

```
>>> A crash course in SQL
>>> Statistical Society of Australia
```

Daniel Fryer [†]

Nov, 2021

[†]daniel@vfryer.com

```
>>> Daily schedule
```

Timetable				
9:00am	-	10:30am	lecture 1	(1.5 hr)
10:30am	-	11:00am	morning tea	(30 min)
11:00am	-	12:30pm	lecture 2	(1.5 hr)
12:30pm	-	1:30pm	lunch	(1 hr)
1:30pm	-	3:00pm	guided exercises	(1.5 hr)
3:00pm	-	5:00pm	one-on-one help	(2 hr)

```
>>> Where are we now?
```

Day 2

1. Reading the docs
 2. Aggregating
 3. Expanding the toolkit
 4. Creating and editing tables
 5. Independent development
- Send me questions and give feedback

Page is hyperlinked: click a topic above to jump to it.

>>> Reading the docs

- * Quickly look something up when you forget syntax
- * Learn new things while you browse and decipher
- * Gain a deeper understanding
- * It actually gets easy pretty quickly

>>> Reading the docs

- * Quickly look something up when you forget syntax
- * Learn new things while you browse and decipher
- * Gain a deeper understanding
- * It actually gets easy pretty quickly

Be brave, the documentation can sense fear

>>> Data Manipulation Language (DML)

Later, we will learn some Data Definition Language (DDL).

- * [Click here for MySQL DML docs](#)

- * [Click here for T-SQL DML docs](#)

>>> Data Manipulation Language (DML)

Later, we will learn some Data Definition Language (DDL).

- * [Click here for MySQL DML docs](#)

- * [Click here for T-SQL DML docs](#)

Wait, what have we been learning?

- * [Click here for MySQL SELECT docs](#)

- * [Click here for T-SQL SELECT docs](#)

The word 'query' actually refers to `SELECT`!

>>> The syntax conventions

Why is it so hard to read? Syntax, used to make the documentation clearer and more succinct.

- * [Click here for MySQL Syntax Conventions](#)
- * [Click here for T-SQL Syntax Conventions](#)

>>> The important syntax conventions

MySQL	T-SQL	Description
[a]	[a]	a is optional
{a}	{a}	a is not optional
[a b]	[a b]	optionally, choose a or b
{a b}	{a b}	not optional, choose a or b
a [, a] ...	a [,...n]	optionally repeat a (with comma)
label	<label>	a label/placeholder

>>> The SELECT clause

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...  
[ INTO <new_table> ]  
  
[ FROM <table_source> [,...n] ]  
  
[ WHERE <search_condition> ]  
  
[ GROUP BY {col_name | expr }, ... ]  
  
[ HAVING <search_condition> ]  
  
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

Square brackets mean content is optional.

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
```

```
[ INTO <new_table> ]
```

```
[ FROM <table_source> [,...n] ]
```

```
[ WHERE <search_condition> ]
```

```
[ GROUP BY {col_name | expr }, ... ]
```

```
[ HAVING <search_condition> ]
```

```
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

Curly brackets mean content is mandatory.

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
```

```
[ INTO <new_table> ]
```

```
[ FROM <table_source> [,...n] ]
```

```
[ WHERE <search_condition> ]
```

```
[ GROUP BY {col_name | expr }, ... ]
```

```
[ HAVING <search_condition> ]
```

```
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

Vertical line means choose one.

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
```

```
[ INTO <new_table> ]
```

```
[ FROM <table_source> [,...n] ]
```

```
[ WHERE <search_condition> ]
```

```
[ GROUP BY {col_name | expr }, ... ]
```

```
[ HAVING <search_condition> ]
```

```
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

must choose one.

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
```

```
[ INTO <new_table> ]
```

```
[ FROM <table_source> [,...n] ]
```

```
[ WHERE <search_condition> ]
```

```
[ GROUP BY {col_name | expr }, ... ]
```

```
[ HAVING <search_condition> ]
```

```
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

optionally choose one.

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
```

```
[ INTO <new_table> ]
```

```
[ FROM <table_source> [,...n] ]
```

```
[ WHERE <search_condition> ]
```

```
[ GROUP BY {col_name | expr }, ... ]
```

```
[ HAVING <search_condition> ]
```

```
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

Repeat with commas (T-SQL)

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
```

```
[ INTO <new_table> ]
```

```
[ FROM <table_source> [,...n] ]
```

```
[ WHERE <search_condition> ]
```

```
[ GROUP BY {col_name | expr }, ... ]
```

```
[ HAVING <search_condition> ]
```

```
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

Repeat with commas (MySQL)

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...  
[ INTO <new_table> ]  
  
[ FROM <table_source> [,...n] ]  
  
[ WHERE <search_condition> ]  
  
[ GROUP BY {col_name | expr }, ... ]  
  
[ HAVING <search_condition> ]  
  
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

Also repeat with commas (MySQL)

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...  
[ INTO <new_table> ]  
  
[ FROM <table_source> [,...n] ]  
  
[ WHERE <search_condition> ]  
  
[ GROUP BY {col_name | expr }, ... ]  
  
[ HAVING <search_condition> ]  
  
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

>>> The SELECT clause

Labels or placeholders

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
```

```
[ INTO <new_table> ]
```

```
[ FROM <table_source> [,...n] ]
```

```
[ WHERE <search_condition> ]
```

```
[ GROUP BY {col_name | expr }, ... ]
```

```
[ HAVING <search_condition> ]
```

```
[ ORDER BY {order_by_expression [ ASC | DESC ]} [,...n] ]
```

```
>>> A discovery...
```

```
SELECT [ALL|DISTINCT] FirstName, LastName, FavColour
```

From the MySQL SELECT docs ([click here](#)):

The ALL and DISTINCT modifiers specify whether duplicate rows should be returned. ALL (the default), specifies that all matching rows should be returned, including duplicates. DISTINCT specifies removal of duplicate rows from the result set. It is an error to specify both modifiers. DISTINCTROW is a synonym for DISTINCT.

```
>>> Read the docs: FROM
```

T-SQL will often use `::=` to define placeholders.

```
FROM {<table_source>} [...n]
```

where

```
{<table_source>} ::= table_or_view_name [[AS] table_alias]
```

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [...n]
```

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [...n]
```

```
* { } curly braces group required items
```

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [...n]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax


```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

where

```
<table_source> ::= table_or_view_name [[AS] table_alias]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between
- * <label> ::= defining the placeholder

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

where

```
<table_source> ::= table_or_view_name [[AS] table_alias]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between
- * <label> ::= defining the placeholder

```
FROM MyTable, MyOtherTable
```

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

where

```
<table_source> ::= table_or_view_name [[AS] table_alias]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between
- * <label> ::= defining the placeholder
- * [] square brackets indicate optional items

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

where

```
<table_source> ::= table_or_view_name [[AS] table_alias]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between
- * <label> ::= defining the placeholder
- * [] square brackets indicate optional items

```
FROM MyTable M
```

```
>>> Reading the docs: FROM
```

```
FROM {<table_source>} [,...n]
```

where

```
<table_source> ::= table_or_view_name [[AS] table_alias]
```

- * { } curly braces group required items
- * <label> placeholder for a block of syntax
- * [,...n] means you can repeat with commas between
- * <label> ::= defining the placeholder
- * [] square brackets indicate optional items

```
FROM MyTable AS M
```



```
>>> Feeling confident?
```

```
Have a look at the T-SQL FROM documentation
```

```
>>> Feeling confident?
```

Have a look at the **T-SQL FROM** documentation

- * It really is more of the same
- * It gets easier very quickly with practice
- * Google, StackExchange, etc
- * Beginner tutorial
- * Syntax guides and **cheat sheets**

```
>>> Practice
```

```
<greeting> ::= {{Hello | Hi} [...n] .}  
              [Do you {love | hate} reading the docs?]
```

```
* Hello.
```

>>> Practice

```
<greeting> ::= {{Hello | Hi} [,...n] .}  
              [Do you {love | hate} reading the docs?]
```

* Hello.

* Hi.

>>> Practice

```
<greeting> ::= {{Hello | Hi} [...n] .}  
              [Do you {love | hate} reading the docs?]
```

* Hello.

* Hi.

* Hello. Do you love reading the docs?

>>> Practice

```
<greeting> ::= {{Hello | Hi} [...n] .}  
              [Do you {love | hate} reading the docs?]
```

- * Hello.
- * Hi.
- * Hello. Do you love reading the docs?
- * Hi. Do you love reading the docs?

>>> Practice

```
<greeting> ::= {{Hello | Hi} [,...n] .}  
              [Do you {love | hate} reading the docs?]
```

- * Hello.
- * Hi.
- * Hello. Do you love reading the docs?
- * Hi. Do you love reading the docs?
- * Hello, Hello, Hi, Hello, Hi, Hi.

>>> Practice

```
<greeting> ::= {{Hello | Hi} [,...n] .}  
              [Do you {love | hate} reading the docs?]
```

- * Hello.
- * Hi.
- * Hello. Do you love reading the docs?
- * Hi. Do you love reading the docs?
- * Hello, Hello, Hi, Hello, Hi, Hi.
- * Hello, Hi, Hello, Hello. Do you love reading the docs?

>>> Practice

```
<greeting> ::= {{Hello | Hi} [,...n] .}  
              [Do you {love | hate} reading the docs?]
```

- * Hello.
- * Hi.
- * Hello. Do you love reading the docs?
- * Hi. Do you love reading the docs?
- * Hello, Hello, Hi, Hello, Hi, Hi.
- * Hello, Hi, Hello, Hello. Do you love reading the docs?
- * etc.

```
>>> Example from the docs (logical operator IN)
```

```
test_expression [ NOT ] IN ( subquery | expression [ ,...n ] )
```

Don't miss the round brackets!

```
>>> Example from the docs (logical operator IN)
```

```
test_expression [ NOT ] IN ( subquery | expression [ ,...n ] )
```

```
* test_expression IN (expression)
```

```
Example: FavColour IN ('red')
```

>>> Example from the docs (logical operator IN)

```
test_expression [ NOT ] IN ( subquery | expression [ ,...n ] )
```

- * test_expression IN (expression)

- * test_expression IN (subquery)

Example: FriendID IN (SELECT FriendID FROM Notes.Pets)

```
>>> Example from the docs (logical operator IN)
```

```
test_expression [ NOT ] IN ( subquery | expression [ ,...n ] )
```

- * test_expression IN (expression)
- * test_expression IN (subquery)
- * test_expression NOT IN (expression)

```
Example: FavColour NOT IN ('red')
```

>>> Example from the docs (logical operator IN)

```
test_expression [ NOT ] IN ( subquery | expression [ ,...n ] )
```

- * test_expression IN (expression)
- * test_expression IN (subquery)
- * test_expression NOT IN (expression)
- * test_expression NOT IN (subquery)

Example: FriendID NOT IN (SELECT FriendID FROM Notes.Pets)

>>> Example from the docs (logical operator IN)

```
test_expression [ NOT ] IN ( subquery | expression [ ,...n ] )
```

- * test_expression IN (expression)
- * test_expression IN (subquery)
- * test_expression NOT IN (expression)
- * test_expression NOT IN (subquery)
- * test_expression NOT IN (expression, expression, expression)

Example: FavColour IN ('red', 'blue', 'green')

```
>>> Kahoot number five
```

Enjoy.


```
>>> Where are we now?
```

Day 2

1. Reading the docs
 2. **Aggregating**
 3. Expanding the toolkit
 4. Creating and editing tables
 5. Independent development
- Send me questions and give feedback

Page is hyperlinked: click a topic above to jump to it.

>>> Aggregating queries

Aggregating queries collect the rows of a table into groups, and return a single value for each group. They can also discard groups.

We will cover:

1. GROUP BY clause
2. Aggregation function
3. HAVING clause

>>> Aggregating queries

Aggregating queries collect the rows of a table into groups, and return a single value for each group. They can also discard groups.

We will cover:

1. GROUP BY clause - creates the groups
2. Aggregation function
3. HAVING clause

>>> Aggregating queries

Aggregating queries collect the rows of a table into groups, and return a single value for each group. They can also discard groups.

We will cover:

1. GROUP BY clause - creates the groups
2. Aggregation function - computes one value for each group
3. HAVING clause

>>> Aggregating queries

Aggregating queries collect the rows of a table into groups, and return a single value for each group. They can also discard groups.

We will cover:

1. GROUP BY clause - creates the groups
2. Aggregation function - computes one value for each group
3. HAVING clause - discards groups

The HAVING clause discards groups just like the WHERE clause discards rows.

```
>>> SQL clause: GROUP BY
```

The `GROUP BY` clause groups the rows of a table according to the values of one or more columns. The easiest way to understand it is with examples.

We will look at the execution of this query:

```
SELECT P.friendID  
FROM Notes.Pets P  
GROUP BY P.friendID;
```

```
>>> SQL clause: GROUP BY
```

```
FROM Notes.Pets P
```

Pets			
PetID	PetName	PetDOB	FriendID
1	Chikin	24/09/2016	2
2	Cauchy	01/03/2012	3
3	Gauss	01/03/2012	3

```
>>> SQL clause: GROUP BY
```

GROUP BY P.FriendID

Pets			
PetID	PetName	PetDOB	FriendID
1	Chikin	24/09/2016	2
2	Cauchy	01/03/2012	3
3	Gauss	01/03/2012	3


```
>>> SQL clause: GROUP BY
```

GROUP BY P.FriendID

Unnamed			
PetID	PetName	PetDOB	FriendID
{2}	{Chikin}	{24/09/2016}	2
{2,3}	{Cauchy, Gauss}	{01/03/2012, 01/03/2012}	3

```
>>> SQL clause: GROUP BY
```

```
SELECT P.FriendID
```

Unnamed			
PetID	PetName	PetDOB	FriendID
{2}	{Chikin}	{24/09/2016}	2
{2,3}	{Cauchy, Gauss}	{01/03/2012, 01/03/2012}	3

```
>>> SQL clause: GROUP BY
```

result

Unnamed
FriendID
2
3

```
>>> Can we select any of the other columns?
```

```
SELECT ???
```

Unnamed			
PetID	PetName	PetDOB	FriendID
{2}	{Chikin}	{24/09/2016}	2
{2,3}	{Cauchy, Gauss}	{01/03/2012, 01/03/2012}	3

Error: SQL can't be sure the entries are atomic.

>>> What will happen if we run this?

```
SELECT P.friendID, P.petDOB  
FROM Notes.Pets P  
GROUP BY P.friendID;
```

```
>>> Error
```

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

```
>>> This will fix the error
```

```
SELECT P.friendID, P.petDOB  
FROM Notes.Pets P  
GROUP BY P.friendID, P.petDOB;
```

```
>>> SQL clause: GROUP BY
```

```
GROUP BY P.FriendID, P.PetDOB
```

Pets			
PetID	PetName	PetDOB	FriendID
1	Chikin	24/09/2016	2
2	Cauchy	01/03/2012	3
3	Gauss	01/03/2012	3

```
>>> SQL clause: GROUP BY
```

GROUP BY P.FriendID, P.PetDOB

Unnamed			
PetID	PetName	PetDOB	FriendID
{2}	{Chikin}	24/09/2016	2
{2,3}	{Cauchy, Gauss}	01/03/2012	3

```
>>> More examples
```

Watch the curly braces

Letters		
<i>A</i>	<i>B</i>	Num
a	b	1
a	c	2
a	b	3
a	c	4

- * GROUP BY B
- * GROUP BY A
- * GROUP BY A, B

```
>>> Solutions
```

```
* GROUP BY B
```

Unnamed		
A	B	Num
{a, a}	b	{1, 3}
{a, a}	c	{2, 4}

>>> Solutions

* GROUP BY B

Unnamed		
<i>A</i>	<i>B</i>	Num
{a, a}	b	{1, 3}
{a, a}	c	{2, 4}

* GROUP BY A

Unnamed		
<i>A</i>	<i>B</i>	Num
a	{b, c, b, c}	{1, 2, 3, 4}

>>> Solutions

* GROUP BY B

Unnamed		
<i>A</i>	<i>B</i>	Num
{a, a}	b	{1, 3}
{a, a}	c	{2, 4}

* GROUP BY A

Unnamed		
<i>A</i>	<i>B</i>	Num
a	{b, c, b, c}	{1, 2, 3, 4}

* GROUP BY A, B

Unnamed		
<i>A</i>	<i>B</i>	Num
a	b	{1, 3}
a	c	{2, 4}

```
>>> Aggregation functions
```

Aggregation functions compute one value for each group.
Aggregating frees us to select more columns.

We will look at the execution of this query:

```
SELECT RP.gender, AVG(RP.age) AS AverageAge  
FROM Notes.RandomPeople RP  
GROUP BY RP.gender;
```

```
>>> Aggregation functions
```

Aggregation functions compute one value for each group.
Aggregating frees us to select more columns.

We will look at the execution of this query:

```
SELECT RP.gender, AVG(RP.age) AS AverageAge  
FROM Notes.RandomPeople RP  
GROUP BY RP.gender;
```

Aggregation function


```
>>> Aggregation functions
```

Aggregation functions compute one value for each group.
Aggregating frees us to select more columns.

We will look at the execution of this query:

```
SELECT RP.gender, AVG(RP.age) AS AverageAge  
FROM Notes.RandomPeople RP  
GROUP BY RP.gender;
```

Column name alias

```
>>> Aggregation function: AVG
```

```
FROM Notes.RandomPeople RP
```

RandomPeople		
Name	Gender	Age
Beyoncé	F	37
Laura Marling	F	28
Darren Hayes	M	46
Bret McKenzie	M	42
Jack Monroe	NB	30

```
>>> Aggregation function: AVG
```

GROUP BY RP.Gender

RandomPeople		
Name	Gender	Age
Beyoncé	F	37
Laura Marling	F	28
Darren Hayes	M	46
Bret McKenzie	M	42
Jack Monroe	NB	30

```
>>> Aggregation function: AVG
```

GROUP BY RP.Gender

Unnamed		
Name	Gender	Age
{Beyoncé, Laura Marling}	F	{37, 28}
{Darren Hayes, Bret McKenzie}	M	{46, 42}
{Jack Monroe}	NB	{30}

```
>>> Aggregation function: AVG
```

AVG(RP.Age)

Unnamed		
Name	Gender	(unnamed)
{Beyoncé, Laura Marling}	F	AVG({37,28})
{Darren Hayes, Bret McKenzie}	M	AVG({46,42})
{Jack Monroe}	NB	AVG({30})

```
>>> Aggregation function: AVG
```

AVG(RP.Age)

Unnamed		
Name	Gender	(unnamed)
{Beyoncé, Laura Marling}	F	32.5
{Darren Hayes, Bret McKenzie}	M	44
{Jack Monroe}	NB	30

```
>>> Aggregation function: AVG
```

```
SELECT RP.Gender, AVG(RP.Age) AS AverageAge
```

Unnamed		
Name	Gender	(unnamed)
{Beyoncé, Laura Marling}	F	32.5
{Darren Hayes, Bret McKenzie}	M	44
{Jack Monroe}	NB	30

```
>>> Aggregation function: AVG
```

result

Unnamed	
Gender	AverageAge
F	32.5
M	44
NB	30

We retrieved the average age for each gender in the table!

>>> What happens if we throw in a WHERE clause?

We will look at the execution of this query:

```
SELECT RP.gender, AVG(RP.age) AS AverageAge
FROM Notes.RandomPeople RP
WHERE RP.gender = 'F'
GROUP BY RP.gender;
```

>>> What happens if we throw in a WHERE clause?

FROM Notes.RandomPeople RP

RandomPeople		
Name	Gender	Age
Beyoncé	F	37
Laura Marling	F	28
Darren Hayes	M	46
Bret McKenzie	M	42
Jack Monroe	NB	30

>>> What happens if we throw in a WHERE clause?

WHERE RP.Gender = 'F'

RandomPeople		
Name	Gender	Age
Beyoncé	F	37
Laura Marling	F	28
Darren Hayes	M	46
Bret McKenzie	M	42
Jack Monroe	NB	30

>>> What happens if we throw in a WHERE clause?

GROUP BY RP.Gender

Unnamed		
Name	Gender	Age
{Beyoncé, Laura Marling}	F	{37, 28}

>>> What happens if we throw in a WHERE clause?

AVG(RP.Age)

Unnamed		
Name	Gender	(unnamed)
{Beyoncé, Laura Marling}	F	32.5

>>> What happens if we throw in a WHERE clause?

```
SELECT RP.Gender, AVG(RP.Age) AS AverageAge
```

Unnamed		
Name	Gender	(unnamed)
{Beyoncé, Laura Marling}	F	32.5

```
>>> What happens if we throw in a WHERE clause?
```

```
result
```

Unnamed	
Gender	AverageAge
F	32.5

We retrieved the average age for females in the table!


```
>>> Order of execution
```

1. FROM
2. WHERE
- 3.
- 4.
- 5.

```
>>> Order of execution
```

1. FROM
2. WHERE
3. GROUP BY
- 4.
- 5.

```
>>> Order of execution
```

1. FROM
2. WHERE
3. GROUP BY
4. Aggregation
- 5.

```
>>> Order of execution
```

1. FROM
2. WHERE
3. GROUP BY
4. Aggregation
5. SELECT

>>> More aggregation functions

T-SQL	MySQL	Purpose
AVG	AVG	Average
STDEV	STDDEV_SAMP	Sample standard deviation
STDEVP	STDDEV_POP	Population standard deviation
VAR	VAR_SAMP	Sample variance
VARP	VAR_POP	Population variance
COUNT	COUNT	Count number of rows
MIN	MIN	Minimum
MAX	MAX	Maximum
SUM	SUM	Sum

See the full list in the [T-SQL](#) or [MySQL](#) docs (click).

>>> More aggregation functions

T-SQL	MySQL	Purpose
AVG	AVG	Average
STDEV	STDDEV_SAMP	Sample standard deviation
STDEVP	STDDEV_POP	Population standard deviation
VAR	VAR_SAMP	Sample variance
VARP	VAR_POP	Population variance
COUNT	COUNT	Count number of rows
MIN	MIN	Minimum
MAX	MAX	Maximum
SUM	SUM	Sum

See the full list in the [T-SQL](#) or [MySQL](#) docs (click).

```
>>> SQL clause: HAVING
```

The HAVING clause was created because WHERE is executed before GROUP BY.

We will look at the execution of this query:

```
SELECT RP.gender, AVG(RP.age) AS AverageAge  
FROM Notes.RandomPeople RP  
GROUP BY RP.gender  
HAVING AVG(RP.age) > 40;
```

The HAVING clause is like WHERE, but it acts on groups.

```
>>> SQL clause: HAVING
```

The HAVING clause was created because WHERE is executed before GROUP BY.

We will look at the execution of this query:

```
SELECT RP.gender, AVG(RP.age) AS AverageAge  
FROM Notes.RandomPeople RP  
GROUP BY RP.gender  
HAVING AVG(RP.age) > 40;
```

Search condition with aggregation function

The HAVING clause is like WHERE, but it acts on groups.


```
>>> SQL clause: HAVING
```

```
FROM Notes.RandomPeople RP
```

RandomPeople		
Name	Gender	Age
Beyoncé	F	37
Laura Marling	F	28
Darren Hayes	M	46
Bret McKenzie	M	42
Jack Monroe	NB	30

```
>>> SQL clause: HAVING
```

GROUP BY RP.Gender

Unnamed		
Name	Gender	Age
{Beyoncé, Laura Marling}	F	{37, 28}
{Darren Hayes, Bret McKenzie}	M	{46, 42}
{Jack Monroe}	NB	{30}

```
>>> SQL clause: HAVING
```

AVG(RP.Age)

Unnamed		
Name	Gender	(unnamed)
{Beyoncé, Laura Marling}	F	32.5
{Darren Hayes, Bret McKenzie}	M	44
{Jack Monroe}	NB	30

```
>>> SQL clause: HAVING
```

```
HAVING AVG(RP.Age) > 40
```

Unnamed		
Name	Gender	(unnamed)
{Beyoncé, Laura Marling}	F	32.5
{Darren Hayes, Bret McKenzie}	M	44
{Jack Monroe}	NB	30

```
>>> SQL clause: HAVING
```

```
SELECT RP.Gender, AVG(RP.Age) AS AverageAge
```

Unnamed		
Name	Gender	(unnamed)
{Darren Hayes, Bret McKenzie}	M	44

```
>>> SQL clause: HAVING
```

result

Unnamed	
Gender	AverageAge
M	44


```
>>> Order of execution
```

1. FROM
2. WHERE
- 3.
- 4.
- 5.
- 6.


```
>>> Order of execution
```

1. FROM
2. WHERE
3. GROUP BY
- 4.
- 5.
- 6.

```
>>> Order of execution
```

1. FROM
2. WHERE
3. GROUP BY
4. Aggregation
- 5.
- 6.

```
>>> Order of execution
```

1. FROM
2. WHERE
3. GROUP BY
4. Aggregation
5. HAVING
- 6.

>>> Order of execution

1. FROM
2. WHERE
3. GROUP BY
4. Aggregation
5. HAVING
6. SELECT

>>> Group practice

The aggregation function in the `HAVING` clause does not have to match the one in the `SELECT` clause.

```
SELECT RP.gender, STDEV(RP.age) AS AverageAge
FROM Notes.RandomPeople RP
GROUP BY RP.gender
HAVING AVG(RP.age) > 40;
```

Explain in words what the above query achieves.

```
>>> Solution
```

The query finds the sample standard deviation of the ages for each gender that has an average age greater than 40.

```
>>> How about this one?
```

Find all the people whose Gender has `AVG(Age)` less than 40.

Easy, right?

>>> How about this one?

Find all the people whose Gender has AVG(Age) less than 40.

Easy, right?

```
SELECT Name
FROM RandomPeople
GROUP BY Gender
HAVING AVG(Age) < 40;
```

>>> How about this one?

Find all the people whose Gender has AVG(Age) less than 40.

Easy, right?

```
SELECT Name
FROM RandomPeople
GROUP BY Gender
HAVING AVG(Age) < 40;
```

Wrong!

Let's try and execute it.

>>> Can you explain why?

FROM RandomPeople GROUP BY Gender

Unnamed		
Name	Gender	Age
(Beyoncé, Laura Marling)	F	(37, 28)
(Darren Hayes, Bret McKenzie)	M	(46, 42)
(Jack Monroe)	NB	(30)

```
SELECT Name
FROM RandomPeople
GROUP BY Gender
HAVING AVG(Age) < 40;
```

>>> Use a subquery instead

```
SELECT Name
FROM RandomPeople
WHERE Gender IN (SELECT Gender
                  FROM RandomPeople
                  GROUP BY Gender
                  HAVING AVG(Age) < 40);
```

Let's execute it and explore.

>>> Correlated subquery

The hardest thing in introductory SQL...

```
SELECT Name
FROM RandomPeople RP
WHERE age > (SELECT AVG(age)
             FROM RandomPeople
             WHERE gender = RP.gender);
```

>>> Correlated subquery

The hardest thing in introductory SQL...

```
SELECT Name
FROM RandomPeople RP
WHERE age > (SELECT AVG(age)
             FROM RandomPeople
             WHERE gender = RP.gender);
```

The notes have a good diagram for this...


```
>>> Where are we now?
```

Day 2

1. Reading the docs
2. Aggregating
3. Expanding the toolkit
4. Creating and editing tables
5. Independent development

Send me questions and give feedback

Page is hyperlinked: click a topic above to jump to it.

```
>>> Group practice
```

Figure out what `UNION` does.

- * [Click here for T-SQL `UNION` documentation.](#)
- * [Click here for MySQL `UNION` documentation.](#)

>>> Live demonstration of UNION

Starting with this...

```
SELECT F.FirstName AS FirstInitial,  
       F.LastName AS LastInitial,  
       ColourName  
FROM Ape.Friends F LEFT JOIN Ape.Colours C  
   ON F.FavColourID = C.ColourID;
```

```
>>> Group practice
```

Figure out what `CAST` does.

- * [Click here for T-SQL `CAST` docs.](#)
- * [Click here for MySQL `CAST` docs.](#)

>>> Casting - aggregation warning!

Up to this point we have largely ignored data types. This can go on no longer. Arithmetic with integers always returns an integer (by rounding down).

$$\text{AVG}(\{1,2\}) = 1$$

So we need to use CAST.

>>> Casting - aggregation warning!

Up to this point we have largely ignored data types. This can go on no longer. Arithmetic with integers always returns an integer (by rounding down).

$$\text{AVG}(\{1,2\}) = 1$$

So we need to use CAST. For example, in `Notes.RandomPeople`, the data type for Age is integer.

```
SELECT gender, AVG(age) AS AverageAge
FROM Notes.RandomPeople
GROUP BY gender;
```

Must be changed to:

```
SELECT gender, AVG(CAST(age AS decimal)) AS AverageAge
FROM Notes.RandomPeople
GROUP BY gender;
```

```
>>> Group practice
```

Figure out what `COUNT` and `COUNT(DISTINCT)` do.

- * [Click here for T-SQL `COUNT` documentation.](#)
- * [Click here for MySQL `COUNT` documentation.](#)

```
>>> Live demo
```

Starting with this...

```
SELECT COUNT(TasteRank)
FROM Ape.Banana;
```

```
>>> Group practice
```

Figure out what `WITH` does.

- * [Click here for T-SQL `WITH` documentation.](#)

- * [Click here for MySQL `WITH` documentation.](#)

>>> Live demo

Reducing repetition via WITH.

```
SELECT BananaID, TasteRank, Ripe,  
       CASE WHEN Ripe = 1 AND TasteRank = 5 THEN 'Ripe and tasty'  
       ELSE 'Imperfect' END AS Category  
FROM Ape.Banana  
GROUP BY Category;
```

```
>>> Many other functions
```

There are many other functions...

- * [Click here for T-SQL functions](#)

- * [Click here for MySQL functions](#)

Check out the notes section on window functions.

```
>>> Where are we now?
```

Day 2

1. Reading the docs
 2. Aggregating
 3. Expanding the toolkit
 4. Creating and editing tables
 5. Independent development
- Send me questions and give feedback

Page is hyperlinked: click a topic above to jump to it.

>>> Live demonstration

- * CREATE DATABASE and DROP DATABASE
- * CREATE SCHEMA (for T-SQL only)
- * CREATE VIEW to store a query like a table
- * SELECT INTO and CREATE TABLE ... SELECT
- * INSERT INTO to create a whole record
- * CREATE TABLE and DROP TABLE
- * ALTER to add columns to a stored table
- * UPDATE to change the entries in a table

You can also see all of the above in the notes.

```
>>> Where are we now?
```

Day 2

1. Reading the docs
2. Aggregating
3. Expanding the toolkit
4. Creating and editing tables
5. Independent development

Send me questions and give feedback

Page is hyperlinked: click a topic above to jump to it.

```
>>> Independent development
```

Live walk-through using StackExchange database.