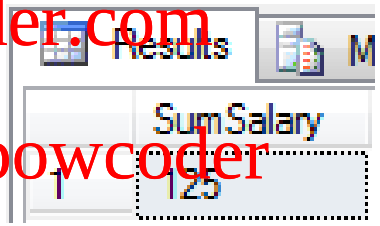
Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries – SUM, AVG, Max and MIN

- Standard syntax of the sum function:
  - <sum|max|min|avg>(<[distinct|all]AttributeList>)

- Find the sum of the salaries of the Administration department:


```
SELECT SUM(Salary) as SumSalary
FROM Employees
WHERE Dept = 'Administration'
```

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries – SUM, AVG, Max and MIN

```
SELECT  Sum(salary) AS TotalSalaries
FROM    Employees
```

Results | Messages

	TotalSalaries
1	405

What is the difference in their meaning?


```
SELECT  Sum(salary) AS TotalSalaries
FROM    Employees
WHERE   Dept = 'Administration'
```

Results | Messages

	TotalSalaries
1	125

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries – SUM, AVG, Max and MIN

- Incorrect query:

```
SELECT FirstName, Surname, MAX(Salary) AS TopSalary
FROM Employees
```

FirstName	Surname	Max(Sal)
Mary	Brown	80
Charles	White	?
Gus	Green	?
…	…	?

- Might be more than one employee - whose name?

```
Msg 8120, Level 16, State 1, Line 1
Column 'Employees.FirstName' is invalid in the select list because
it is not contained in either an aggregate function or the GROUP BY clause.
```


  - Remember, SELECT returns ROWS (tuples) of data whereas an aggregate like MAX or AVG aggregates all those rows and returns a single value
    - They are not compatible expressions in the same query

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries – SUM, AVG, Max and MIN

- Multiple aggregates can be combined into a single query
  - This is acceptable since they all return a single value

```
SELECT MAX(Salary) AS MaxSal, MIN(Salary) AS MinSal
FROM Employees
```


Results | Messages

	MaxSal	MinSal	AvgSal
1	80	36	50

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

SQL – Aggregate Queries and GROUP BY

- Multiple aggregates can be combined into a single query
  - This is acceptable since they all return a single value
  - The GROUP BY clause aggregates operators into subsets of rows

- Find the highest salary paid in each department:


```
SELECT Dept, MAX(Salary) AS HighestSalary
FROM Employees GROUP BY Dept;
```


Results | Messages

	Dept	HighestSalary
1	Administration	45
2	Distribution	45
3	Planning	80
4	Production	46

When one or more columns in the SQL SELECT statement is not encapsulated in the Aggregate function, the column(s) **must be in a SQL GROUP BY clause**.

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries with GROUP BY

- GROUP BY collects data across multiple records and then groups the results by one or more columns
  - Find the sum of salaries for every department:

```
SELECT Dept, Salary FROM Employees
ORDER BY Dept ASC
```


	Dept	Salary
1	Administration	40
2	Administration	45
3	Administration	40
4	Distribution	45
5	Planning	73
6	Planning	80
7	Production	36
8	Production	46

Un-grouped query

Results | Messages

	Dept	TotalSalaries
1	Administration	125
2	Distribution	45
3	Planning	153
4	Production	82

```
SELECT Dept, SUM(Salary) AS TotalSalaries
FROM Employees GROUP BY Dept;
```

Grouped query

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries with GROUP BY

- GROUP BY collects data across multiple records and then groups the results by one or more columns
  - Find the MAX salary by first name:

```
SELECT FirstName, Salary
FROM Employees
```

Un-grouped query

```
SELECT FirstName, MAX(Salary)
FROM Employees
GROUP BY FirstName
```

Grouped query

Results | Messages

	FirstName	Salary
1	Alice	46
2	Charles	80
3	Charles	36
4	Gus	40
5	Jackson	45
6	Laurence	73
7	Mary	45
8	Pauline	40

Results | Messages

	FirstName	(No column name)
1	Alice	46
2	Charles	80
3	Gus	40
4	Jackson	45
5	Laurence	73
6	Mary	45
7	Pauline	40

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries GROUP BY vs Nested Query

- GROUP BY queries can also be represented using Nested Queries
  - The nested query helps visualise how the group by statement works
  - For each row returned, the sub-query is run and returns a single value:


```
SELECT Dept, MAX(Salary) AS HighestSalary
FROM Employees GROUP BY Dept;
```



```
SELECT DISTINCT E1.Dept,
        (
        SELECT MAX(Salary) FROM Employees AS E2
        WHERE  E1.Dept = E2.Dept
        ) AS HighestSalary
FROM Employees E1;
```

Results | Messages

	Dept	HighestSalary
1	Administration	45
2	Distribution	45
3	Planning	80
4	Production	46

Returns a single value for each row of outer query

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries – How GROUP BY works

1. First, the query is executed without group by clause and aggregate operators
2. Then the query result is divided into subsets with the same values for the attributes in the GROUP BY clause
3. Finally, the aggregate operators are applied


```
SELECT Dept, SUM(Salary) AS TotalSal
FROM Employees
GROUP BY Dept
```


1

Dept	Salary
Administration	45
Production	36
Administration	40
Distribution	45
Planning	80
Planning	73
Administration	40
Production	46

2

Dept	Salary
Administration	45
Administration	40
Administration	40
Distribution	45
Planning	80
Planning	73
Production	36
Production	46

3

Dept	TotalSal
Administration	125
Distribution	45
Planning	153
Production	82

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

SQL – Aggregate Queries and GROUP BY

- Multiple aggregates can be combined into a single query
  - This is acceptable since they all return a single value
  - The GROUP BY clause aggregates operators into subsets of rows
  - **Invalid** Query:

```
SELECT DeptName, COUNT(*), D.City
FROM Employees E JOIN Departments D
ON E.Dept = D.Deptname
GROUP BY D.DeptName
```


Messages

```
Msg 8120, Level 16, State 1, Line 1
Column 'Departments.City' is invalid in
the select list because it is not contained
in either an aggregate function or the
GROUP BY clause.
```

  - **Correct** Query:

```
SELECT DeptName, COUNT(*), D.City
FROM Employees E JOIN Departments D
ON E.Dept = D.Deptname
GROUP BY D.DeptName, D.City
```

Results | Messages

	DeptName	(No column name)	City
1	Administration	3	London
2	Distribution	1	Brighton
3	Planning	2	London
4	Production	2	Toulouse

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries With JOINs

- Find the maximum salary among the employees who work in a department based in London:
  - 1. Join Employees to Departments to take into account what city they work in (not live in!) – filter on London
    - This returns many tuples
  - 2. Find the max salary from the filtered result – MAX(Salary)
    - This returns the single value


```
SELECT MAX(e.Salary) AS MaxLondonSal
FROM Employees e INNER JOIN Departments d
ON e.Dept = d.DeptName
WHERE d.City = 'London'
```

Results | Messages

	MaxLondonSal
1	80

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Aggregate Queries With GROUP BY and JOINs

- Incorrect query:

```
SELECT Office FROM Employees
GROUP BY Dept
```

- Incorrect query:

```
SELECT DeptName, COUNT(*), D.City
FROM Employees E INNER JOIN Departments D
ON E.Dept = D.Deptname
GROUP BY D.DeptName
```

1. All non-aggregate attributes in the SELECT clause must appear in the GROUP BY clause
2. The GROUP BY clause may have more attributes than those non-aggregate attributes in the SELECT clause

- Correct query:

```
SELECT DeptName, COUNT(*), D.City
FROM Employees E INNER JOIN Departments D
ON E.Dept = D.Deptname
GROUP BY D.DeptName, D.City
```

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

GROUP BY Predicates (conditions) - HAVING

- The **HAVING** clause is used to place conditions on the result of an aggregate operator
  - Find which departments spend more than 100 on salaries

```
SELECT Dept, SUM(Salary) AS theSalary
FROM Employees
GROUP BY Dept
HAVING SUM(Salary) > 100
```

```
SELECT Dept, SUM(Salary) AS theSalary
FROM Employees
GROUP BY Dept
HAVING theSalary > 100
```


Results | Messages

	Dept	theSalary
1	Administration	125
2	Planning	153

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

GROUP BY Predicates (conditions) - HAVING

- When conditions are based on the result of an aggregate operator use the HAVING clause
  - Find which departments spend more than 100 on salaries

```
SELECT Dept
FROM Employees
GROUP BY Dept
HAVING SUM (Salary) > 100
```


Results | Mess

	Dept
1	Administration
2	Planning

  - If you want to see the sum of the salaries:

```
SELECT Dept, SUM(Salary)
FROM Employees
GROUP BY Dept
HAVING SUM (Salary) > 100
```

Results | Messages

	Dept	(No column name)
1	Administration	125
2	Planning	153

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

GROUP BY Predicates (conditions) - HAVING

- WHERE vs HAVING?
  - Conditions involving **aggregate** operators must appear in a **HAVING** clause
  - Conditions involving **non-aggregate** operators must appear in the **WHERE** clause

  - EXAMPLE: Find the departments in which the average salary of employees working in office 20 is higher than 25


```
SELECT Dept, AVG(Salary)
FROM Employees
WHERE office = '20'
GROUP BY Dept
HAVING AVG(Salary) > 25
```

Normal condition

Aggregate Condition

Results | Messages

	Dept	(No column name)
1	Administration	40
2	Production	36

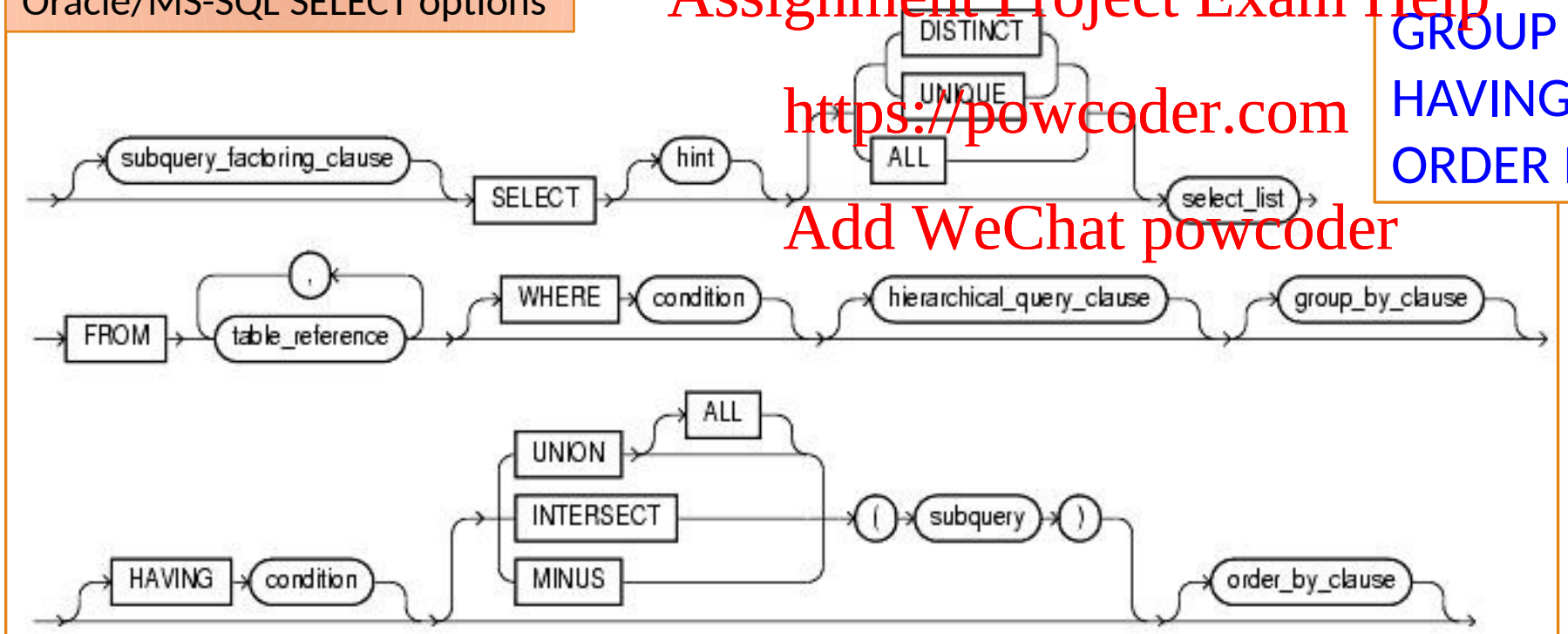
Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

Syntax of an SQL query

- Example complete query with all available options:

```
SELECT TargetList
FROM TableList
WHERE Condition
GROUP BY GroupingAttributeList
HAVING AggregateCondition
ORDER BY OrderingAttributeList
```

Oracle/MS-SQL SELECT options



SET Operators

SQL Queries - SET Operators

- Set operations are used to combine query results
- Standard SQL syntax:
  - SelectSQL {< UNION|INTERSECT|EXCEPT>[ALL] } SelectSQL
  - The standard uses **EXCEPT** instead of **minus** for difference

  - A set is a result table (eg, results returned by a SELECT query).
  - Both sets must have the same number of attributes (columns)

  - Both sets must have compatible attributes (domains/data types)

SET 1 Columns	SET 2 Columns
Varchar	Varchar
Int	Int
Bit	Bit
Varchar	Int
Varchar	NULL | 'TBA'

Result
Varchar
Int
Bit
???
Varchar

NULL and 'TBA' are both examples of varchar

SET Queries – UNION

- Set operations are used to combine query results
  - Duplicates are removed (unless the all option is used)
  - List the first names and surnames of employees
  - This will remove any duplicates


```
SELECT FirstName AS Name
FROM Employees
UNION
SELECT Surname
FROM Employees
```


Results

	Name
1	Alice
2	Bradshaw
3	Brown
4	Charles
5	Chen
6	Green
7	Gus
8	Jackson
9	Laurence
10	Mary
11	Neri
12	Pauline
13	White

No Duplicates

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

SET Queries – UNION ALL

- Example: Find the first names and surnames of the employees

```
SELECT FirstName AS Name
FROM Employees
UNION ALL
SELECT Surname
FROM Employees
```


Results

	Name
1	Alice
2	Charles
3	Charles
4	Gus
5	Jackson
6	Laurence
7	Mary
8	Pauline
9	Jackson
10	Brown
11	White

Duplicates retained by ALL

- Use **ALL** if you want to keep any duplicates
  - By default, duplicates are removed

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

SET Queries – INTERSECT

- Find the surnames of employees that are also first names

```
SELECT FirstName AS Name FROM Employees
INTERSECT
SELECT Surname AS Name FROM Employees
```


  - This is equivalent to


```
FROM Employees E1 INNER JOIN Employees E2
ON E1.FirstName = E2.Surname
```



```
SELECT DISTINCT E1.FirstName AS Name
FROM Employees E1 WHERE EXISTS (
SELECT * FROM Employees E2
WHERE E1.FirstName = E2.Surname
)
```

Can also use IN|ANY

Results

	Name
1	Jackson

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

SET Queries – EXCEPT

- Finds non-matching values between the two SQL data sets:
  - Find the surnames of employees that are not also first names

```
SELECT FirstName AS Name FROM Employees
EXCEPT
SELECT Surname AS Name FROM Employees
```


  - This is equivalent to:

```
SELECT DISTINCT FirstName AS Name FROM Employees
        WHERE FirstName NOT IN (
        SELECT Surname AS Name FROM Employees
)
```


Nested Query

Results

	Name
1	Alice
2	Charles
3	Gus
4	Laurence
5	Mary
6	Pauline

```
SELECT DISTINCT E1.FirstName AS Name
FROM Employees E1 WHERE NOT EXISTS (
SELECT * FROM Employees E2
WHERE E1.FirstName = E2.Surname
)
```

Can also use <> ANY

SQL Functions

DATE, STRING. . .

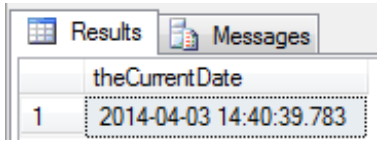
SQL – Functions – STRING

- Functions are calculations performed by the DBMS
- Common functions include:

Function	Example	Output
UPPER(col)	UPPER(Name)	Sam → SAM
LOWER(col)	LOWER(Name)	Sam → sam
RTRIM(col)	RTRIM(Name)	[Sam] → [Sam]
LTRIM(col)	LTRIM(Name)	[Sam] → [Sam]
LEN(col)	LEN(Name)	Returns int length of string


```
SELECT GETDATE() AS theCurrentDate

SELECT SUBSTRING(FirstName, 1, 1) AS Initial
```



https://msdn.microsoft.com/en-us/library/ms181984.aspx

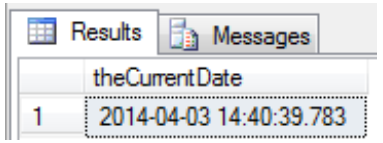
SQL – Functions – STRING

- Functions are calculations performed by the DBMS
- Common functions include:

Function	Example	Output
LEFT(string, length)	UPPER(Name)	Sam → SAM
RIGHT(string, length)	LOWER(Name)	Sam → sam
CHARINDEX(string1, string2)	CHARINDEX('Fun', 'DBFundamentals')	3
SUBSTRING(col, start, length)	SUBSTRING(Name, 1, 1)	Returns char(s) at start position


```
SELECT GETDATE() AS theCurrentDate

SELECT SUBSTRING(FirstName, 1, 1) AS Initial
```



https://msdn.microsoft.com/en-us/library/ms181984.aspx

SQL – Functions – STRING

- Working with Strings in SQL is not easy:

```
DECLARE @address varchar(100) = '13 Wayville road, Woodville, SA 5000'

SELECT
  LEFT(@address, CHARINDEX(',', @address) - 1) AS streetAddress,
  LEFT(secondPart, LEN(secondPart) - CHARINDEX(' ', REVERSE(secondPart)) -1) AS suburb,
  RIGHT(secondPart, CHARINDEX(' ', REVERSE(secondPart))) AS state,
  REVERSE(SUBSTRING(REVERSE(@address), 1, 4)) AS postcode
  FROM (
  SELECT
    RTRIM(
          REVERSE(
                  SUBSTRING(
                          REVERSE(@address), 6, LEN(@address) - CHARINDEX(',', @address) – 5
                              )
                  )
          ) AS secondPart
) AS t1
```

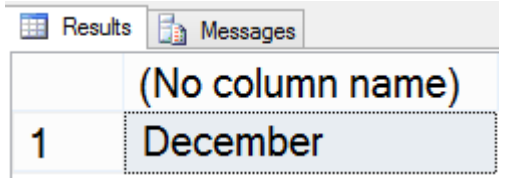

streetAddress	suburb	state	postcode
13 Wayville road	Woodville	SA	5000

SQL – Functions – DATE

- Functions are calculations performed by the DBMS
- Common functions include:

Function	Example	Output
GETDATE()		01/09/2015
DATEPART(datepart, date)	DATEPART(d, GetDate())	3
DATENAME(datepart, date)	DATENAME(dw, GetDate())	Wednesday
	DATENAME(m, GetDate())	September
DATEADD(datepart, number, date)	DATEADD(d, 7, GetDate())	Date 7 days from today!

- Dateparts:
  - d | m | y => day | month | year number of the calander date
  - dw returns the day of the week number except when used with DATENAME where it returns the name of the day!
  - m will return the name of the month when used with DATENAME
  - Test it out for yourself by placing SELECT before the function
  - SELECT DATENAME(m, '31/Dec/2015')



https://msdn.microsoft.com/en-us/library/ms186724.aspx

SQL CONTROL – IF and ELSE

- Control statements allow actions to happen depending on a condition
  - The action may involve setting a value
  - The action may involve running a different query

- Basic Syntax


```
IF (<someTrue|FalseCondition)
-- Do Something

ELSE
-- Do Something Else
```



```
IF (1 = 0)
SELECT 'True' AS Result
ELSE
SELECT 'False' AS Result
```

Results | Messages

	Result
1	False

SQL CONTROL – IF and ELSE

- Control statements allow actions to happen depending on a condition
  - The action may involve setting a value
  - The action may involve running a different query

- Basic Syntax


```
IF (<someTrue|FalseCondition)
-- Do Something

ELSE
-- Do Something Else
```



```
IF (1 = 0) -- (false)
SELECT 'Unlikely' AS Result

ELSE IF (1 = 2) -- (false)
SELECT 'Just as Unlikely' AS Result

ELSE
SELECT 'False' AS Result
```

Results | Messages

	Result
1	False

SQL CONTROL – CASE WHEN

- Control statements allow actions to happen depending on a condition
  - The CASE WHEN statement is similar to if-else but can be used within a query to change a particular value

- Basic Syntax


```
CASE
  WHEN (<someTrue|FalseCondition)
  THEN -- Do Something

  WHEN (<someTrue|FalseCondition2)
  THEN -- Do Something a bit different

ELSE
-- Do Something different again
END
```



```
SELECT (
  CASE WHEN (1 = 0) -- (false)
  THEN  'Unlikely'

  WHEN (1 = 2) -- (false)
  THEN  'Just as Unlikely'

ELSE
   'False'
END
)
AS Result
```

Results | Messages

	Result
1	False

SQL CONTROL – CASE WHEN

- Control statements allow actions to happen depending on a condition
  - The CASE WHEN statement is similar to if-else but can be used within a query to change a particular value
  - Example


```sql
SELECT FirstName, Surname, Salary,
(
CASE WHEN (E.Salary > 75)
THEN  'Over Paid'

WHEN (E.Salary <= 40)
THEN  'Under Paid'

ELSE
 'Adequately Paid'
END
) AS PayConclusion
FROM Employees AS E
```


Single column result called "PayConclusion" with value that depends on the employee salary

Results | Messages

	FirstName	Surname	Salary	PayConclusion
1	Alice	Jackson	46	Adequately Paid
2	Charles	Brown	80	Over Paid
3	Charles	White	36	Under Paid
4	Gus	Green	40	Under Paid
5	Jackson	Neri	45	Adequately Paid
6	Laurence	Chen	73	Adequately Paid
7	Mary	Brown	45	Adequately Paid
8	Pauline	Bradshaw	40	Under Paid

Departments(DeptName, Address, City)
Employee(FirstName, Surname, Dept, Office, Salary, City)
FK(Dept) -> Departments(DeptName)

SQL – NULL replacement Value

- ISNULL is a function that can be used to provide a value when an unknown or NULL value is returned
  - ISNULL(expressionORattribute, replacementValue)

  - Find ALL Simpsons characters and where available their first aired episode else show 'TBA'


```
SELECT CharacterName,
ISNULL(EpisodeName, 'TBA') AS FirstEpisode
FROM Characters C LEFT OUTER JOIN Episodes E
ON C.EpisodeID = E.EpisodeID
```


Results | Messages

	CharacterName	FirstEpisode
1	"Mayor "Diamond Joe" Quimby	TBA
2	Blinky	TBA
3	Homer Simpson	TBA
4	Mr. Teeny	TBA
5	Old Jewish Man	TBA
6	Rabbi Hyman Krustofski	TBA
7	Sanjay Nahasapeemapetilon	TBA
8	Veterinarian	TBA
9	Waylon Smithers	TBA
10	Wendell Borton	TBA
11	Troy McClure	TBA
12	The Rich Texan	"$pringfield (Or, How I Learned
13	Ernst and Gunter	"$pringfield (Or, How I Learned
14	Gloria	"A Hunka Hunka Burns in Love
15	Chase	"A Milhouse Divided"
16	Rachel Jordan	"Alone Again, Natura-Diddily"