

Learning MySQL By Example

Mathew Miles

Table of Contents

Introduction	1
1. How to Retrieve Data From a Single Table	3
1.1. The Five Clauses of the SELECT Statement	17
1.2. Column Specifications	21
1.3. LIKE and REGEXP Operators	23
1.4. Arithmetic Operators	27
1.5. Column Aliases	29
1.6. Comparison Operators	31
1.7. IS NULL, BETWEEN, IN Operators	33
1.8. AND, OR, NOT Logical Operators	37
1.9. DISTINCT Clause	39
2. How to Retrieve Data from Multiple Tables	41
2.1. The JOIN Clause	43
2.2. Joining More Than Two Tables	47
2.3. The OUTER JOIN Clause	49
2.4. How to Code a UNION	53
3. Using Functions	57
3.1. Date Functions	59
3.2. Numeric Functions	67
3.3. String Functions	71
4. How to Insert, Update, Delete Data in Tables	77
4.1. The INSERT Clause With a Column List	79
4.2. The INSERT Clause Without a Column List	83
4.4. The UPDATE Clause With a Column List	85
4.4. The DELETE Clause	87
5. Summary Queries and Aggregate Functions	89
5.1. Aggregate Functions	91

5.2. Grouping Data	93
5.3. Simple GROUP BY Query	95
5.4. Improving the GROUP BY Query	97
5.5. Using the HAVING Clause	101
5.5. Using the HAVING and WHERE Clauses Together	103
5.6. COUNT(column_name) and COUNT(*)	105
5.7. Using the DISTINCT Statement	107
6. Working With Subqueries	109
6.1. The Subquery In a SELECT Statement	111
6.2. The Subquery in an UPDATE statement	115
6.3. Create a Duplicate Table From An Existing Table	117
6.4. The Subquery In a Delete Statement	119
7. SQL Views	123
7.1. SQL View Explained	125
7.2. Benefits of Using Views	127
7.3. Views That Allow UPDATE Statements	131
8. SQL Indexes	133
8.1. SQL Indexes Explained	135
8.2. Clustered vs. Non-clustered Indexes	137
8.3. Create an Index in Workbench Using an ERD	139
8.4. How to Manually Add an Index to an Existing Table	141
Glossary	143
Index	149



Mathew Miles



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql.

Introduction

Before You Begin

This book has examples from two databases that you can download and install in your local MySQL instance. You can practice by copying the code snippets from the chapters and running them in your local environment. In order for this book to be most effective, you must have MySQL installed locally on your machine and have also installed MySQL Workbench.

In a future edition, this book will include SQL design basics and guidance on how to install MySQL and MySQL Workbench. It may also include screencasts that will walk you through the various concepts.

[Click here to download the Bikes database](#)

[Click here to download the World database](#)



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/introduction.

How to Retrieve Data From a Single Table

The Five Clauses of the SELECT statement

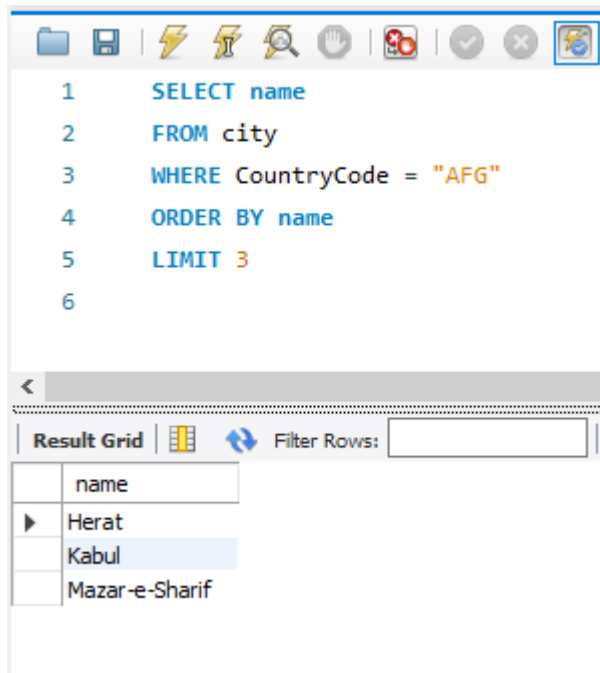
- SELECT – the columns in the result set
- FROM – names the base table(s) from which results will be retrieved
- WHERE – specifies any conditions for the results set (filter)
- ORDER BY – sets how the result set will be ordered
- LIMIT – sets the number of rows to be returned

The clauses **MUST** appear in the order shown above.

Code Example:

```
1  USE world;  
2  SELECT name  
3  FROM city  
4  WHERE CountryCode = "AFG"  
5  ORDER BY name  
6  LIMIT 3
```

Results:



Let us break the statement line by line:

USE world;

- The **USE** clause sets the database that we will be querying. You typically have more than one database on your database server. You have to specify which database you are working in.
- The semicolon ";" indicates the end of a statement. You can execute multiple statements in sequence by defining each statement with a semicolon

SELECT name

- The **SELECT** clause defines the columns and column order that you want to retrieve in your results set. If you want to retrieve all of the columns from the base table you can simply use `SELECT *`
- You separate each column name with a comma "," ex., `SELECT name, CountryCode`
- There is no trailing comma at the end of a column list

FROM city

- The **FROM** clause specifies the table that the results will be coming from
- You can specify multiple tables by using a `JOIN` clause, but we will address that topic at a future time

ORDER BY name

- The **ORDER BY** clause is not required but when used it defines the sort order of the results
- By default, the sort order is ascending. This is *implicit*. However, you can use *explicit* syntax of `ASC`. If you want the sort, order to be descending you can use the keyword `DESC`.
- You can specify more than one column in an Order By statement separated by commas. The sort order `DESC`, `ASC` applies to each column individually. Below are some examples
 - **ORDER BY** population ASC, name DESC
 - **ORDER BY** population, name (ASC is always implied if not explicitly stated)

LIMIT 5;

- If you only want to return a specified number of rows from the result set, you can use the LIMIT clause. This can be helpful when you want to test a query for accuracy that could potentially bring back a very large number of rows.
- The semicolon ; defines the end of the statement

Table 1. Column Specifications

Source	Option	Syntax
Base Table Value	Show all columns	
Base Table Value	Column Name	Comma separated list of column names
Calculated Value	Calculation result	Arithmetic expression
Calculated Value	Calculation result	Functions

LIKE and REGEXP Operators

- The LIKE keyword is used with the WHERE clause.
- The LIKE keyword can use two symbols as wildcards. The percent (%) symbol matches any number of characters and the underscore (_) matches a single character
- REGEXP keyword allows you to do more complex pattern matching than a LIKE keyword/
- Some version of REGEXP exists in many computer languages. Refer to the “LIKE and REGEXP” handout for a full list of examples.

Table 2. LIKE Keyword

LIKE Symbol	Description
%	Match any string of characters to the left of the symbol
_	Match a single character

Code Example:

```
USE world;
SELECT name
FROM country
WHERE name LIKE 'A%'
```

Results:

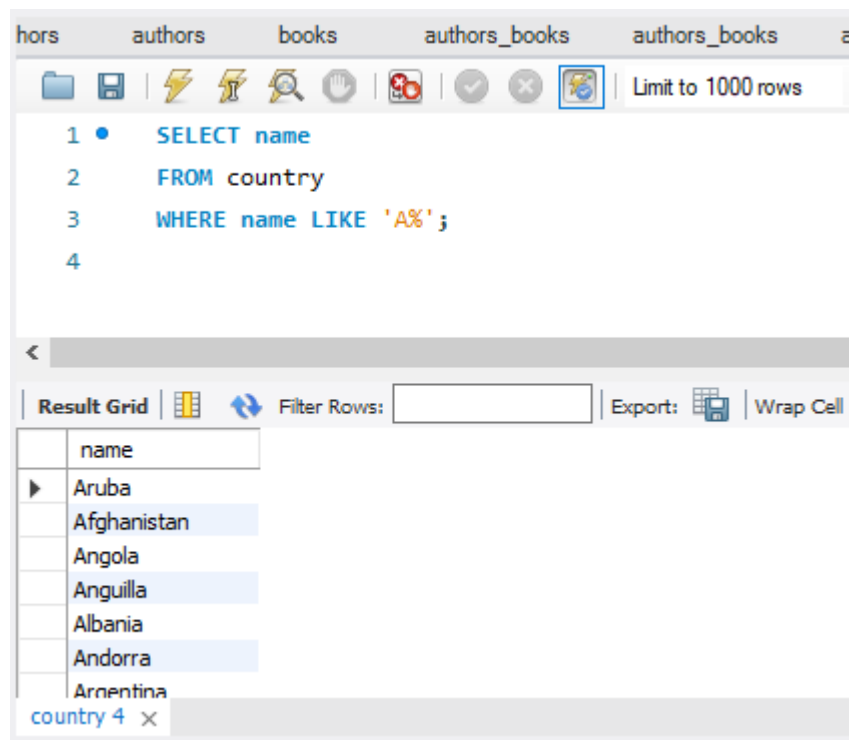


Table 3. REGEXP Keyword

REGEXP Characters	Description
^	Match the pattern to the beginning of the value being tested.
\$	Match the pattern to the end of the value being tested.
.	Matches any single character.
[charlist]	Matches any single character listed within the brackets.
[char1 – char2]	Matches any single character within the given range.
	Separates two string patterns and matches either one

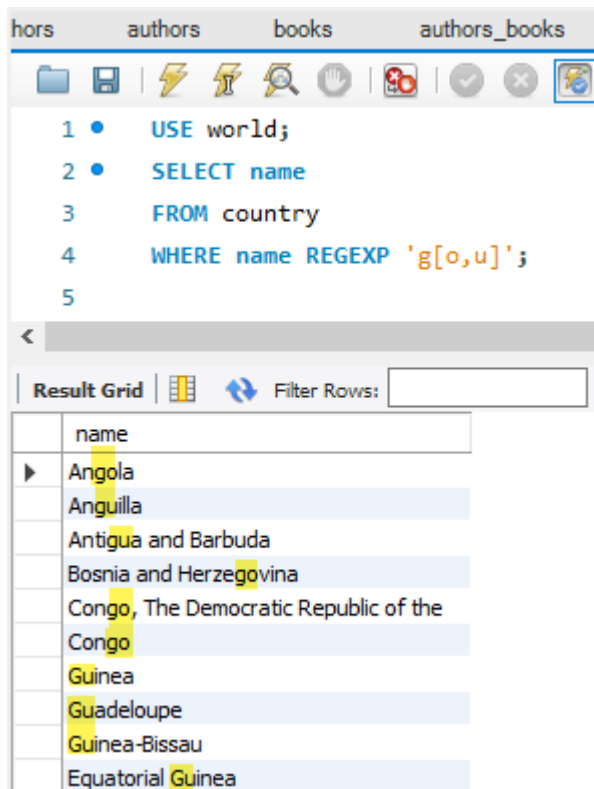
Code Example:

```

USE world;
SELECT name
FROM country
WHERE name REGEXP 'g[o,u]';

```

Results:



Arithmetic Operators

- Arithmetic operators can be used in the SELECT, WHERE, and ORDER BY clauses.
- Operators are evaluated in the same way as arithmetic in other contexts.

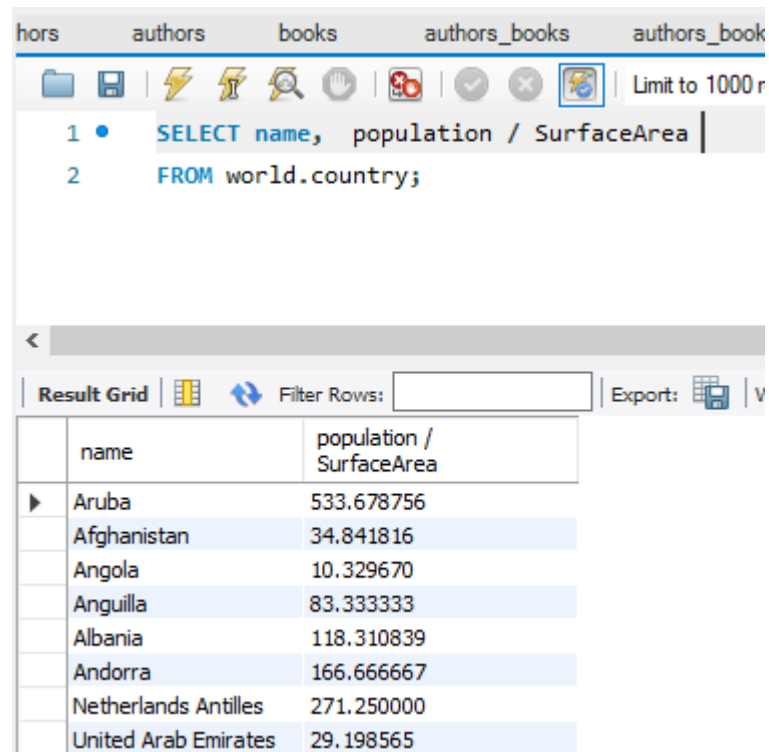
Table 4. Operators and precedence order

Operator	Name	Order of Precedence
*	Multiplication	1
/	Division	1
DIV	Integer Division	1
%(MOD)	Modulo (remainder)	1
+	Addition	2
-	Subtraction	2

Code Example:

```
USE world;
SELECT name, population / SurfaceArea
AS "People per square mile"
FROM country;
```

Results:



The screenshot shows a database query interface with a toolbar and a results grid. The SQL query is: `SELECT name, population / SurfaceArea FROM world.country;` The results grid displays the following data:

	name	population / SurfaceArea
▶	Aruba	533.678756
	Afghanistan	34.841816
	Angola	10.329670
	Anguilla	83.333333
	Albania	118.310839
	Andorra	166.666667
	Netherlands Antilles	271.250000
	United Arab Emirates	29.198565

Column Aliases

- A column alias provides a way to create a clean or more descriptive header for a results set.
- A column alias **cannot** be used in a SELECT, WHERE, GROUP BY or HAVING clause due to the order of execution. You must refer to the original column name.

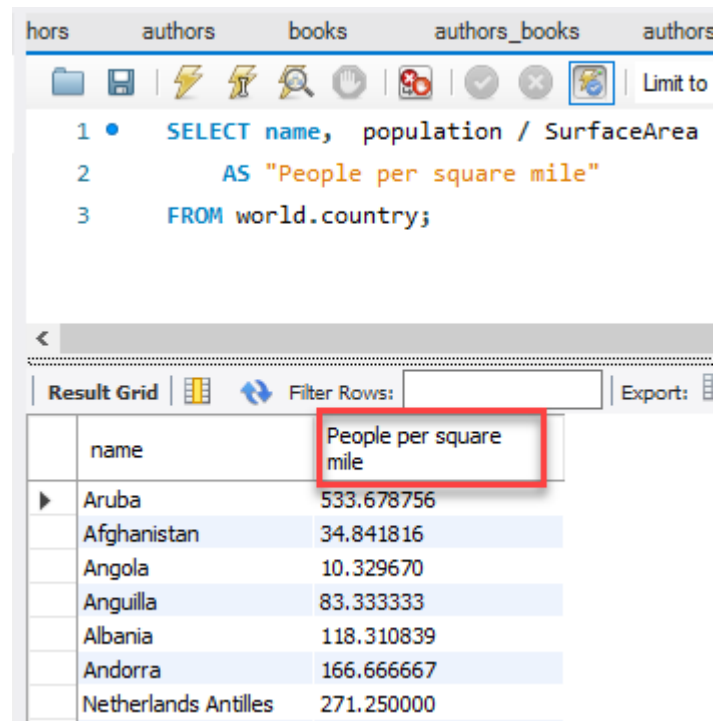
In the previous example, we created a new column that was a *calculated value*. The problem is that the column header is now population / SurfaceArea. However we can rename the column header to something cleaner by creating a *column alias*. Look at the code snippet below.

Code Example:

```
SELECT name, population / SurfaceArea
      AS "People per square mile"
FROM country;
```

We used the AS keyword then in quotes we put the new column alias of "People per square mile." Which changes the column header as seen show below.

Results:



The screenshot shows a database query interface. At the top, there are tabs for 'hors', 'authors', 'books', 'authors_books', and 'authors'. Below the tabs is a toolbar with various icons. The main area displays a SQL query:

```
1 • SELECT name, population / SurfaceArea
2     AS "People per square mile"
3     FROM world.country;
```

Below the query, there is a 'Result Grid' section. It includes a 'Filter Rows:' input field and an 'Export:' button. The result grid shows a table with two columns: 'name' and 'People per square mile'. The first row is highlighted in blue and shows 'Aruba' and '533.678756'. The second row shows 'Afghanistan' and '34.841816'. The third row shows 'Angola' and '10.329670'. The fourth row shows 'Anguilla' and '83.333333'. The fifth row shows 'Albania' and '118.310839'. The sixth row shows 'Andorra' and '166.666667'. The seventh row shows 'Netherlands Antilles' and '271.250000'.

name	People per square mile
Aruba	533.678756
Afghanistan	34.841816
Angola	10.329670
Anguilla	83.333333
Albania	118.310839
Andorra	166.666667
Netherlands Antilles	271.250000

Comparison Operators

- Comparison operators compare two expressions.
- The result of a comparison results to true or false.
- Comparison operators are not case sensitive and are used with text and dates as well as numbers.

Table 5. Comparison Operators

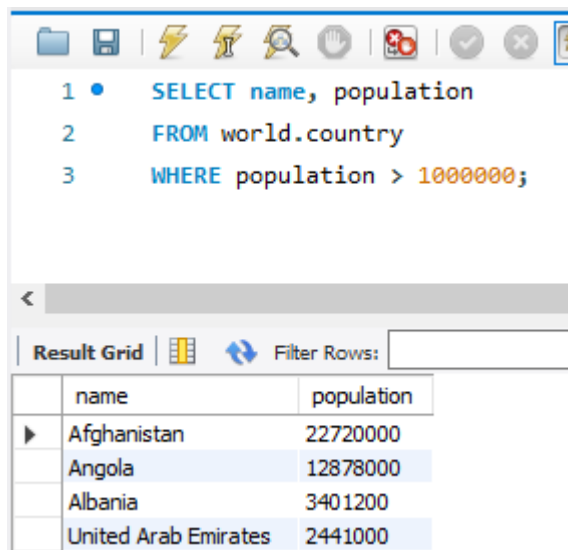
Operator	Description
=	Equal
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal

Operator	Description
!=	Not equal

Code Example:

```
USE world;
SELECT name, population
FROM country
WHERE population > 1000000;
```

Results:



The screenshot shows a SQL query editor with a toolbar at the top. The query text is as follows:

```
1 • SELECT name, population
2 FROM world.country
3 WHERE population > 1000000;
```

Below the query editor, there is a 'Result Grid' tab. The results are displayed in a table with two columns: 'name' and 'population'.

	name	population
▶	Afghanistan	22720000
	Angola	12878000
	Albania	3401200
	United Arab Emirates	2441000

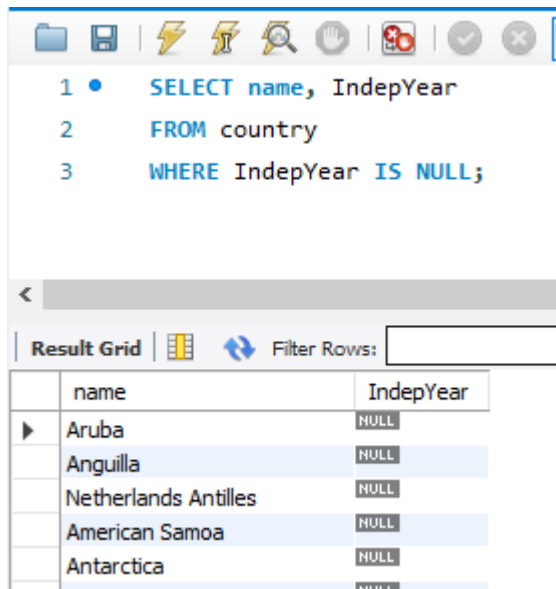
IS NULL

- *Null values* indicate an unknown or non-existent value and is different from an empty string ('').
- To test for a *null value* you use the IS NULL clause
- The test for a value use IS NOT NULL clause

Code Example:

```
SELECT name, IndepYear
FROM country
WHERE IndepYear IS NULL;
```

Results:



```

1 • SELECT name, IndepYear
2 FROM country
3 WHERE IndepYear IS NULL;

```

Result Grid

	name	IndepYear
▶	Aruba	NULL
	Anguilla	NULL
	Netherlands Antilles	NULL
	American Samoa	NULL
	Antarctica	NULL

BETWEEN Operators

- The BETWEEN operator is similar to \geq and \leq .
- BETWEEN includes everything between the two values indicated.
- BETWEEN works with both text and number.

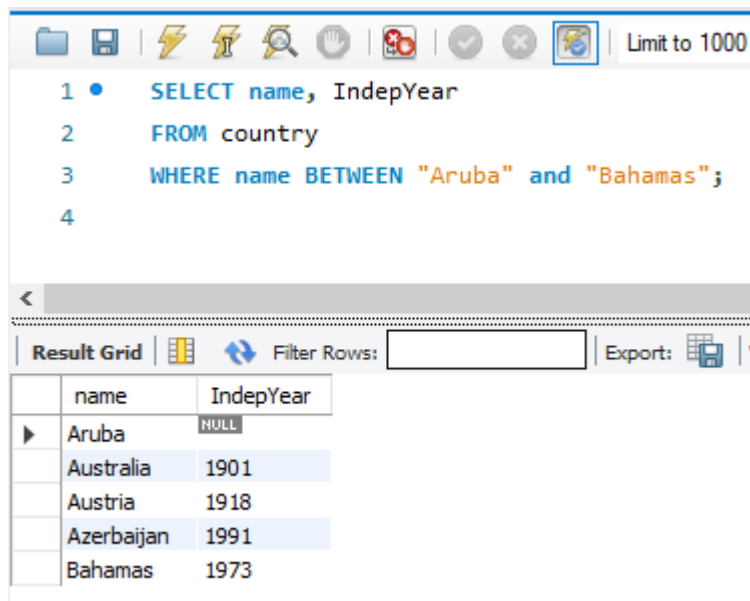
Code Example:

```

USE world;
SELECT name, IndepYear
FROM country
WHERE name BETWEEN "Aruba" and "Bahamas";

```

Results:



The screenshot shows a database query editor with a toolbar at the top containing icons for file operations, execution, and search. The SQL query is as follows:

```

1 • SELECT name, IndepYear
2 FROM country
3 WHERE name BETWEEN "Aruba" and "Bahamas";
4

```

Below the query editor, the results are displayed in a table grid. The table has two columns: 'name' and 'IndepYear'. The results are:

name	IndepYear
Aruba	NULL
Australia	1901
Austria	1918
Azerbaijan	1991
Bahamas	1973

The IN Keyword

- The IN clause tests whether an expression is equal to a value or values in a list of expressions.
- The order of the items in the list does not matter.
- You can use the NOT operator to test for items not in the list.
- The IN clause may be used with a subquery.

Code Example:

```

USE world;
SELECT name
FROM country
WHERE name IN ('Aruba', 'Barbados', 'Cuba', 'Bahamas')
ORDER BY population ASC;

```

Results:

```

1 • USE world;
2 • SELECT name
3   FROM country
4   WHERE name IN ('Aruba', 'Barbados', 'Cuba', 'Bahamas')
5   ORDER BY population ASC;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content

	name
▶	Aruba
	Barbados
	Bahamas
	Cuba

AND, OR, NOT Logical Operators

- *Logical operators* are used in the WHERE clause
- You may use multiple *logical operators* in a WHERE clause to create a *compound condition*. The order of evaluation when multiple operators are used is shown in the table above.

Table 6. Logical Operators

Operator	Description	Order of Evaluation
NOT	(a NOT b) – a must be present but b must NOT be present to be included	1
AND	(a AND b) –If both a and b are present, item is included	2
OR	(a OR b) – If either a OR b is present item is included	3

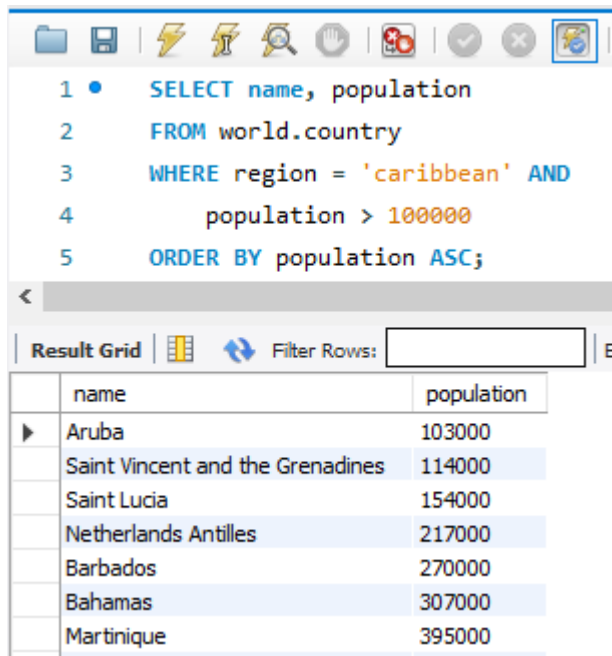
Example:

```

USE world;
SELECT name, population
FROM country
WHERE region = 'caribbean'
AND population > 100000
ORDER BY population ASC;

```

Results:



The screenshot shows a SQL IDE window with a query editor and a results grid. The query is as follows:

```

1 • SELECT name, population
2 FROM world.country
3 WHERE region = 'caribbean' AND
4     population > 100000
5 ORDER BY population ASC;

```

Below the query editor, the 'Result Grid' tab is active, displaying the following data:

	name	population
▶	Aruba	103000
	Saint Vincent and the Grenadines	114000
	Saint Lucia	154000
	Netherlands Antilles	217000
	Barbados	270000
	Bahamas	307000
	Martinique	395000

DISTINCT Keyword

- DISTINCT appears directly after the SELECT clause.
- You can specify multiple columns, which means that the combination of columns must be unique.

Table 7. *DISTINCT* Keyword

Keyword	Description	Order of Evaluation
DISTINCT	Eliminates duplicate rows	1

Example:

```

SELECT DISTINCT continent, name
FROM country
ORDER BY continent;

```

Results:


```

1 • SELECT DISTINCT continent
2   FROM country
3   ORDER BY continent;

```

Result Grid		Filter Rows:
	continent	
▶	Asia	
	Europe	
	North America	
	Africa	
	Oceania	
	Antarctica	
	South America	

The Five Clauses of the SELECT Statement

Column Specifications

LIKE and REGEXP Operators

Arithmetic Operators

Column Aliases

Comparison Operators

IS NULL, BETWEEN, IN Operators

AND, OR, NOT Logical Operators

DISTINCT Clause



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/how_to_retrieve_data.

The Five Clauses of the SELECT Statement

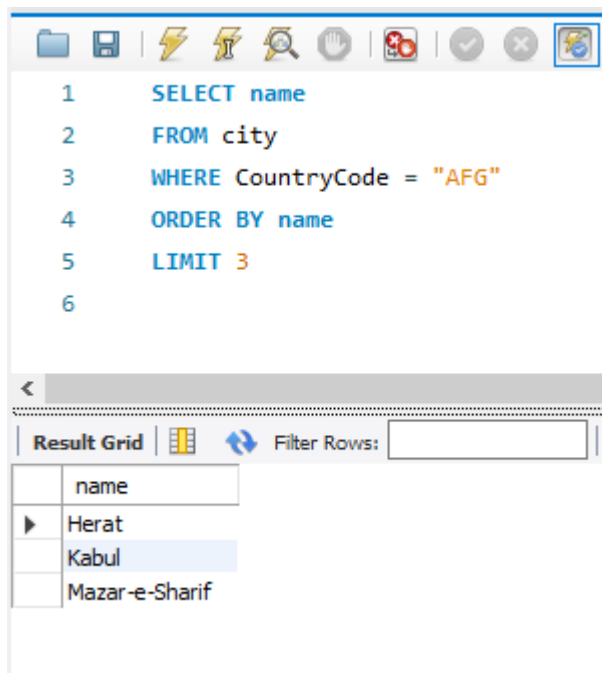
The Five Clauses of the SELECT statement

- SELECT – the columns in the result set
- FROM – names the base table(s) from which results will be retrieved
- WHERE – specifies any conditions for the results set (filter)
- ORDER BY – sets how the result set will be ordered
- LIMIT – sets the number of rows to be returned

The clauses **MUST** appear in the order shown above.

```
Code Example:1  USE world;
2  SELECT name
3  FROM city
4  WHERE CountryCode = "AFG"
5  ORDER BY name
6  LIMIT 3
```

Results:



Let us break the statement line by line:

USE world;

- The **USE** clause sets the database that we will be querying. You typically have more than one database on your database server. You have to specify which database you are working in.
- The semicolon ";" indicates the end of a statement. You can execute multiple statements in sequence by defining each statement with a semicolon

SELECT name

- The **SELECT** clause defines the columns and column order that you want to retrieve in your results set. If you want to retrieve all of the columns from the base table you can simply use `SELECT *`
- You separate each column name with a comma "," ex., `SELECT name, CountryCode`
- There is no trailing comma at the end of a column list

FROM city

- The **FROM** clause specifies the table that the results will be coming from
- You can specify multiple tables by using a `JOIN` clause, but we will address that topic at a future time

ORDER BY name

- The **ORDER BY** clause is not required but when used it defines the sort order of the results
- By default, the sort order is ascending. This is *implicit*. However, you can use *explicit* syntax of `ASC`. If you want the sort, order to be descending you can use the keyword `DESC`.
- You can specify more than one column in an Order By statement separated by commas. The sort order `DESC`, `ASC` applies to each column individually. Below are some examples
 - **ORDER BY** population ASC, name DESC
 - **ORDER BY** population, name (ASC is always implied if not explicitly stated)

LIMIT 5;

- If you only want to return a specified number of rows from the result set, you can use the LIMIT clause. This can be helpful when you want to test a query for accuracy that could potentially bring back a very large number of rows.
- The semicolon; defines the end of the statement.

Table 1. Column Specifications

Source	Option	Syntax
Base Table Value	Show all columns	
Base Table Value	Column Name	Comma-separated list of column names
Calculated Value	Calculation result	Arithmetic expression
Calculated Value	Calculation result	Functions



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_five_clauses_of_.

Column Specifications

Column Specifications

Source	Option	Syntax
Base Table Value	Show all columns	*
Base Table Value	Column Name	Comma separated list of column names
Calculated Value	Calculation result	Arithmetic expression
Calculated Value	Calculation result	Functions



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/12_column_specificat.

1.3

LIKE and REGEXP Operators

LIKE and REGEXP Operators

- The LIKE keyword is used with the WHERE clause.
- The LIKE keyword can use two symbols as wildcards. The percent (%) symbol matches any number of characters and the underscore (_) matches a single character
- REGEXP keyword allows you to do more complex pattern matching than a LIKE keyword/
- Some version of REGEXP exists in many computer languages. Refer to the “LIKE and REGEXP” handout for a full list of examples.

Table 2. LIKE Keyword

LIKE Symbol	Description
%	Match any string of characters to the left of the symbol
_	Match a single character

Code Example:

```
USE world;  
SELECT name  
FROM country  
WHERE name LIKE 'A%'
```

Results:

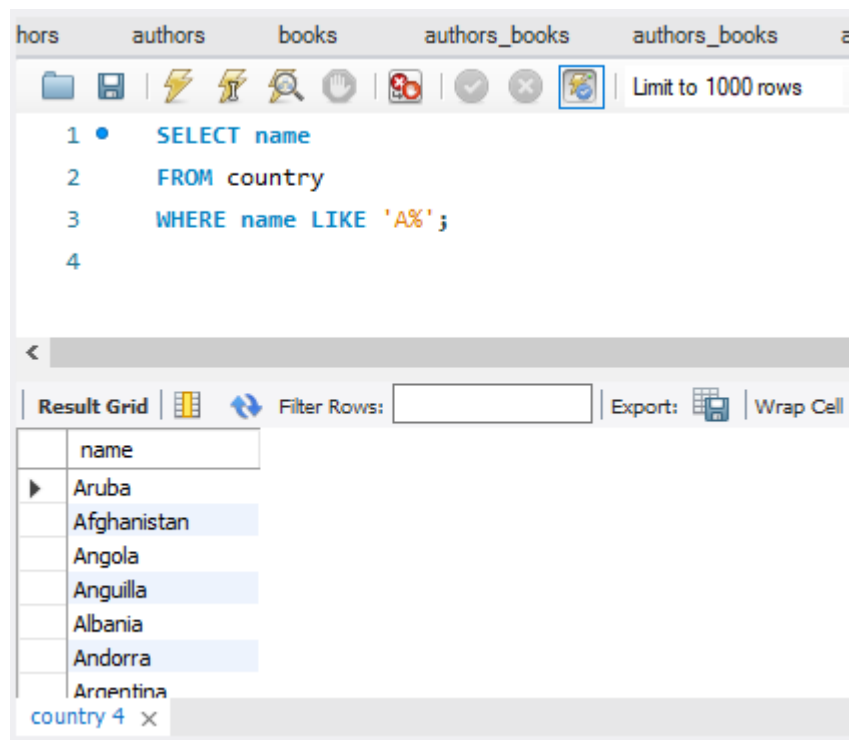


Table 3. REGEXP Keyword

REGEXP Characters	Description
^	Match the pattern to the beginning of the value being tested.
\$	Match the pattern to the end of the value being tested.
.	Matches any single character.
[charlist]	Matches any single character listed within the brackets.
[char1 – char2]	Matches any single character within the given range.
	Separates two string patterns and matches either one

Code Example:

```

USE world;
SELECT name
FROM country
WHERE name REGEXP 'g[o,u]';

```

Results:

hors

authors

books

authors_books

1

2

3

4

5

USE world;

SELECT name

FROM country

WHERE name REGEXP 'g[o,u]';

<

Result Grid

Filter Rows:

	name
▶	Angola
	Anguilla
	Antigua and Barbuda
	Bosnia and Herzegovina
	Congo, The Democratic Republic of the
	Congo
	Guinea
	Guadeloupe
	Guinea-Bissau
	Equatorial Guinea



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/13_like_and_regexp_o.

1.4

Arithmetic Operators

Arithmetic Operators

- Arithmetic operators can be used in the SELECT, WHERE, and ORDER BY clauses.
- Operators are evaluated in the same way as arithmetic in other contexts.

Table 4. Operators and precedence order

Operator	Name	Order of Precedence
*	Multiplication	1
/	Division	1
DIV	Integer Division	1
% (MOD)	Modulo (remainder)	1
+	Addition	2
-	Subtraction	2

Code Example:

```
USE world;  
SELECT name, population / SurfaceArea  
AS "People per square mile"  
FROM country;
```

Results:

Column Aliases

Column Aliases

- A column alias provides a way to create a clean or more descriptive header for a results set.
- A column alias **cannot** be used in a SELECT, WHERE, GROUP BY or HAVING clause due to the order of execution. You must refer to the original column name.

In the previous example, we created a new column that was a *calculated value*. The problem is that the column header is now population / SurfaceArea. However, we can rename the column header to something cleaner by creating a *column alias*. Look at the code snippet below.

Example:

```
SELECT name, population / SurfaceArea  
       AS "People per square mile"  
FROM country;
```

We used the AS keyword then in quotes we put the new column alias of "People per square mile." Which changes the column header as seen show below.

Results:

hous authors books authors_books authors

Limit to

```

1 • SELECT name, population / SurfaceArea
2     AS "People per square mile"
3     FROM world.country;

```

<

Result Grid Filter Rows: Export:

	name	People per square mile
▶	Aruba	533.678756
	Afghanistan	34.841816
	Angola	10.329670
	Anguilla	83.333333
	Albania	118.310839
	Andorra	166.666667
	Netherlands Antilles	271.250000



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/column_aliases.

1.6

Comparison Operators

Comparison Operators

- Comparison operators compare two expressions.
- The result of a comparison results to true or false.
- Comparison operators are not case sensitive and are used with text and dates as well as numbers.

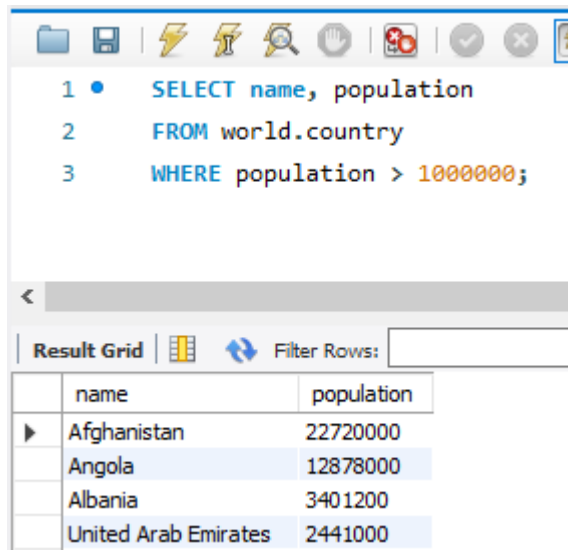
Table 5. Comparison Operators

Operator	Description
=	Equal
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal
!=	Not equal

Example:

```
USE world;  
SELECT name, population  
FROM country  
WHERE population > 1000000;
```

Results:



The screenshot shows a MySQL query editor window. The query is as follows:

```
1 • SELECT name, population
2 FROM world.country
3 WHERE population > 1000000;
```

Below the query, there is a 'Result Grid' section. It includes a 'Filter Rows' input field and a table with the following data:

	name	population
▶	Afghanistan	22720000
	Angola	12878000
	Albania	3401200
	United Arab Emirates	2441000



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/comparison_operators.

IS NULL, BETWEEN, IN Operators

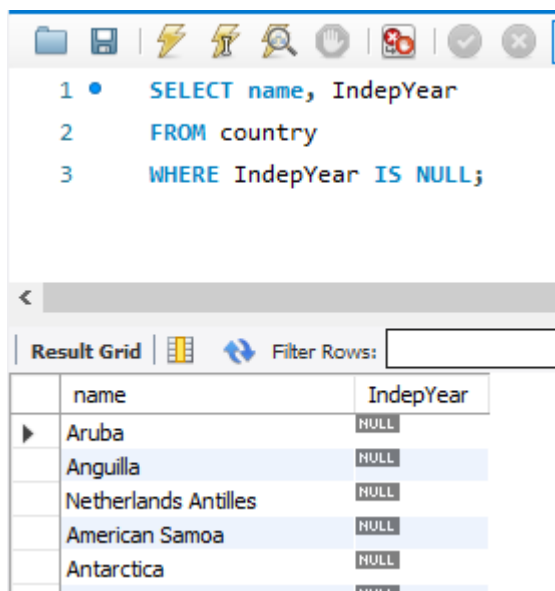
IS NULL

- *Null values* indicate an unknown or non-existent value and is different from an empty string (' ').
- To test for a *null value* you use the IS NULL clause
- The test for a value use IS NOT NULL clause

Example:

```
SELECT name, IndepYear
FROM country
WHERE IndepYear IS NULL;
```

Results:



The screenshot shows a database query tool interface. At the top, there is a toolbar with icons for file operations, execution, and navigation. Below the toolbar, the SQL query is displayed in a text area, numbered 1 to 3. The query is:
1 SELECT name, IndepYear
2 FROM country
3 WHERE IndepYear IS NULL;
Below the query, there is a 'Result Grid' section. It contains a table with two columns: 'name' and 'IndepYear'. The table lists several countries with their independence years, all of which are NULL. The countries listed are Aruba, Anguilla, Netherlands Antilles, American Samoa, and Antarctica. The 'IndepYear' column for each country contains the value 'NULL'.

name	IndepYear
Aruba	NULL
Anguilla	NULL
Netherlands Antilles	NULL
American Samoa	NULL
Antarctica	NULL

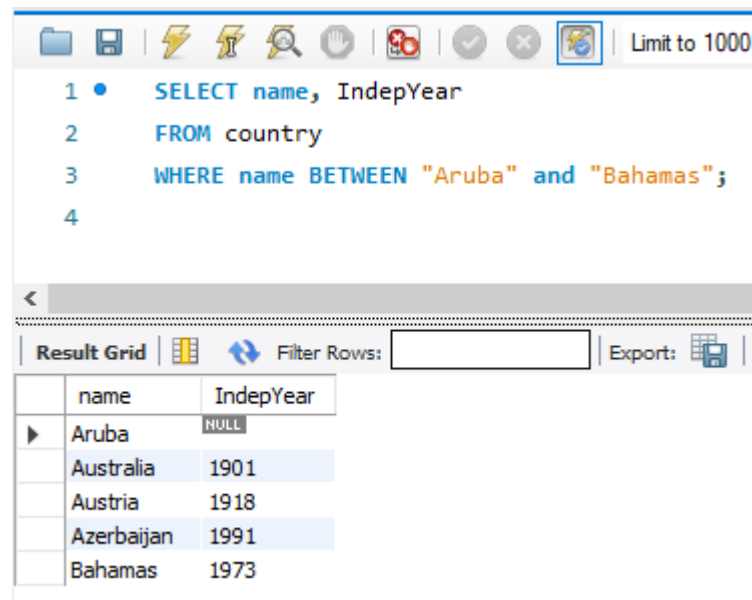
BETWEEN Operators

- The BETWEEN operator is similar to \geq and \leq .
- BETWEEN includes everything between the two values indicated.
- BETWEEN works with both text and number.

Example:

```
USE world;
SELECT name, IndepYear
FROM country
WHERE name BETWEEN "Aruba" and "Bahamas";
```

Results:



The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
1 • SELECT name, IndepYear
2 FROM country
3 WHERE name BETWEEN "Aruba" and "Bahamas";
4
```

Below the query, the results are displayed in a table grid. The table has two columns: 'name' and 'IndepYear'. The results are:

name	IndepYear
Aruba	NULL
Australia	1901
Austria	1918
Azerbaijan	1991
Bahamas	1973

The IN Keyword

- The IN clause tests whether an expression is equal to a value or values in a list of expressions.
- The order of the items in the list does not matter.
- You can use the NOT operator to test for items not in the list.
- The IN clause may be used with a subquery.

Examples:

```
USE world;
SELECT name
FROM country
WHERE name IN ('Aruba', 'Barbados', 'Cuba', 'Bahamas')
ORDER BY population ASC;
```

Results:

```
1 • USE world;
2 • SELECT name
3   FROM country
4   WHERE name IN ('Aruba', 'Barbados', 'Cuba', 'Bahamas')
5   ORDER BY population ASC;
```

< | Result Grid | Filter Rows: | Export: | Wrap Cell Content

	name
▶	Aruba
	Barbados
	Bahamas
	Cuba



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/is_null_between_in_o.

AND, OR, NOT Logical Operators

AND, OR, NOT Logical Operators

- *Logical operators* are used in the WHERE clause
- You may use multiple *logical operators* in a WHERE clause to create a Compound condition. The order of evaluation when multiple operators are used is shown in the table above.

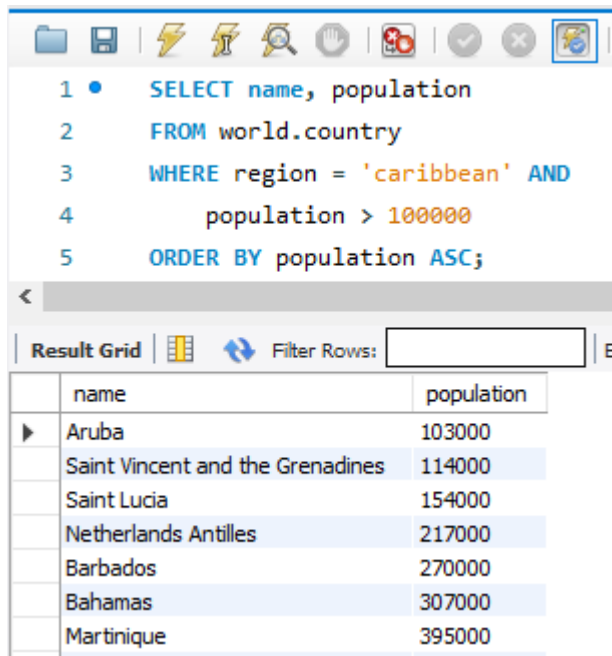
Table 6. Logical Operators

Operator	Description	Order of Evaluation
NOT	(a NOT b) – a must be present but b must NOT be present to be included	1
AND	(a AND b) – If both a and b are present, item is included	2
OR	(a OR b) – If either a OR b is present item is included	3

Example:

```
USE world;
SELECT name, population
FROM country
WHERE region = 'caribbean'
AND population > 100000
ORDER BY population ASC;
```

Results:



The screenshot shows a MySQL query editor window. The query is as follows:

```

1 • SELECT name, population
2 FROM world.country
3 WHERE region = 'caribbean' AND
4     population > 100000
5 ORDER BY population ASC;

```

Below the query, the results are displayed in a table with the following data:

name	population
Aruba	103000
Saint Vincent and the Grenadines	114000
Saint Lucia	154000
Netherlands Antilles	217000
Barbados	270000
Bahamas	307000
Martinique	395000



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/and_or_not_logical_o.

DISTINCT Clause

DISTINCT Keyword

- The DISTINCT clause removes duplicate rows from a query.
- DISTINCT appears directly after the SELECT clause.
- You can specify multiple columns, which means that the combination of columns must be unique.


Table 7. DISTINCT Keyword

Keyword	Description	Order of Evaluation
DISTINCT	Eliminates duplicate rows	1

Example:

```
SELECT DISTINCT continent, name  
FROM country  
ORDER BY continent;
```

Results:





```

1 • SELECT DISTINCT continent
2 FROM country
3 ORDER BY continent;

```

<

Result Grid   Filter Rows:

	continent
▶	Asia
	Europe
	North America
	Africa
	Oceania
	Antarctica
	South America



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/distinct_clause.

How to Retrieve Data from Multiple Tables

The JOIN Clause

Joining More Than Two Tables

The OUTER JOIN Clause

How to Code a UNION



This content is provided to you freely by BYU-I Books.

Access it online or download it at

https://books.byui.edu/learning_mysql/multiple_table_retrieve_data.

The JOIN Clause

The Join Clause

- A JOIN clause allows you to access data from two or more tables in a query.
- A join links to tables on a common key between the two tables. Usually the primary key on one table is compared to the foreign key on another table using the equals (=) sign. This is an equijoin or an inner-join. However, other comparison operators are also valid.
- If column names from each table in the join have the same name, they must be qualified with the table name or a table alias.

Below is a basic example of a SQL statement with an inner join clause using **explicit** syntax.

```
1  USE world;
2  SELECT city.name AS "City Name",
3         country.name AS "Country Name"
4  FROM country
5         JOIN city
6         ON city.CountryCode = country. Code;
```

You could write SQL statements more succinctly with an inner join clause using *table aliases*. Instead of writing out the whole table name to qualify a column, you can use a table alias.

```
1  USE world;
2  SELECT ci.name AS "City Name",
3         co.name AS "Country Name"
4  FROM city ci
5         JOIN country co
6         ON ci.CountryCode = co.Code;
```

The results of the join query would yield the same results as shown below whether or not table names are completely written out or are represented with table aliases. The table aliases of co for country and ci for city are defined in the FROM clause and referenced in the SELECT and ON clause:

Results:

books authors_books authors_books authors country country WorldDB Query 8

Limit to 1000 rows

```

1 • USE world;
2 • SELECT ci.name AS "City Name",
3     co.name AS "Country Name"
4 FROM city ci JOIN country co
5     ON ci.CountryCode = co.Code;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

	City Name	Country Name
▶	Oranjestad	Aruba
	Kabul	Afghanistan
	Qandahar	Afghanistan
	Herat	Afghanistan
	Mazar-e-Sharif	Afghanistan
	Luanda	Angola
	Huambo	Angola
	Lobito	Angola
	Benguela	Angola
	Namibe	Angola

= table aliases
 = column aliases
 = join condition

Let us break the statement line by line:

USE world;

- The **USE** clause sets the database that we will be querying. You typically have more than one database on your database server. You have to specify which database you are working in.
- The semicolon ";" indicates the end of a statement. You can execute multiple statements in sequence by defining each statement with a semicolon

SELECT ci.name AS "City Name", co.name AS "Country Name"

- The **SELECT** clause defines the columns and column order that you want to retrieve in your result set. In this example, we have columns from two separate tables. These columns have the same name, so they **MUST** be qualified with the full table name or table alias. Otherwise, the column names are ambiguous.
- You separate each column name with a comma "," including the corresponding table alias if one is provided
- To create a friendlier column name in the output, we assign a *column alias* to each qualified column name. Instead of ci.name showing in the column header of the report, we assign a friendlier column alias of "City Name" and for co.name "Country Name."

FROM city ci

- The **FROM** clause specifies the table(s) from which results will be returned.
- In a **JOIN** clause, the first table to be joined is specified after the **FROM** clause.

JOIN country co

- Use a **JOIN** clause between the two tables.
- Include the alias if desired.

ON ci.CountryCode = co.Code;

- The **ON** clause specifies the common column from each table (usually a PK in one table and its corresponding foreign key in the other). Each column name is separated with an operator (join condition usually the equals (=) sign).



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_join_clause.

2.2

Joining More Than Two Tables

How to Join More than Two Tables

- To include more tables in the query, you simply add more additional **JOIN** clauses

Code Snippet:

```
1  USE world;
2  SELECT ci.name AS "City Name",
3         co.name AS "Country Name",
4         cl.language AS "Country Language"
5  FROM city ci
6       JOIN country co
7         ON ci.CountryCode = co.Code
8       JOIN country language cl
9         ON cl.CountryCode = ci.CountryCode;
```

Results:

books authors_books authors_books authors country

Limit to 1000 rows

```

1 • USE world;
2 • SELECT ci.name AS "City Name",
3       co.name AS "Country Name",
4       cl.language AS "Country Language"
5 FROM city ci
6      JOIN country co
7          ON ci.CountryCode = co.Code
8      JOIN countrylanguage cl
9          ON cl.CountryCode = ci.CountryCode;

```

Result Grid Filter Rows: Export: Wrap Cell C

	City Name	Country Name	Country Language
▶	Oranjestad	Aruba	Dutch
	Oranjestad	Aruba	English
	Oranjestad	Aruba	Papiamentu
	Oranjestad	Aruba	Spanish
	Kabul	Afghanistan	Balochi
	Kabul	Afghanistan	Dari

JOIN countrylanguage cl.

- The "cl" is the alias for countrylanguage.
- You can refer to tables already specified in a previous join.

ON cl.CountryCode = ci.CountryCode;

- The common column between the two tables being joined is the CountryCode column from the countrylanguage table and the CountryCode column from the city table.
- The "cl" alias previously defined for countrylanguage is used to specify the CountryCode column.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/joining_more_than_tw.

The OUTER JOIN Clause

The Outer Join Clause

- An outer join will return all the rows from one table and only the rows from the other table that match the join condition
- You can use **LEFT JOIN** or **RIGHT JOIN**. If you use **LEFT JOIN**, all the rows from the table on the left of the equals (=) sign will be included in the result set whether the join condition is satisfied or not
- If you use **RIGHT JOIN**, all the rows from the table on the right of the equals (=) sign will be included in the result set whether the join condition is satisfied or not.

Below is a code snippet of a SQL statement with an outer join clause.

```
1 USE world;  
2 SELECT c.name, c.continent, cl.language  
3 FROM country c LEFT JOIN countrylanguage cl  
4 ON c.code = cl.CountryCode  
5 ORDER BY cl.language ASC;
```

Results:

country	country	city	Query 2	country
---------	---------	------	---------	---------

1	•	USE world;
2	•	SELECT c.name, c.continent, cl.language
3		FROM country c LEFT JOIN countrylanguage cl
4		ON c.code = cl.CountryCode
5		ORDER BY cl.language ASC;

Result Grid	Filter Rows:	Export:	Wrap Cell Contents:
-------------	--------------	---------	---------------------

	name	continent	language
▶	Antarctica	Antarctica	NULL
	French Southern territories	Antarctica	NULL
	Bouvet Island	Antarctica	NULL
	Heard Island and McDonald Islands	Antarctica	NULL
	British Indian Ocean Territory	Africa	NULL
	South Georgia and the South Sandwich Islands	Antarctica	NULL
	Georgia	Asia	Abkhazi
	Uganda	Africa	Acholi
	Benin	Africa	Adja
	Djibouti	Africa	Afor

SELECT c.name, c.continent, cl.language

- The “c.” pre-pended to name and continent is a table alias to the country table. Therefore, return name and continent from the country table.
- The “cl” prepended to the language table is a table alias to the countrylanguage table. Therefore, return language from the countryLanguage table.

FROM country c LEFT JOIN countrylanguage cl

- “Country c” assigns “c” as an alias for “country”
- “countrylanguage cl” assigns “cl” as an alias for “countrylanguage”
- LEFT JOIN means that all rows on the left side of the JOIN operator (=) are included in the results whether they have a matching key from the table on the RIGHT side of the operator.

ON c.code = cl.CountryCode

- ON is the second part of the JOIN clause. It precedes the JOIN condition
- c.code refers to the code column from the country table and is a primary key. Since the key is on the LEFT side of the join condition, all rows from the country table will be included in the results whether they have a matching key in the countrylanguage table or not.
- Cl.CountryCode refers to the CountryCode on the countrylanguage table and is a foreign key to the country table. Only the rows that have a matching key in the country table will be included in the results.





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_outer_join_claus.

How to Code a UNION

How to Code a UNION

- A **UNION** combines the results of two or more queries into a single result set
- Each result set must have the same number of columns
- The corresponding data types for each column must be compatible. However, the column names may be different from each result set
- A **UNION** removes duplicate rows by default
- You may interfile the results using an **ORDER BY** clause if there is a column with a common name.

Code Example:

```
1 USE world;  
2 SELECT name, population  
3 FROM city WHERE CountryCode = 'AUS'  
4 UNION  
5 SELECT name, population  
6 FROM country  
7 WHERE continent = 'Oceania'  
8 ORDER BY name;
```

Results:

```

1 • USE world;
2 • SELECT name, population
3   FROM city WHERE CountryCode = 'AUS'
4   UNION
5   SELECT name, population
6   FROM country
7   WHERE continent = 'Oceania'
8   ORDER BY name;
9

```

	name	population	
▶	Adelaide	978100	Country and population from country table
	American Samoa	68000	
	Australia	18886000	City and population from cities table
	Brisbane	1291117	
	Cairns	92273	
	Canberra	322723	
	Central Coast	227657	

SELECT name, population

FROM city

WHERE CountryCode = 'AUS'

- The first query returns the name and population from the city table.
- The filter (**WHERE CLAUSE**) of the query limits the country code to Australia.

UNION

- The '**UNION**' clause will combine this query with the results of the subsequent query.

SELECT name, population

FROM country

WHERE continent = 'Oceania'

- The second query returns the name and population from the country table.
- The filter (**WHERE CLAUSE**) of the query limits the continent code to Oceania.

ORDER BY name;

- It is possible to sort (**ORDER BY CLAUSE**) and interfile the results of both queries because each query shares a column with the same name. Otherwise, the **ORDER BY** clause would generate an error.





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/how_to_code_a_union.

Using Functions

Date Functions
Numeric Functions
String Functions



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/functions.

3.1

Date Functions

Current Date/Time Functions

- There are a number of functions that give the current date and time. The DATE() function is a date formatting function, but I include it in the list because it is often confused with the NOW() function
- CURRENT_DATE, CURRENT_TIME, UTC_DATE, UTC_TIME can be used with the parentheses “()” or not. They accept no parameters

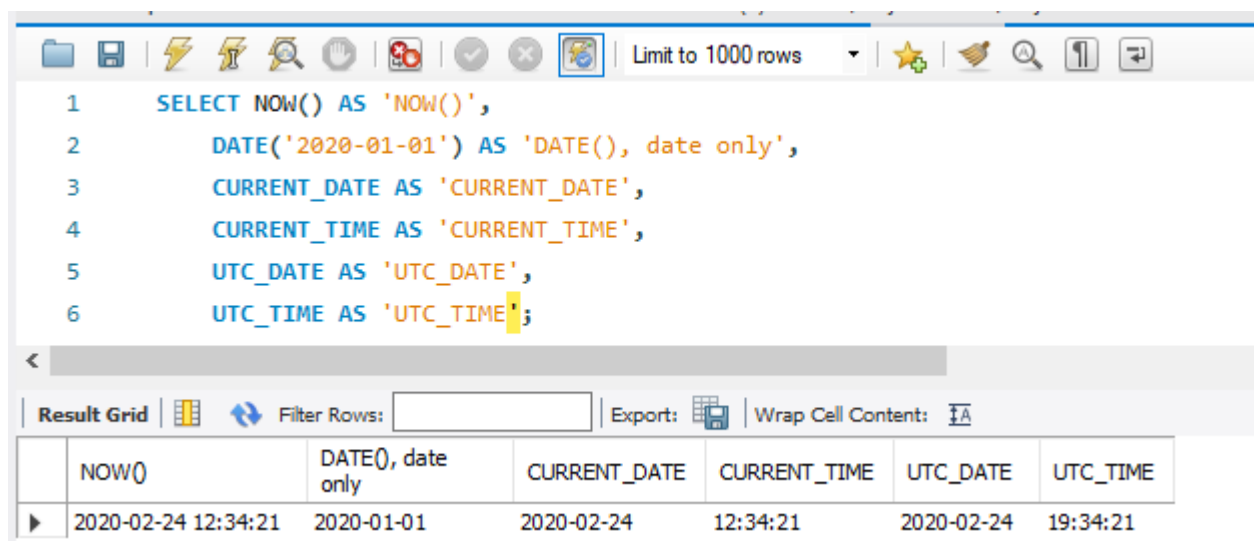
Table 1. Current Date Functions

Function	Type	Example	Result
NOW() * Returns current local date and time.	date/time	NOW()	ex. '2020-02-24 09:31:31'
DATE(date) * extracts the date from input. If time is included, the time is dropped.	date/time	DATE('2020-01-01 11:31:31')	'2020-02-24'
CURRENT_DATE() * Returns current local date	date	CURRENT_DATE	'2020-02-24'
CURRENT_TIME() * Returns current local time.	time	CURRENT_TIME	'11:52:10'
UTC_DATE() * Returns current UTC date.	date	UTC_DATE	'2020-02-24'
UTC_TIME()	time	UTC_TIME	'18:52:10'

Function	Type	Example	Result
* Returns current UTC date.			

```
SELECT NOW() AS 'NOW()',
       DATE('2020-01-01') AS 'DATE()', date only',
       CURRENT_DATE AS 'CURRENT_DATE',
       CURRENT_TIME AS 'CURRENT_TIME',
       UTC_DATE AS 'UTC_DATE',
       UTC_TIME AS 'UTC_TIME';
```

Results:



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The query editor contains the following SQL code:

```
1 SELECT NOW() AS 'NOW()',
2    DATE('2020-01-01') AS 'DATE()', date only',
3    CURRENT_DATE AS 'CURRENT_DATE',
4    CURRENT_TIME AS 'CURRENT_TIME',
5    UTC_DATE AS 'UTC_DATE',
6    UTC_TIME AS 'UTC_TIME';
```

Below the query editor is a toolbar with a 'Result Grid' button, a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are displayed in a table with 7 columns:

	NOW()	DATE(), date only	CURRENT_DATE	CURRENT_TIME	UTC_DATE	UTC_TIME
▶	2020-02-24 12:34:21	2020-01-01	2020-02-24	12:34:21	2020-02-24	19:34:21

DATE_ADD

- Returns a date with a DATE or DATETIME value equal to the original value plus the specified interval.

Table 2. DATE_ADD Function

Function	Type	Example	Result
DATE_ADD(date, interval expression unit)	DATE, DATETIME	DATE_ADD('2020-01-01', INTERVAL 1 DAY)	'202-01-02'

Code Snippet:

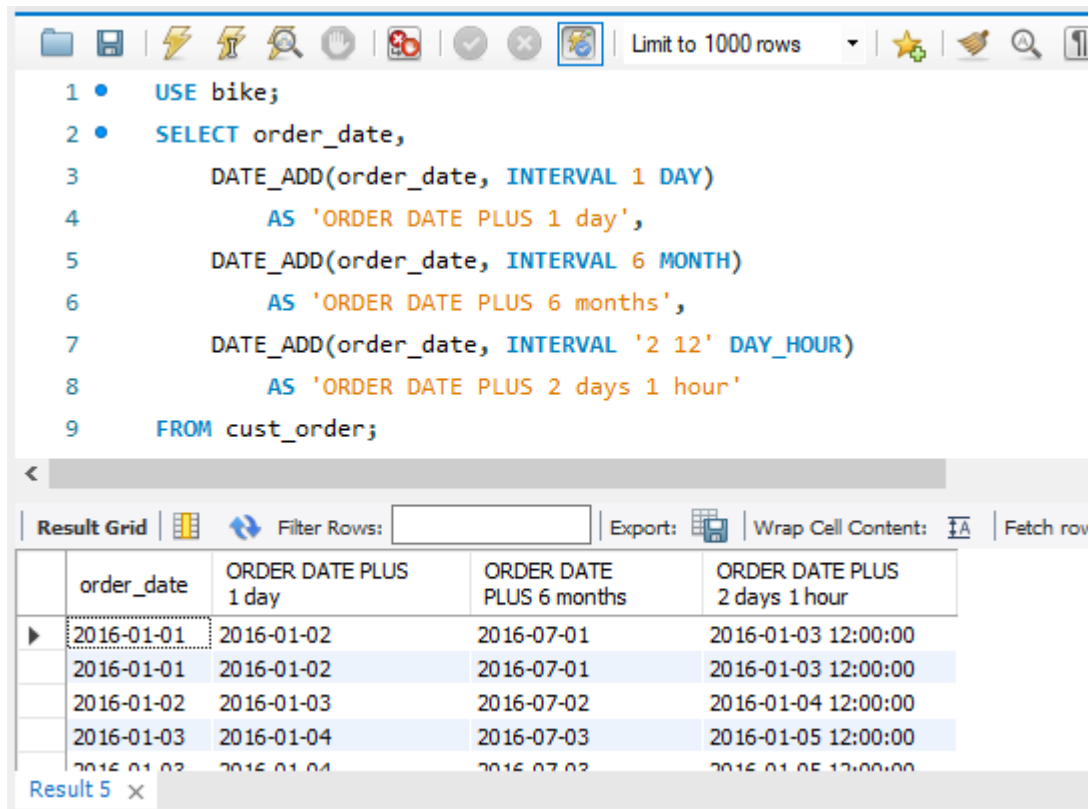
```
USE bike;
SELECT order_date,
       DATE_ADD(order_date, INTERVAL 1 DAY) AS 'ORDER DATE PLUS 1 day',
```

```

DATE_ADD(order_date, INTERVAL 6 MONTH) AS 'ORDER DATE PLUS 6 months',
DATE_ADD(order_date, INTERVAL '2 12' DAY_HOUR)
AS 'ORDER DATE PLUS 2 days 1 hour'
FROM cust_order;

```

Results:



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search, along with a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```

1 • USE bike;
2 • SELECT order_date,
3       DATE_ADD(order_date, INTERVAL 1 DAY)
4       AS 'ORDER DATE PLUS 1 day',
5       DATE_ADD(order_date, INTERVAL 6 MONTH)
6       AS 'ORDER DATE PLUS 6 months',
7       DATE_ADD(order_date, INTERVAL '2 12' DAY_HOUR)
8       AS 'ORDER DATE PLUS 2 days 1 hour'
9 FROM cust_order;

```

Below the editor is the 'Result Grid' tab, which displays the query results in a table. The table has four columns: 'order_date', 'ORDER DATE PLUS 1 day', 'ORDER DATE PLUS 6 months', and 'ORDER DATE PLUS 2 days 1 hour'. The first five rows of data are visible.

order_date	ORDER DATE PLUS 1 day	ORDER DATE PLUS 6 months	ORDER DATE PLUS 2 days 1 hour
2016-01-01	2016-01-02	2016-07-01	2016-01-03 12:00:00
2016-01-01	2016-01-02	2016-07-01	2016-01-03 12:00:00
2016-01-02	2016-01-03	2016-07-02	2016-01-04 12:00:00
2016-01-03	2016-01-04	2016-07-03	2016-01-05 12:00:00
2016-01-03	2016-01-04	2016-07-03	2016-01-05 12:00:00

DATE_FORMAT

- Dates must be enclosed in quotes
- You can pass a DATE or DATETIME datatype to DATE_FORMAT

Table 3. DATE_FORMAT Function

Function	Type	Example	Result
DATE_FORMAT	DATE	DATE_FORMAT('2020-09-03', '%m/%d/%y')	09/03/14

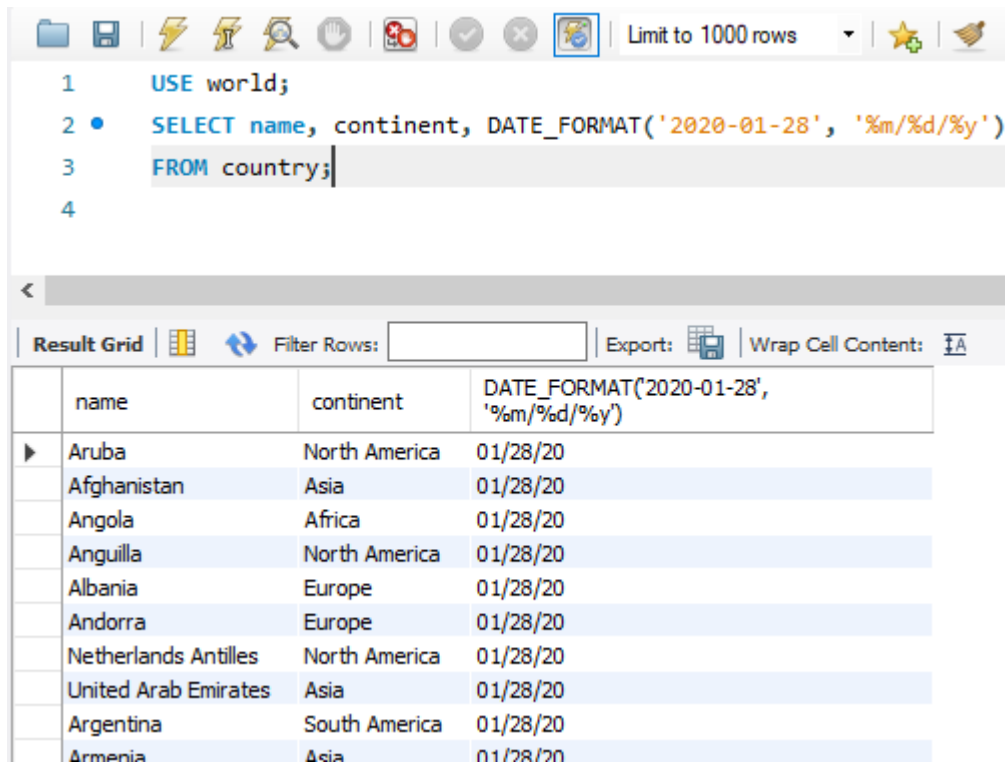
Code Snippet:

```

USE world;
SELECT name, continent, DATE_FORMAT('2020-01-28', '%m/%d/%y')
FROM country;

```

Results:



The screenshot shows a database query editor with a toolbar at the top. The query entered is:

```
1 USE world;
2 SELECT name, continent, DATE_FORMAT('2020-01-28', '%m/%d/%y')
3 FROM country;
4
```

The results are displayed in a grid below the query. The grid has columns for 'name', 'continent', and 'DATE_FORMAT('2020-01-28', '%m/%d/%y')'. The first few rows of data are:

	name	continent	DATE_FORMAT('2020-01-28', '%m/%d/%y')
▶	Aruba	North America	01/28/20
	Afghanistan	Asia	01/28/20
	Angola	Africa	01/28/20
	Anguilla	North America	01/28/20
	Albania	Europe	01/28/20
	Andorra	Europe	01/28/20
	Netherlands Antilles	North America	01/28/20
	United Arab Emirates	Asia	01/28/20
	Argentina	South America	01/28/20
	Armenia	Asia	01/28/20

Table 4. Format List

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)

Specifier	Description
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week; WEEK() , mode 0
%u	Week (00..53), where Monday is the first day of the week; WEEK() , mode 1
%V	Week (01..53), where Sunday is the first day of the week; WEEK() , mode 2; used with %X
%v	Week (01..53), where Monday is the first day of the week; WEEK() , mode 3; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)

Specifier	Description
%%	A literal % character
% <i>x</i>	<i>x</i> , for any “ <i>x</i> ” not listed above

DATEDIFF

- The DATEDIFF function has two parameters. Both are dates.
- The value returned by the function is an integer and is the number of days between the two dates.
- If you provide the latest date, first the results will be positive. Otherwise, it will be negative.

Example:

```
SELECT DATEDIFF('2018-01-01', '2019-01-01')
AS 'Date Difference';
```

Results:

The screenshot shows a SQL query editor interface. At the top, there is a toolbar with various icons and a 'Limit to 1000 rows' dropdown. The query text is as follows:

```
1
2 • SELECT DATEDIFF('2018-01-01', '2019-01-01')
3     AS 'Date Difference';
```

Below the query editor, there is a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The result grid displays the following data:

Date Difference
-365





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/date_functions.

3.2

Numeric Functions

ROUND

- The ROUND function has two parameters. The first is a number, usually a DECIMAL or a FLOAT. The second defines the number of decimals to which the number will be rounded.
- If no length is provided, the number is rounded to a whole number.

Table 5. ROUND function

Function	Type	Example	Result
ROUND(number[, length])	Number	ROUND(13.37, 1)	13.4

Example:

```
USE world;  
SELECT name, LifeExpectancy, ROUND(LifeExpectancy)  
FROM world.country;
```

Results:

```

1 • SELECT name, LifeExpectancy, ROUND(LifeExpectancy)
2 FROM world.country;

```

	name	LifeExpectancy	ROUND(LifeExpectancy)
▶	Aruba	78.4	78
	Afghanistan	45.9	46
	Angola	38.3	38
	Anguilla	76.1	76
	Albania	71.6	72
	Andorra	83.5	84
	Netherlands Antilles	74.7	75
	United Arab Emirates	74.1	74

FLOOR, CEILING, TRUNCATE

- FLOOR() will return the next lowest whole number no matter what the decimal point.
- CEILING() will return the next highest whole number no matter what the decimal point.
- TRUNCATE() will return the number truncated to the precision specified.

Table 6. FLOOR, CEILING, TRUNCATE functions

Function	Type	Example	Result
FLOOR(number)	number	FLOOR(7.7)	7
CEILING(number)	number	CEILING(6.2)	7
TRUNCATE(NUMBER, length)	number	TRUNCATE(7.9)	7

Example:

```

USE bike;
SELECT list_price, FLOOR(list_price), CEILING(list_price),
       TRUNCATE(list_price, 0)
FROM product;

```

Results:

Limit to 1000 rows

```

1 • USE bike;
2 • SELECT list_price, FLOOR(list_price), CEILING(list_price),
3       TRUNCATE(list_price, 0)
4 FROM product;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [↕](#)

	list_price	FLOOR(list_price)	CEILING(list_price)	TRUNCATE(list_price, 0)
▶	379.99	379	380	379
	749.99	749	750	749
	999.99	999	1000	999
	2899.99	2899	2900	2899
	1320.99	1320	1321	1320



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/numeric_functions.

3.3

String Functions

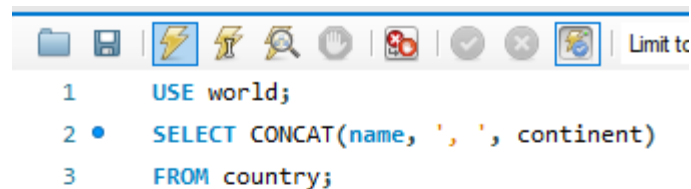
CONCAT

- Combines a list of strings into a single string.
- Can include column values and literal values.
- In MySQL literal values can be enclosed with either single (') or double quotes (") .

Example:

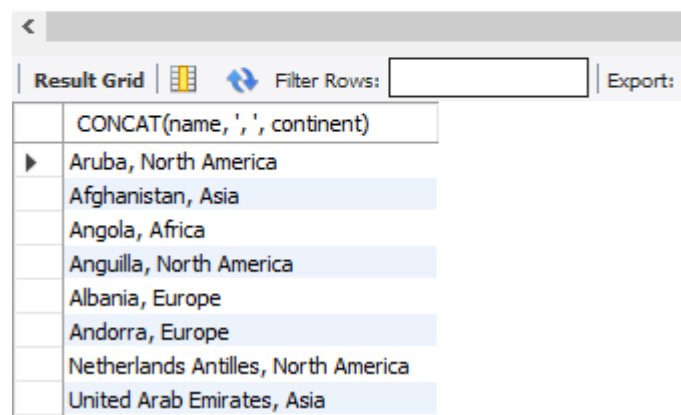
```
USE world;  
SELECT CONCAT(name, ', ', continent)  
FROM country;
```

Results:



The screenshot shows a toolbar with icons for file operations, execution, and navigation. Below the toolbar, the following SQL query is entered:

```
1 USE world;  
2 SELECT CONCAT(name, ', ', continent)  
3 FROM country;
```



The screenshot shows a result grid with a toolbar at the top containing a back arrow, a table icon, a refresh icon, a filter input field, and an export button. The table has one column with the header 'CONCAT(name, ', ', continent)' and several rows of data.

	CONCAT(name, ', ', continent)
▶	Aruba, North America
	Afghanistan, Asia
	Angola, Africa
	Anguilla, North America
	Albania, Europe
	Andorra, Europe
	Netherlands Antilles, North America
	United Arab Emirates, Asia

RIGHT, LEFT

- The RIGHT and LEFT functions have two parameters. The first is a string and the second is the number of characters to be returned.
- The RIGHT function starts counting from the right side of the string. • The LEFT function starts counting from the left side of the string.

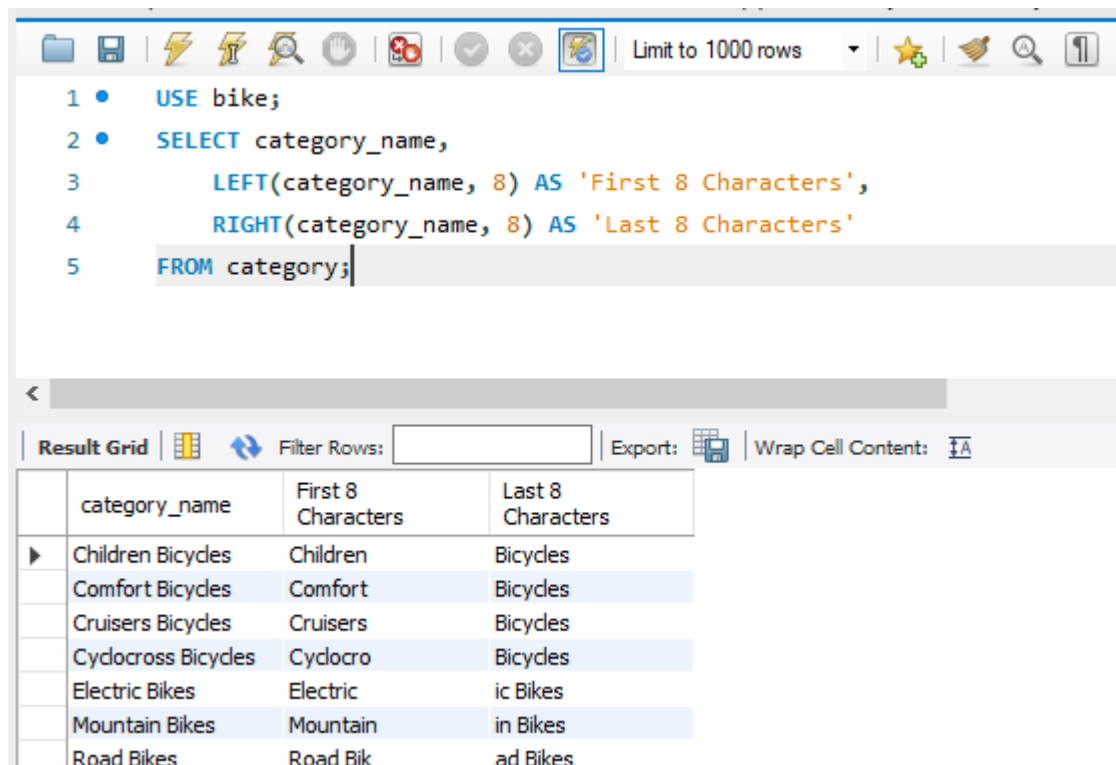
Table 7. RIGHT, LEFT functions

Function	Type	Example	Result
RIGHT(string, num. characters)	string	RIGHT('Salmon', 3)	mon
LEFT(string, num. characters)	string	LEFT('Salmon', 3)	Sal

Example:

```
USE bike;
SELECT category_name,
       LEFT(category_name, 8) AS 'First 8 Characters',
       RIGHT(category_name, 8) AS 'Last 8 Characters'
FROM category;
```

Results:



The screenshot shows a database query editor with a toolbar at the top. The SQL query is entered in the main area. Below the query, the results are displayed in a grid format. The grid has four columns: category_name, First 8 Characters, and Last 8 Characters. The data rows show various bicycle categories and their corresponding first and last 8 characters.

category_name	First 8 Characters	Last 8 Characters
Children Bicycles	Children	Bicycles
Comfort Bicycles	Comfort	Bicycles
Cruisers Bicycles	Cruisers	Bicycles
Cyclocross Bicycles	Cyclocro	Bicycles
Electric Bikes	Electric	ic Bikes
Mountain Bikes	Mountain	in Bikes
Road Bikes	Road Bik	ad Bikes

TRIM, LTRIM, RTRIM

- The TRIM function will remove leading and trailing spaces from a string.
- The LTRIM function will remove leading spaces from a string.
- The RTRIM function will remove trailing spaces from a string.

Table 8. TRIM functions

Function	Type	Example	Result
TRIM(string)	string	TRIM(' Salmon ')	'salmon'
LTRIM(string)	string	LEFT('Salmon ')	'salmon '
RTRIM(string)	string	RIGHT(' Salmon')	' salmon'

Example:

```
SELECT LTRIM(' Salmon ') AS "Left Trim",
       RTRIM(' Salmon ') AS "Right Trim",
       TRIM(' Salmon ') AS "Trim";
```

Results:

The screenshot shows a database query editor with the following SQL query:

```
1 • SELECT LTRIM(' Salmon ') AS "Left Trim",
2       RTRIM(' Salmon ') AS "Right Trim",
3       TRIM(' Salmon ') AS "Trim";
```

Below the query, there are three annotations in boxes:

- Spaces to the right remain**: Points to the LTRIM function.
- Spaces to the left remain**: Points to the RTRIM function.
- All spaces removed**: Points to the TRIM function.

The results grid shows the following data:

Left Trim	Right Trim	Trim
salmon	salmon	salmon

FORMAT

- FORMAT() accepts a decimal but returns a comma formatted string.

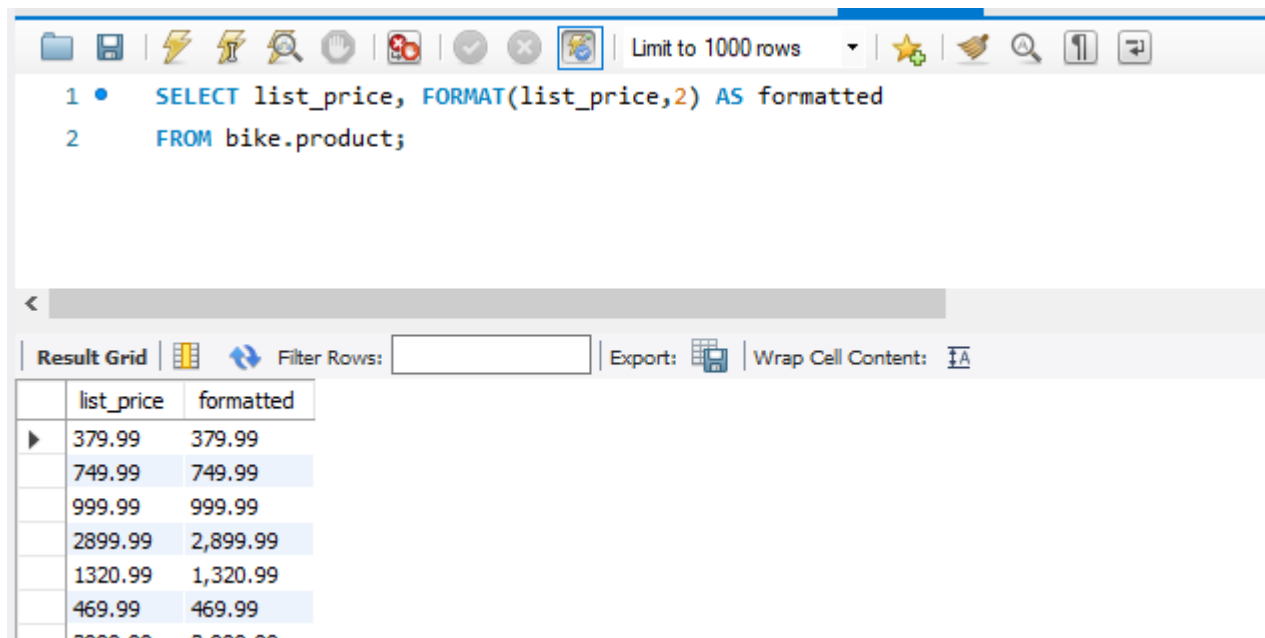
Table 9. FORMAT functions

Function	Type	Example	Result
FORMAT(number, decimal)	string	FORMAT(1234.342, 2)	-356

Code Sample:

```
SELECT FORMAT(list_price,2)
FROM bike.product;
```

Results:



list_price	formatted
379.99	379.99
749.99	749.99
999.99	999.99
2899.99	2,899.99
1320.99	1,320.99
469.99	469.99
2000.00	2,000.00

LOWER, UPPER

- LOWER() converts all characters to lower case.
- UPPER() converts all characters to upper case.

Table 9. LOWER, UPPER functions

Function	Type	Example	Result
LOWER(string)	string	LOWER('Salmon')	'salmon'
UPPER(string)	string	UPPER('Salmon')	'SALMON'

Example:

```
SELECT UPPER('Salmon'),
       LOWER('Salmon');
```

Results:

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The query editor contains the following SQL code:

```
1 • SELECT UPPER('Salmon'),
2     LOWER('Salmon');
3
4
5
```

Below the editor, the 'Result Grid' tab is active, displaying the results of the query in a table:

	UPPER('Salmon')	LOWER('Salmon')
▶	SALMON	salmon

LOCATE, LENGTH, SUBSTRING

LOCATE(), and LENGTH() accept a string but return an integer. • SUBSTRING() accepts a string and returns a string.

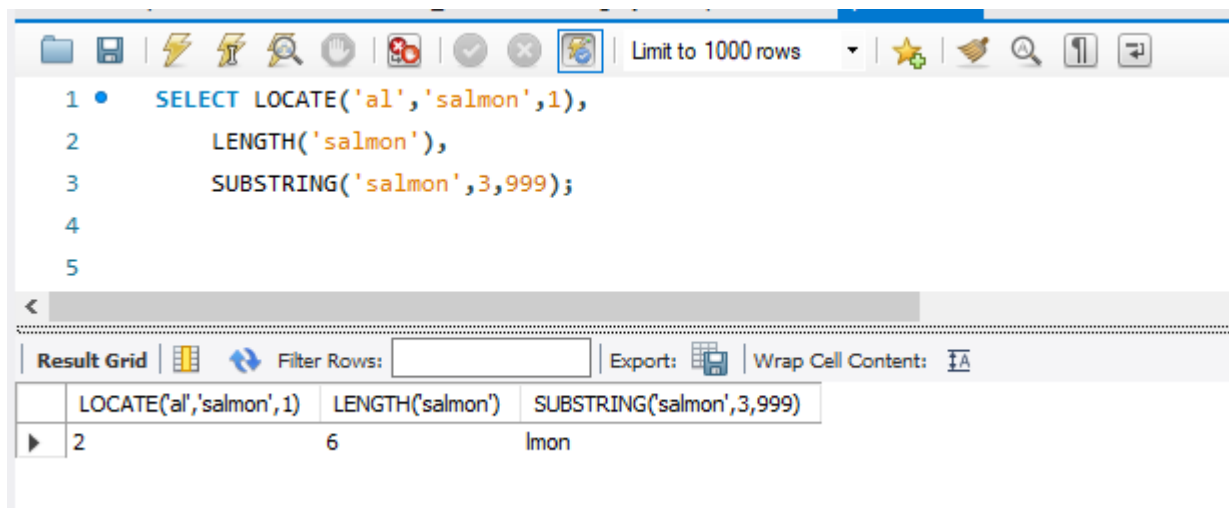
Table 9. LOCATE. LENGTH, SUBSTRING functions

Function	Type	Example	Result
LOCATE(find,search[,start])	string	LOCATE('al','salmon',1)	2
LENGTH(str)	string	LENGTH('salmon')	6
SUBSTRING(str,start[,length])	string	SUBSTRING('salmon',3,999)	'lmon'

Example:

```
SELECT LOCATE('al','salmon',1),
       LENGTH('salmon'),
       SUBSTRING('salmon',3,999);
```

Results:



The screenshot shows a MySQL query editor window. The top toolbar includes icons for file operations, execution, and navigation. The query text is as follows:

```
1 • SELECT LOCATE('a1','salmon',1),  
2     LENGTH('salmon'),  
3     SUBSTRING('salmon',3,999);  
4  
5
```

Below the query editor, the 'Result Grid' is displayed. It has a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are shown in a table with three columns: 'LOCATE('a1','salmon',1)', 'LENGTH('salmon')', and 'SUBSTRING('salmon',3,999)'. The first row of data shows the values 2, 6, and lmon.

	LOCATE('a1','salmon',1)	LENGTH('salmon')	SUBSTRING('salmon',3,999)
▶ 2	2	6	lmon



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/string_functions.

How to Insert, Update, Delete Data in Tables

The INSERT Clause With a Column List

The INSERT Clause Without a Column List

The UPDATE Clause With a Column List

The DELETE Clause



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/how_to_insert_update.

The INSERT Clause With a Column List

The INSERT Clause With a Column List

- You can INSERT single or multiple rows at a time.
- An INSERT with a column list DOES NOT require you to provide a value for each column. If you do not want to provide a value for a specific column, you do not have to include it in the column list. For columns that allow null values, the system will automatically provide a null value for you.
- If you want a column that provides a default value such as an auto-increment column to be populated with the default value, you do not need to list the column in the column list. The system will automatically provide the default value.
- When coding with a column list, the columns may appear in any order as long as the VALUES list matches the order of the column list.

Below is a basic example of an INSERT statement with a column list:

```
1  USE world;
2  INSERT INTO city
3      (name, countryCode, district, population)
4  VALUES
5      ("San Felipe", "CHL", "Valparaiso", 64126);
```

Results:

The screenshot shows a SQL IDE with a query editor and an output window. The query is:

```

1 • USE world;
2 • INSERT INTO city
3   (name, countrycode, district, population)
4   VALUES
5   ("San Felipe", "CHL", "Valparaiso", 64126);
6

```

Annotations in the image:

- A box labeled "Column List" points to the column list in parentheses on line 3.
- A box labeled "No quotes because this is a number" points to the value 64126 on line 5.
- A box labeled "Quotes required because these values are strings." points to the string values "San Felipe", "CHL", and "Valparaiso" on line 5.

The Output window shows the execution results:

#	Time	Action
1	09:53:09	USE world
2	09:53:09	INSERT INTO city (name, countrycode, district, population) VALUES ("San Felipe", "CHL", "Valparaiso", 64126)

Results of the Insert:

The screenshot shows a SQL IDE with a query editor and a result grid. The query is:

```

1 • SELECT *
2   FROM CITY
3   WHERE name = 'san felipe'
4         AND countrycode = 'chl';
5

```

An annotation points to the WHERE clause with the text: "An auto-increment value was automatically provided."

The Result Grid shows the following data:

ID	Name	CountryCode	District	Population
4082	San Felipe	CHL	Valparaiso	64126
	NULL	NULL	NULL	NULL

INSERT INTO city

- Insert the value into the city table. The INTO keyword is not required.

(name, countryCode, district, population)

- The column list is comma-separated and enclosed in parentheses.

VALUES

- The VALUES keyword is between the column list and the actual values. No commas are necessary.

("San Felipe", "CHL", "Valparaiso", 64126);

- The values order must appear in the corresponding order of the column list.
- You must enclose strings in quotes.
- You must not enclose numbers in quotes.
- You do not have to specify columns that allow null values or default values in the column list. They will automatically get a null or default value.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_insert_clause_wi.

4.2

The INSERT Clause Without a Column List

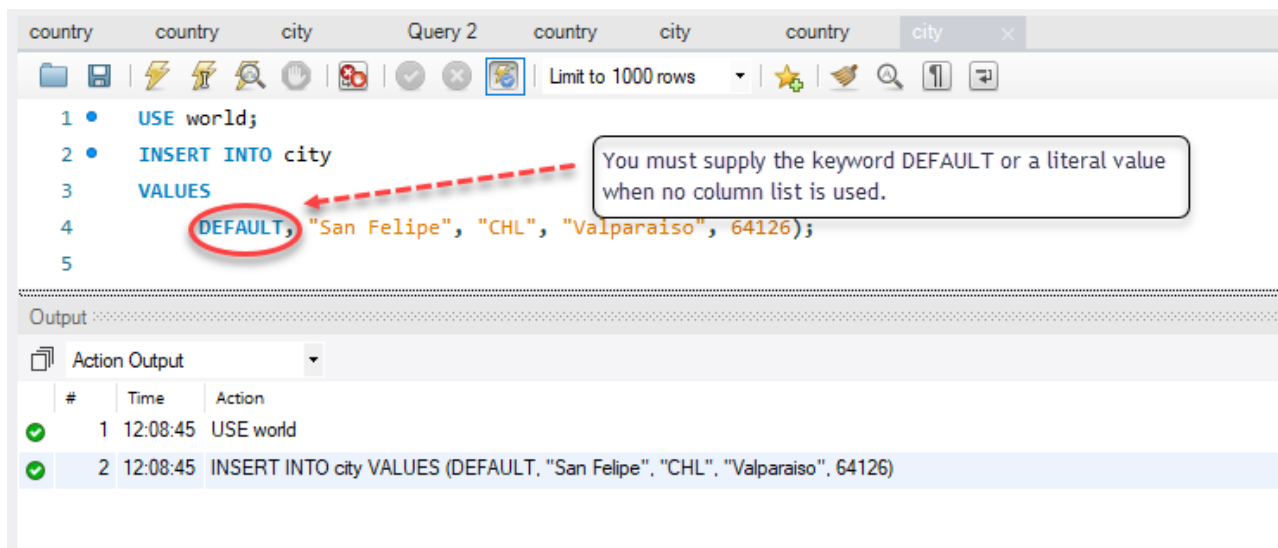
The INSERT Clause Without a Column List

- You can INSERT single or multiple rows at a time.
- An INSERT without a column list requires you to provide a value for every column.
- You must list values in the same order that they appear on the table.
- You must explicitly use the keyword "null" for columns that allow for nulls if you do not want to provide a value.
- You must explicitly use the keyword "DEFAULT" for columns that provide a default value if you do not want to provide one.

Code Sample:

```
1  USE world;
2  INSERT INTO city
3  VALUES
4      (DEFAULT, "San Felipe", "CHL", "Valparaiso", 64126);
```

Results:



The screenshot shows a database query editor with a toolbar and a query window. The query is as follows:

```
1 • USE world;
2 • INSERT INTO city
3  VALUES
4      (DEFAULT, "San Felipe", "CHL", "Valparaiso", 64126);
5
```

A red dashed arrow points from a callout box to the word "DEFAULT" in the query. The callout box contains the text: "You must supply the keyword DEFAULT or a literal value when no column list is used."

Below the query editor is an "Output" section. It has a dropdown menu set to "Action Output". The output is a table with the following data:

#	Time	Action
✓ 1	12:08:45	USE world
✓ 2	12:08:45	INSERT INTO city VALUES (DEFAULT, "San Felipe", "CHL", "Valparaiso", 64126)

(DEFAULT "San Felipe", "CHL", "Valparaiso", 64126);

- The values order must appear in the same order they exist in the table.
- You must enclose strings in quotes.
- You must NOT enclose numbers in quotes.
- You must specify all column names and provide the keyword "DEFAULT" or a literal value for columns that provide a default option.
- If you do not want to provide a value for columns that allow null values, you must provide the keyword "null".



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_insert_clause_wiu.

4.4

The UPDATE Clause With a Column List

The UPDATE Clause

- You can UPDATE single or multiple rows at a time.
- In a SET clause, you define the column along with its new value that may be a literal value or an expression.
- You can update one or all of the columns in a row.
- You can use a subquery or WHERE clause in an UPDATE statement.

Code Sample:

```
1  USE world;
2  UPDATE city
3  SET Population = 65000, district = 'Aconcagua';
```

Results:

The screenshot shows a database query editor with a toolbar at the top. The query text is as follows:

```
1 • USE world;
2 • UPDATE city
3   SET Population = 65000,
4     district = 'Aconcagua';
```

Annotations on the right side of the query editor:

- Numbers do not require quotes (pointing to 65000)
- Strings (VARCHAR) require quotes (pointing to 'Aconcagua')

Below the query editor is an "Output" section with a dropdown menu set to "Action Output". It displays a table with the following data:

#	Time	Action
1	13:48:08	USE world
2	13:48:08	UPDATE city SET Population = 65000, district = 'Aconcagua'

UPDATE city

- You indicate the table you want to UPDATE.

SET Population = 65000, district = 'Aconcagua';

- You indicate the table columns and associated values you want to change them to by using the equals sign (=).
- You must separate each column and value with a comma.
- There is no trailing comma



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_update_clause_wiv.

4.4

The DELETE Clause

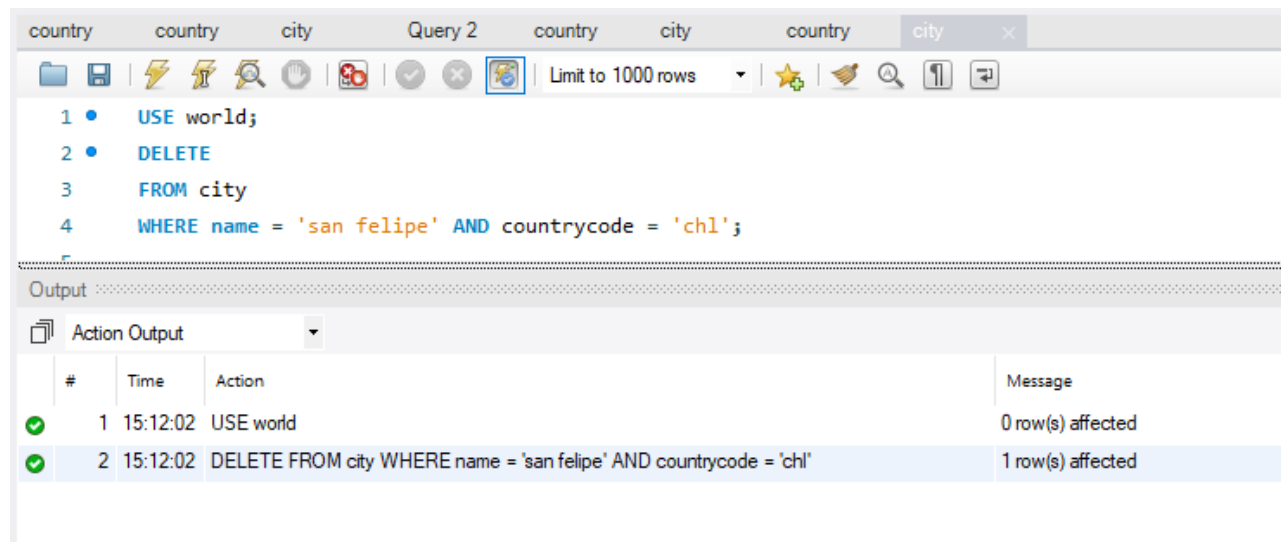
The DELETE Clause

- You can delete single or multiple columns with a single statement.
- You can use a subquery or a WHERE clause with a DELETE statement.
- By default MySQL is in safe update mode which prevents coding a delete statement without a WHERE clause.

Code Example:

```
1  USE world;
2  DELETE
3  FROM city
4  WHERE name = 'san felipe' AND countrycode = 'chl';
```

Results:



The screenshot shows a MySQL query editor with the following SQL code:

```
1 • USE world;
2 • DELETE
3 FROM city
4 WHERE name = 'san felipe' AND countrycode = 'chl';
```

Below the query editor, the 'Output' tab is selected, showing the 'Action Output' table:

#	Time	Action	Message
✓ 1	15:12:02	USE world	0 row(s) affected
✓ 2	15:12:02	DELETE FROM city WHERE name = 'san felipe' AND countrycode = 'chl'	1 row(s) affected

DELETE

- You begin a delete statement with the DELETE clause.

FROM city

- You must specify the table from which you are deleting rows.

WHERE name = 'san felipe' AND countrycode = 'chl';

- You should use a WHERE clause with a DELETE statement to avoid deleting every row in a table.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_delete_clause.

Summary Queries and Aggregate Functions

Aggregate Functions
Grouping Data
Simple GROUP BY Query
Improving the GROUP BY Query
Using the HAVING Clause
Using the HAVING and WHERE Clauses Together
COUNT(column_name) and COUNT(*)
Using the DISTINCT Statement



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/working_with_summary.

5.1

Aggregate Functions

Aggregate Functions

- Aggregate functions are synonymous with column functions.
- A summary query uses at least one column function.
- AVG, SUM return numeric values.
- MIN, MAX, COUNT can return numeric, date, or string values
- All values are included in aggregate functions by default unless you specify the DISTINCT keyword
- Duplicate rows are excluded in all aggregate functions with the exception of COUNT(*)
- ***** IF YOU CODE AN AGGREGATE FUNCTION IN THE SELECT STATEMENT, YOU CANNOT ALSO INCLUDE NON-AGGREGATE FUNCTIONS IN THE SELECT STATEMENT UNLESS THOSE NON-AGGREGATE COLUMNS ARE INCLUDED IN A GROUP BY CLAUSE

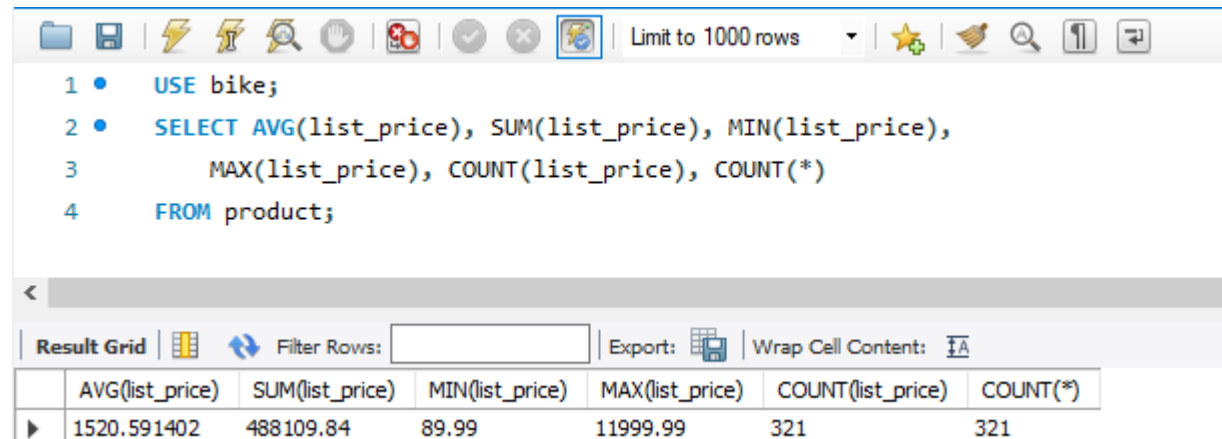
Table 1. Aggregate Functions List

Aggregate Function	Output data-type	Result
AVG([DISTINCT] <i>column_values</i>)	numeric	The average of the non-null columns in the expression
SUM([DISTINCT] <i>column_values</i>)	numeric	The total of the non-null columns in the expression
MIN([DISTINCT] <i>column_values</i>)	numeric, date, string	The lowest value of the non-null columns in the expression
MAX([DISTINCT] <i>column_values</i>)	numeric, date, string	The highest value of the non-null columns in the expression
COUNT([DISTINCT] <i>column_values</i>)	numeric	The number of the non-null columns in the expression
COUNT(*)	numeric	The number of rows returned by the query

Code Sample:

```
USE bike;
SELECT AVG(list_price), SUM(list_price), MIN(list_price),
       MAX(list_price), COUNT(list_price), COUNT(*)
FROM product;
```

Output:



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The query editor displays the following SQL code:

```
1 • USE bike;
2 • SELECT AVG(list_price), SUM(list_price), MIN(list_price),
3       MAX(list_price), COUNT(list_price), COUNT(*)
4 FROM product;
```

Below the query editor, the 'Result Grid' is visible. It has a toolbar with 'Filter Rows', 'Export', and 'Wrap Cell Content' options. The results are displayed in a table with 7 columns and 1 row of data.

	AVG(list_price)	SUM(list_price)	MIN(list_price)	MAX(list_price)	COUNT(list_price)	COUNT(*)
▶	1520.591402	488109.84	89.99	11999.99	321	321



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/aggregate_functions.

Grouping Data

Using the GROUP BY Clause

- Group rows based on a column(s) or expression(s).
- If you use an aggregate function with a GROUP BY clause, the aggregation is calculated for each group.

Table 1. GROUP BY Function

Aggregate Function	Order of Execution	Description
GROUP BY	3	Groups rows of a result set based on columns or expressions separated by commas.

Filtering With WHERE And HAVING

- Notice the order of execution. GROUP BY happens before WHERE but after HAVING.
- It is possible to use WHERE and HAVING in the same statement. They are not mutually exclusive.





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/grouping_data.

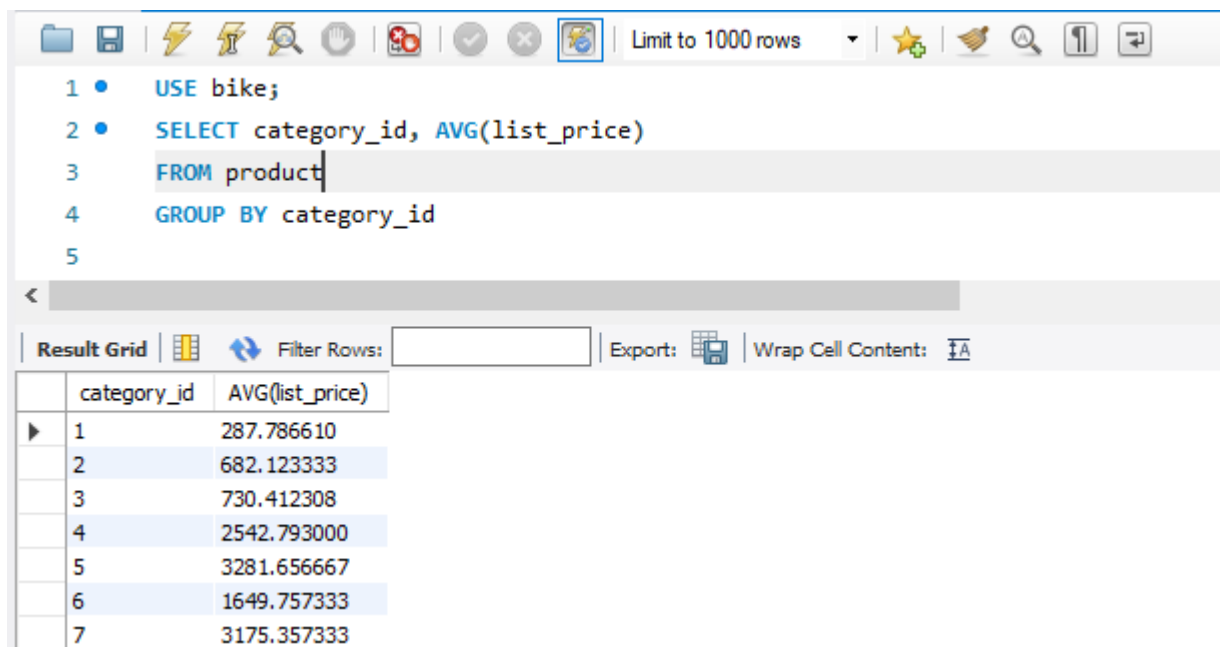
5.3

Simple GROUP BY Query

Code Example:

```
USE bike;
SELECT category_id, AVG(list_price)
FROM product
GROUP BY category_id
```

Results:



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and settings. The query editor contains the following SQL code:

```
1 • USE bike;
2 • SELECT category_id, AVG(list_price)
3   FROM product
4   GROUP BY category_id
5
```

Below the query editor is a 'Result Grid' tab. It shows a table with two columns: 'category_id' and 'AVG(list_price)'. The table contains 7 rows of data. The 'Filter Rows' field is empty, and the 'Export' button is visible. The 'Wrap Cell Content' option is also present.

category_id	AVG(list_price)
1	287.786610
2	682.123333
3	730.412308
4	2542.793000
5	3281.656667
6	1649.757333
7	3175.357333

USE bike:

- Set the bike database to be the default

SELECT category_id, AVG(list_price):

- Select the category_id from the base table
- Calculate the Average of the list price for all rows in the table

FROM product:

- Product is the base table from which data will be returned

GROUP BY category_id:

- Instead of returning a single value that is the average of all list_price items in the product table, return an average list_price for each category
- Without the **GROUP BY** clause, we see from our first example only a single row is returned with an average list_price of 1520.591402.
- With the **GROUP BY** clause, we return an average for each category_id.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/simple_group_by_quer.

5.4

Improving the GROUP BY Query

Improving the GROUP BY Query

- The report would be nicer if we showed the category name instead of the category_id. This will require joining the product table to the category table.
- We can **ROUND** the **AVG** list price by category to TWO decimals points.
- We can **CONCAT** the dollar sign to the left of the list_price.

Code Sample:

```
USE bike;
SELECT category_name,
       CONCAT('$', ROUND(AVG(list_price),2)) AS 'Average List Price'
FROM product p
     JOIN category c
     ON p.category_id = c.category_id
GROUP BY category_name
ORDER BY category_name;
```

Output:

The screenshot shows a SQL IDE interface. At the top, there's a toolbar with various icons and a 'Limit to 1000 rows' dropdown. Below the toolbar, a SQL query is entered in a text area:

```

1 • USE bike;
2 • SELECT category_name,
3       CONCAT('$', ROUND(AVG(list_price),2)) AS 'Average List Price'
4 FROM product p
5     JOIN category c
6     ON p.category_id = c.category_id
7 GROUP BY category_name
8 ORDER BY category_name;

```

Below the query editor, there's a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are displayed in a table:

	category_name	Average List Price
▶	Children Bicycles	\$287.79
	Comfort Bicycles	\$682.12
	Cruisers Bicycles	\$730.41
	Cyclocross Bicycles	\$2542.79
	Electric Bikes	\$3281.66
	Mountain Bikes	\$1649.76
	Road Bikes	\$3175.36

USE bike:

- Set the bike database to be the default

SELECT category_name,

CONCAT('\$', ROUND(AVG(list_price),2)) AS 'Average List Price'

- Return the category_name from the category table.
- You do not have to qualify the column name with the table name because category_name only exists in one table of the join.
- Return the list price with the '\$' followed by the list_price rounded to the 2nd decimal and assigned a column alias of 'Average List Price'.
- You do not have to qualify the column name of list_price because it exists in only one table of the join.

FROM product p

JOIN category c

ON p.category_id = c.category_id

- JOIN the product table to the category table
- Assign a table alias of "p" to product and "c" to category
- The join condition is the primary key of category_id from the category table equal to the foreign key of category_id in the product table.

GROUP BY category_name

- Instead of retrieving a single value with the average price of all products, return a list of average prices by category name.

ORDER BY category_name;

- Sort the results by category_name



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/improving_the_group_.

5.5

Using the HAVING Clause

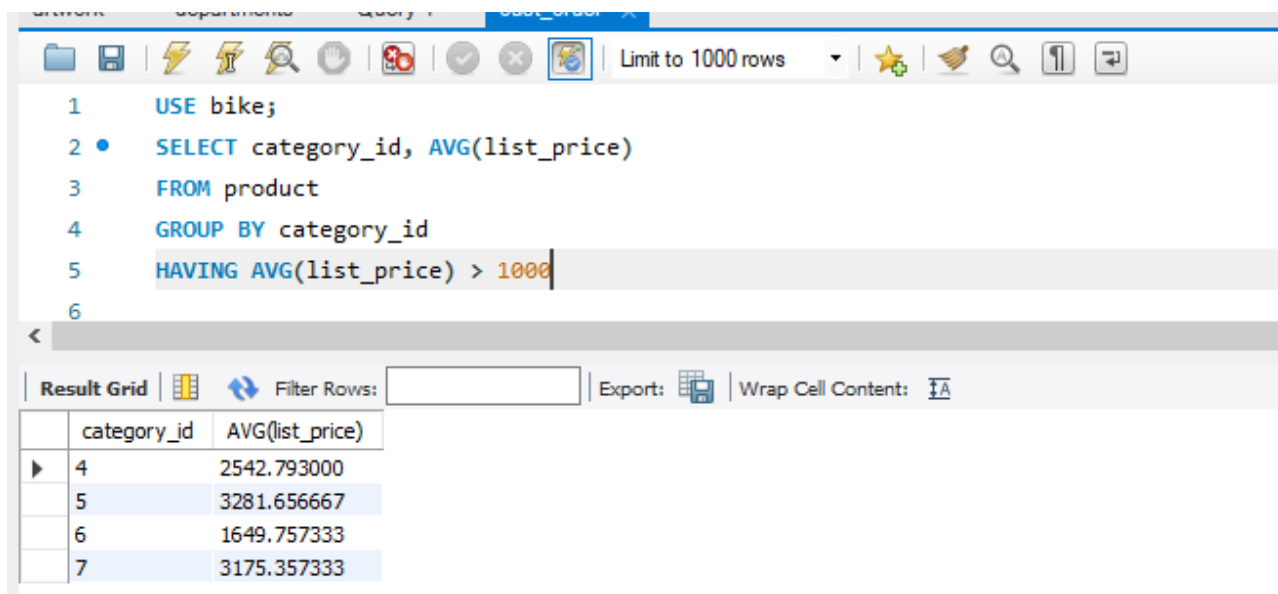
Filtering Aggregate Functions With The HAVING Clause

- The HAVING CLAUSE allows you to use an aggregate function as a filter. This is not allowed in a WHERE clause.
- Any columns or expressions you want to use in a HAVING clause, MUST BE DEFINED IN THE SELECT CLAUSE as well.

Code Sample:

```
USE bike;
SELECT category_id, AVG(list_price)
FROM product
GROUP BY category_id
HAVING AVG(list_price) > 1000
```

Output:



The screenshot shows a SQL IDE window with a query editor and a result grid. The query editor contains the following SQL code:

```
1  USE bike;
2  SELECT category_id, AVG(list_price)
3  FROM product
4  GROUP BY category_id
5  HAVING AVG(list_price) > 1000
6
```

The result grid displays the output of the query, showing the category_id and the average list price for each category. The results are as follows:

category_id	AVG(list_price)
4	2542.793000
5	3281.656667
6	1649.757333
7	3175.357333

We previously discussed the preceding lines of code for this query so we will focus solely on the HAVING clause.

HAVING AVG(list_price) > 1000

- The **HAVING** clause executes after the **GROUP BY** clause but before the **SELECT**
- If you use an aggregate function in the **HAVING** clause, you must include the same aggregate function in the **SELECT**
- If you reference a column or expression in the **HAVING** clause, you must include the same column or expression in the **SELECT**
- You cannot use aggregate functions in a **WHERE** clause



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/using_the_having_clause.

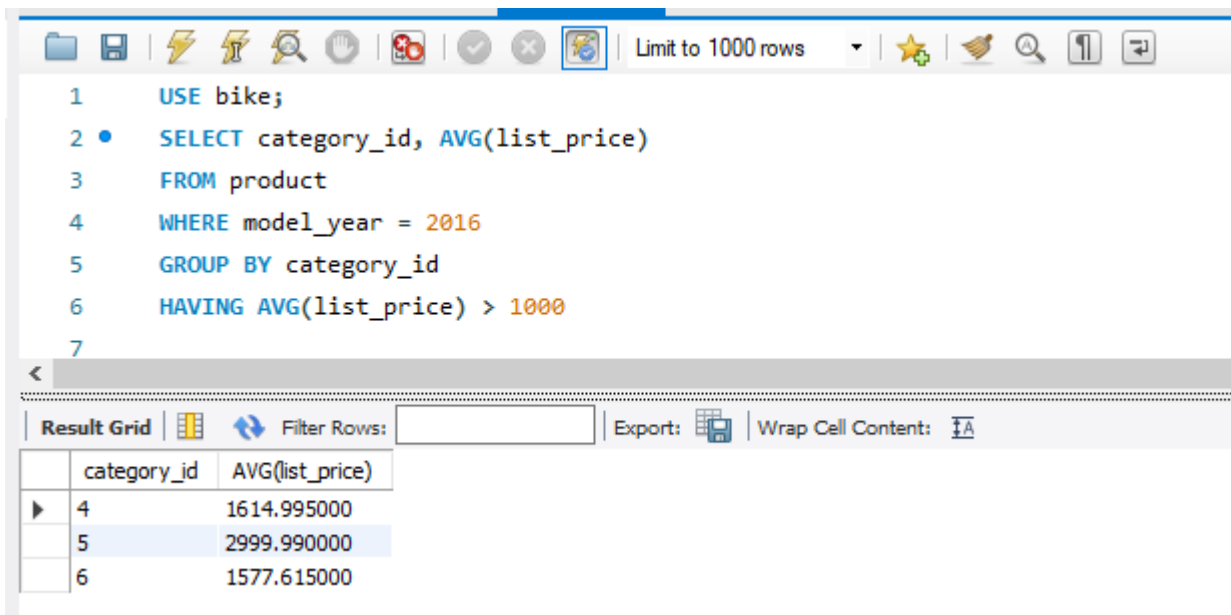
5.5

Using the HAVING and WHERE Clauses Together

Below is an example of a statement that includes both the HAVING and WHERE clause in the same SQL statement.

```
USE bike;
SELECT category_id, AVG(list_price)
FROM product
WHERE model_year = 2016
GROUP BY category_id
HAVING AVG(list_price) > 1000
```

Output:



```
1  USE bike;
2  SELECT category_id, AVG(list_price)
3  FROM product
4  WHERE model_year = 2016
5  GROUP BY category_id
6  HAVING AVG(list_price) > 1000
7
```

	category_id	AVG(list_price)
▶	4	1614.995000
	5	2999.990000
	6	1577.615000

WHERE model_year = 2016

- The **WHERE** clause executes before the **GROUP BY**
- You can refer to columns not defined in the **SELECT**
- You cannot use aggregate functions in the **WHERE**

HAVING AVG(list_price) > 1000

- The **HAVING** clause executes after the **GROUP BY** clause but before the **SELECT**
- If you use an aggregate function in the **HAVING** clause, you must include the same aggregate function in the **SELECT**
- If you reference a column or expression in the **HAVING** clause, you must include the same column or expression in the **SELECT**
- You cannot use aggregate functions in a **WHERE**



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/using_the_having_and.

5.6

COUNT(column_name) and COUNT(*)

How They Are Different

COUNT(column_name) and COUNT(*)

- COUNT(*) is the only aggregate function that counts rows with null values.
- When you specify a count based on a specific column, null values will not be counted.

Code Sample:

```
USE bike;  
SELECT COUNT(phone), COUNT(*)  
FROM CUSTOMER
```

Output:

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following code:

```
1 • USE bike;  
2 • SELECT COUNT(phone), COUNT(*)  
3 • FROM CUSTOMER
```

Below the editor, the 'Result Grid' tab is active, displaying the query results in a table:

	COUNT(phone)	COUNT(*)
▶	178	1445



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_difference_betwe.

Using the DISTINCT Statement

Removing Duplicate Values With DISTINCT

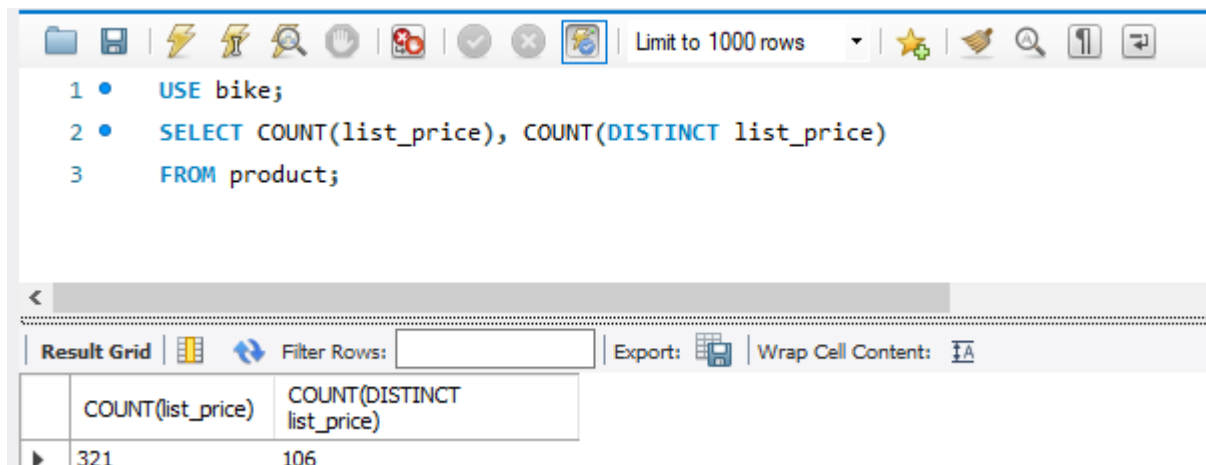
- The DISTINCT keyword allows you to eliminate duplicate rows in aggregate functions.
- You may also use the DISTINCT keyword with columns of the base table in a SELECT statement.
- COUNT(list_price) counts all the rows in the product table that have a list price.
- COUNT(DISTINCT list_price) eliminates duplicate values in the list_price.

Code Sample:

Example

```
USE bike;
SELECT COUNT(list_price), COUNT(DISTINCT list_price)
FROM product;
```

Output:



The screenshot shows a database IDE interface. At the top, there is a toolbar with various icons for file operations, execution, and search. Below the toolbar, the SQL code is displayed in a text area, numbered 1 to 3. The code is:
1 • USE bike;
2 • SELECT COUNT(list_price), COUNT(DISTINCT list_price)
3 FROM product;
Below the code, there is a horizontal scrollbar. At the bottom, there is a 'Result Grid' section. It contains a table with two columns: 'COUNT(list_price)' and 'COUNT(DISTINCT list_price)'. The first row of data shows the values 321 and 106 respectively. To the right of the table, there are options for 'Filter Rows', 'Export', and 'Wrap Cell Content'.

	COUNT(list_price)	COUNT(DISTINCT list_price)
▶	321	106



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/using_the_distinct_s.

Working With Subqueries

The Subquery In a SELECT Statement

The Subquery in an UPDATE statement

Create a Duplicate Table From An Existing Table

The Subquery In a Delete Statement



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/working_with_subquer.

The Subquery In a SELECT Statement

The Subquery in a SELECT Statement

- A subquery is a SELECT statement coded within another SELECT statement.
- A subquery can return a single value or a list of values.
- A subquery can return multiple columns.
- A subquery cannot make use of the ORDER BY clause
- A subquery can be nested within another subquery
- You can use a subquery in a WHERE, HAVING, FROM and SELECT clause.

Code Sample:

```
1  USE world;
2  SELECT name, population
3  FROM city
4  WHERE CountryCode IN
5      (SELECT code
6       FROM country
7       WHERE region = 'Caribbean')
8  ORDER BY population
9  LIMIT 5;
```

Results:

```

1 • USE world;
2 • SELECT name, population
3   FROM city
4  WHERE CountryCode IN
5    (SELECT code
6     FROM country
7     WHERE region = 'Caribbean')
8   ORDER BY population
9   LIMIT 5;

```

Filter to only items that have a CountryCode found in the results returned by the subquery

Subquery returns a list of codes from the country table.

Show only the first 5 rows

name	population
The Valley	595
South Hill	961
Plymouth	2000
Castries	2301
Willemstad	2345

SELECT name, population

FROM city

- Return the name and population columns from the city table.

WHERE CountryCode IN

- Filter to only rows that have a value found in the subsequent subquery.

(SELECT code

FROM country

WHERE region = 'Caribbean')

- The subquery shown above returns a result list of all of the codes from the country table that have a region of 'Caribbean'.
- The subquery must be in parentheses in MySQL.
- Each code (PK in country table) returned by the subquery is checked against CountryCode (FK in city table). If they match, the name and population are retrieved from the city table.





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_subquery_in_a_se.

6.2

The Subquery in an UPDATE statement

The Subquery in an UPDATE statement

- Subqueries may be used in an UPDATE statement
- Since it is possible to change many values at once with a subquery, take special care before running an UPDATE statement with a subquery. You might make a copy of the table and data you are trying to change to test with before running your statement on live data.
- It is also possible to run your UPDATE statement inside of a transaction block that allows you to ROLLBACK or undo a statement. We will address the topic of ROLLBACK in a future lesson.

Code Sample:

```
1 UPDATE country
2 SET GNPOld = 0.00
3 WHERE Code IN
4 (SELECT CountryCode FROM countrylanguage WHERE population = 0)
```

Results:

The screenshot shows a database management interface with a toolbar at the top containing icons for file operations, execution, and search. Below the toolbar, the SQL statement is entered in a text area:

```
1 • UPDATE country
2   SET GNPOld = 0.00
3   WHERE Code IN
4   (SELECT CountryCode
5    FROM countrylanguage
6    WHERE population = 0)
```

Below the SQL editor, there is an 'Output' section with a dropdown menu set to 'Action Output'. The output table shows the execution of the SQL statement:

#	Time	Action
1	15:14:45	UPDATE country SET GNPOld = 0.00 WHERE Code IN (SELECT CountryCode FROM countrylanguage WHERE population = 0)

UPDATE country

- Update the country table

SET GNPOld = 0.00

- Set the value of the GNPOld table = 0.00.
- No quotes are required because the GNPOld column is a decimal datatype

WHERE Code IN

- Update only the rows where the Code column value is in the results list returned in the subquery show below.

(SELECT CountryCode FROM countrylanguage WHERE population = 0)

- Return a list of values from the CountryCode column from the countrylanguage table that has a population equal to zero.
- If these values match a code in the country table, the row is updated.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_subquery_in_an_u.

Create a Duplicate Table From An Existing Table

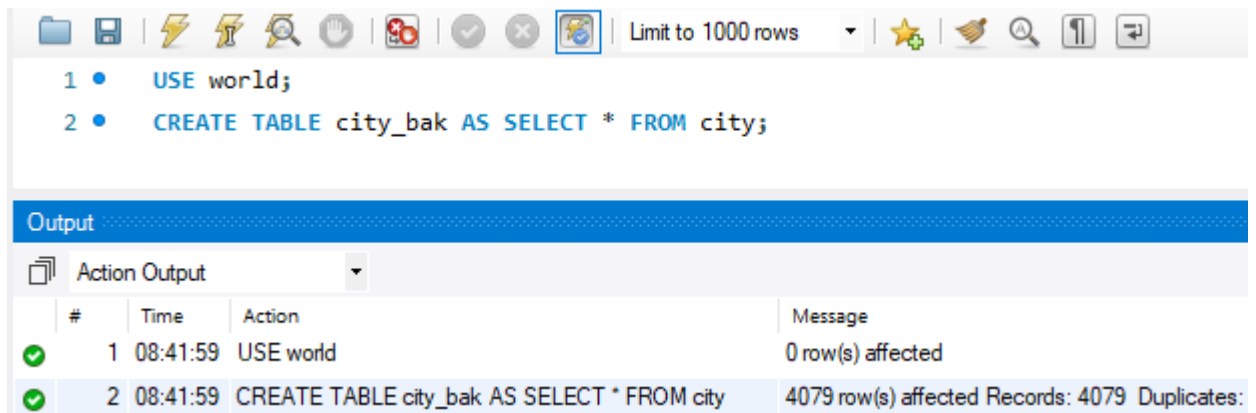
Create a Duplicate Table from an Existing Table with a Select Statement

- It is often helpful to create a duplicate table from an existing table for testing purposes
- You can combine the CREATE TABLE command with a select statement to create a duplicate of a table structure as well as the data in the table.

Code Sample:

```
1  USE world;
2  CREATE TABLE city_bak AS SELECT * FROM city;
```

Results:



The screenshot shows a database client window with a toolbar at the top. Below the toolbar, two SQL commands are listed:

- 1 • **USE world;**
- 2 • **CREATE TABLE city_bak AS SELECT * FROM city;**

Below the commands is an "Output" section with a dropdown menu set to "Action Output". It displays a table of execution results:

	#	Time	Action	Message
✓	1	08:41:59	USE world	0 row(s) affected
✓	2	08:41:59	CREATE TABLE city_bak AS SELECT * FROM city	4079 row(s) affected Records: 4079 Duplicates:

USE world;

- Select world as the default schema

CREATE TABLE city_bak AS SELECT * FROM city;

- Create a new table named city_bak with the exact same structure as the city table.
- Copy all of the data from the city table to the city_bak table



This content is provided to you freely by BYU-I Books.

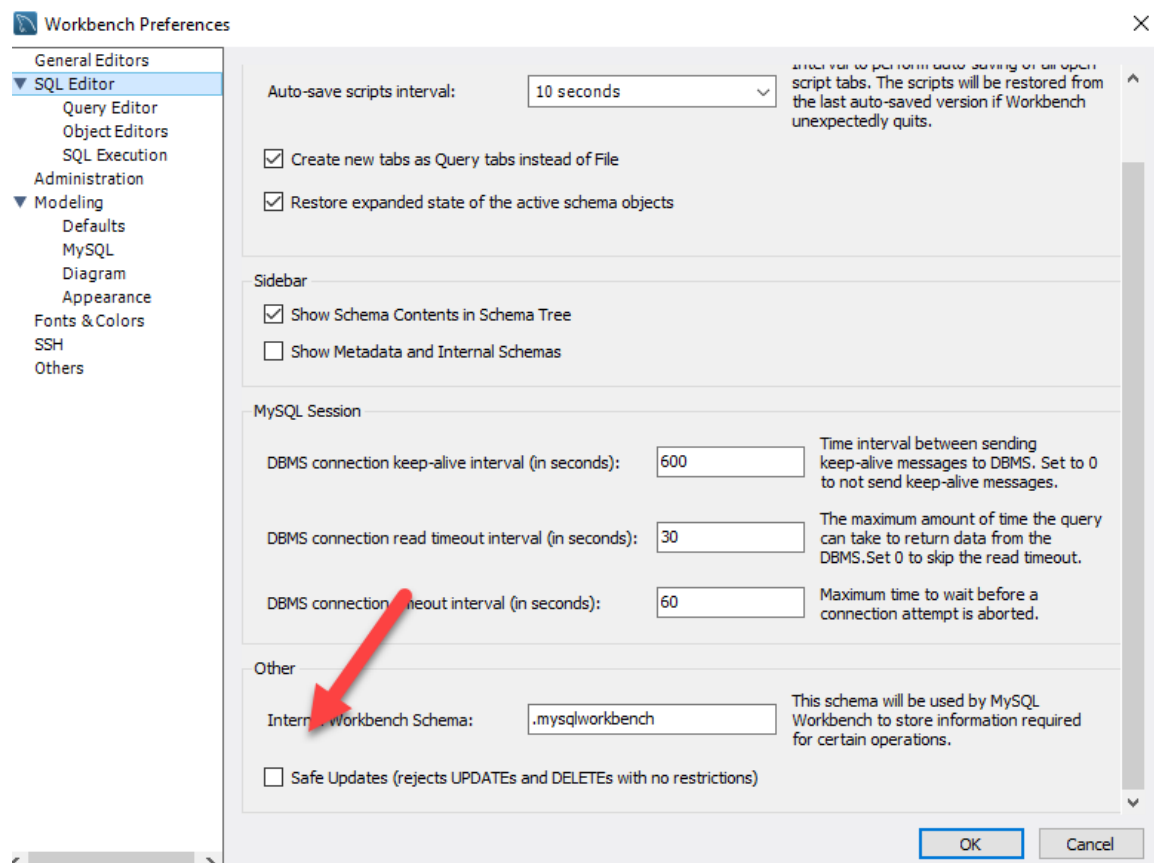
Access it online or download it at https://books.byui.edu/learning_mysql/create_a_duplicate_t.

The Subquery In a Delete Statement

The Subquery in a DELETE statement

- A subquery can be used in a DELETE statement.
- Always back up your data and test your DELETE statement before running it on live data.

NOTE: Before you can run a DELETE or UPDATE statement without a WHERE clause, you must uncheck "Safe Updates" checkbox in MySQL Preference. Please see below.



Code Sample:

```
USE world;
DELETE FROM city_bak
WHERE CountryCode IN
    (SELECT code FROM country
     WHERE region = 'Central Africa');
```

Results:



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The SQL editor contains the following query:

```
1 • USE world;
2 • DELETE FROM city_bak
3   WHERE CountryCode IN
4     (SELECT code FROM country
5      WHERE region = 'Central Africa');
```

Below the editor is the 'Output' pane, which is currently showing 'Action Output'. It contains a table with the following data:

#	Time	Action
1	10:49:11	USE world
2	10:49:11	DELETE FROM city_bak WHERE CountryCode IN (SELECT code FROM country WHERE region = 'Central Africa')

USE world;

- The tables used in this example are in the world database. Make sure it is selected as the default

DELETE FROM city_bak

- We are going to execute a DELETE statement on the city_bak table

WHERE CountryCode IN

- We are going to use a filter to delete items from the city_bak table where the CountryCode is found in a list of values that we will pass to it.

(SELECT code FROM country

WHERE region = 'Central Africa');

- We will execute a subquery on the country table and return a list of code values (PK to FK in city_bak table) where the region is equal to 'Central Africa'.
- You could accomplish the same thing by joining the city_bak table to the country table, then filtering on the region column from the country table.





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_subquery_in_a_de.

SQL Views

SQL View Explained
Benefits of Using Views
Views That Allow UPDATE Statements



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/sql_views.

SQL View Explained

SQL Views

- A SQL view is a SELECT statement that is stored as a database object.
- A SQL view acts as a virtual table but contains no data.
- You can use a view anywhere you would use a table including in a SELECT, INSERT, UPDATE, or DELETE statement.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/the_elements_of_a_vi.

Benefits of Using Views

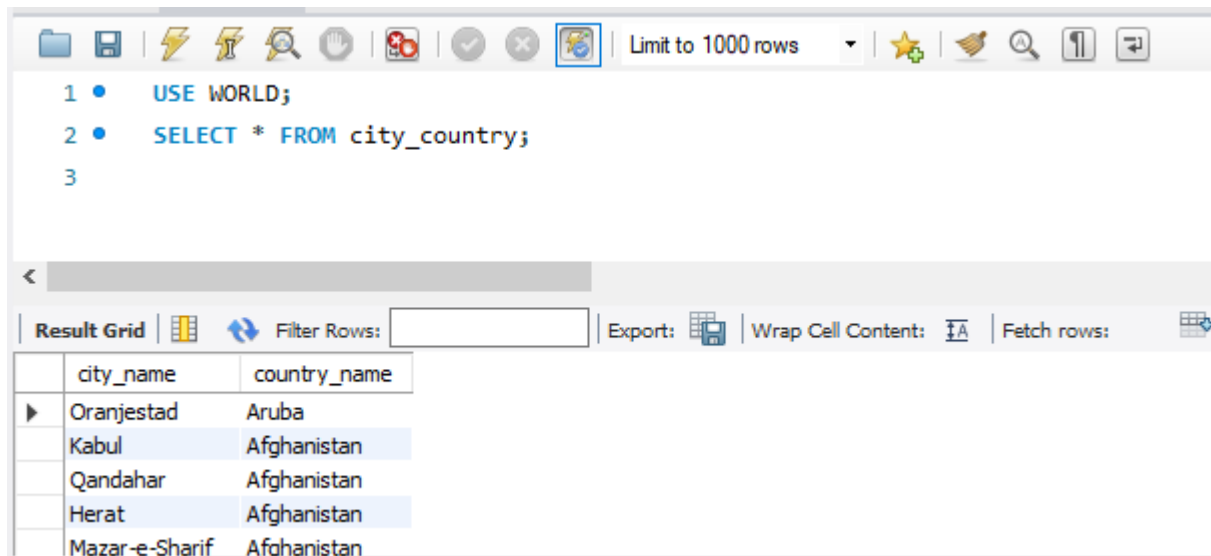
Benefits of Using Views

- **Design Flexibility:** By using a view instead of a query in an application, it is easier to make changes to the underlying table structure.
- **Improved Security:** By using a view to return data from tables instead of a SELECT, you can hide the WHERE clause or other columns to which you do not want the user to have access.
- **Query Simplification:** You can write simple select statements against views, which handle complex queries and joins.

Code Sample:

```
USE WORLD;  
CREATE VIEW city_country AS  
SELECT ci.name AS city_name, co.name AS country_name  
FROM city ci  
      JOIN country co  
      ON ci.CountryCode = co.Code;
```

Results by selecting from the city_country view:



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search, along with a 'Limit to 1000 rows' dropdown. The query editor contains the following SQL statements:

```

1 • USE WORLD;
2 • SELECT * FROM city_country;
3

```

Below the query editor is a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, a 'Wrap Cell Content' checkbox, and a 'Fetch rows' button. The result grid displays the following data:

city_name	country_name
Oranjestad	Aruba
Kabul	Afghanistan
Qandahar	Afghanistan
Herat	Afghanistan
Mazar-e-Sharif	Afghanistan

CREATE VIEW city_country AS

- Create a new VIEW object and give it the name city_country
- The AS statement precedes the query that will be assigned to the VIEW

SELECT ci.name AS city_name, co.name AS country_name

- Only the columns defined in the SELECT statement will be available to the VIEW
- It is a good idea to provide a column alias in the select because the VIEW will not have access to the underlying table structure.

FROM city ci

JOIN country co

ON ci.CountryCode = co.Code;

- The JOIN statement of the SELECT.
- Once you have created a VIEW, you can run SQL statements using the VIEW as if it were a table.
- By creating a VIEW, we can run selects that retrieve data from multiple tables without having to re-code a join.
- Notice how the SELECT * retrieves only the rows defined in the SELECT statement used in the VIEW creation.
- If you want to drop a VIEW, we can run the DROP VIEW statement
- If you want to modify an existing view you can use the statement CREATE OR REPLACE VIEW. That way you do not have to run a DROP VIEW statement and then a CREATE VIEW statement.





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/benefits_of_using_vi.

Views That Allow UPDATE Statements

Creating Views That Can Be Used With an UPDATE Statement

- There are some restrictions to creating a VIEW if you want to be able to run an UPDATE statement against it.
 - SELECT list cannot include a DISTINCT clause.
 - SELECT list cannot contain aggregate functions (SUM, COUNT, MIN, MAX, AVG, COUNT(*))
 - SELECT statement cannot use GROUP BY or HAVING. The VIEW cannot include a UNION operator.
- If you use any of the restricted statements, your view will be read-only.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/creating_views_that.

SQL Indexes

SQL Indexes Explained

Clustered vs. Non-clustered Indexes

Create an Index in Workbench Using an ERD

How to Manually Add an Index to an Existing Table



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/sql_indexes.

SQL Indexes Explained

SQL Indexes

- You can create SQL indexes from single or multiple columns.
- A SQL index is like the index of a book. It speeds up the retrieval of a record. The relational database management system (RDBMS) can retrieve a record with the index key instead of having to perform a table scan.
- MySQL automatically creates indexes for primary and foreign keys significantly speeding up join performance.
- You should only create indexes on columns used in a join or search because the RDMS must update an index every time you execute an INSERT, UPDATE, or DELETE.

When to Create an Index

- When a column is used frequently in a search or a join.
- When a column contains a large number of distinct values.
- When the column is updated infrequently



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/sql_indexes_explaine.

Clustered vs. Non-clustered Indexes

Clustered vs. Non-clustered Indexes

- Clustered index: The values in the column indexed are physically stored in alphabetical or numeric order.
 - You can only have one clustered index per table.
 - If you assign a primary key, the system automatically creates a clustered index on that column.
 - If no primary key is defined on a table, the first column that has an index defined for it becomes the clustered index.
- Non-clustered index: Column values are not in alphabetical or numeric order
 - You can add as many non-clustered indexes to a table as you want.
 - You should only create additional non-clustered indexes on a table if you need to search or perform a join on that column. When you create a foreign key column, a non-clustered index is automatically created for that column.



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/when_to_create_an_in.

8.3

Create an Index in Workbench Using an ERD

- Right-click on the table and select 'Edit'
- Click on the 'Indexes' tab
- Type the name of the index in the 'Index Name' field
- Under 'Type' select 'INDEX' • Click on the column(s) that you want to index.
- Tab to a new line

The screenshot shows the MySQL Workbench interface. At the top, the 'Diagram' tab is active, displaying a table structure for 'students' with columns: id INT, fname VARCHAR(45), lname VARCHAR(45), gender ENUM('m', 'f'), city VARCHAR(45), state_id INT, and dob DATE. Below this, the 'students - Table' tab is selected, showing the 'Indexes' configuration for the 'students' table in the 'enrollment' schema.

Index Name	Type
PRIMARY	PRIMARY
id_UNIQUE	UNIQUE
fk_student_state_idx	INDEX
idx_lname	INDEX

The 'Index Columns' section shows the following configuration:

Column	#	Order	Length
<input type="checkbox"/> id		ASC	
<input checked="" type="checkbox"/> fname	1	ASC	
<input checked="" type="checkbox"/> lname		ASC	
<input type="checkbox"/> gender		ASC	
<input type="checkbox"/> city		ASC	
<input type="checkbox"/> state_id		ASC	
<input type="checkbox"/> dob		ASC	

The 'Index Options' section on the right includes fields for Storage Type, Key Block Size (0), Parser, and Visible (unchecked). The Index Comment field is also present.



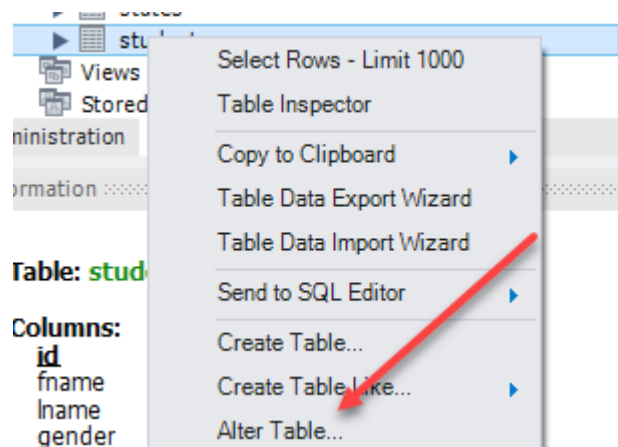


This content is provided to you freely by BYU-I Books.

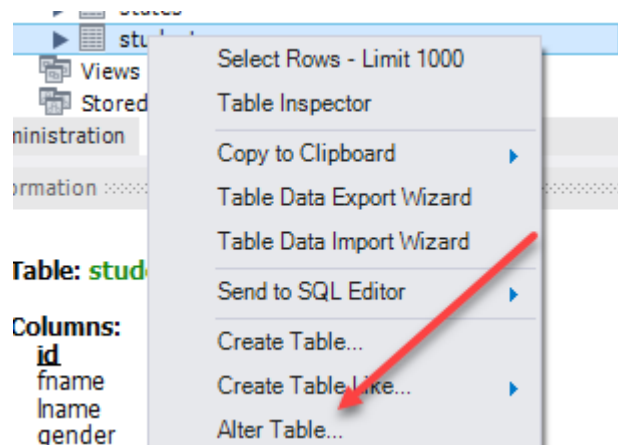
Access it online or download it at https://books.byui.edu/learning_mysql/clustered_vs_non_clu.

How to Manually Add an Index to an Existing Table

- Right-click on the table
- Select 'Alter Table'



- Click on the 'Indexes' tab



- Type the name of the index in the 'Index Name' field
- Under 'Type' select 'INDEX'
- Click on the column(s) that you want to index.
- Tab to a new line




Table Name:

Schema: **enrollment**

Charset/Collation:

Engine:

Comments:

Index Name	Type
PRIMARY	PRIMARY
id_UNIQUE	UNIQUE
fk_student_state_idx	INDEX
idx_fname	INDEX

Index Columns

Column	#	Order	Length
<input type="checkbox"/> id		ASC	
<input checked="" type="checkbox"/> fname	1	ASC	
<input type="checkbox"/> lname		ASC	



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/how_to_manually_add_.

Glossary

 Find something...

Aggregate Function

Performs an operation on a set of records in a column and returns a single value.

Arithmetic Operators

Arithmetic operators ARE: * (multiplication), / (division), DIV (integer division), % (MOD) or remainder, + (addition), - (subtraction). These operators can be used in the SELECT, WHERE, and ORDER BY clauses. Operators are evaluated in the same way as arithmetic in other contexts.

AVG function

Returns the average of the non-null columns in the expression.

BETWEEN operator

The BETWEEN operator is similar to \geq and \leq . BETWEEN includes everything between the two values indicated. BETWEEN works with both text and number.

CEILING function

Returns the next highest whole number no matter what the decimal point.

Column Aliases

A column alias provides a way to create a clean or more descriptive header for a results set. A column alias cannot be used in a SELECT, WHERE, GROUP BY or HAVING clause due to the order of execution. You must refer to the original column name.

Column Specifications

A column specification may be derived from a base table. Or it may be a calculated value as a result of an arithmetic expression or a function.

Comparison Operators

The comparison operators are = (equals), < (less than), > (greater than), <= (less than or equal to), >=, <> (not equal), != (not equal). Comparison operators compare two expressions. The result of a comparison results to true or false. Comparison operators are not case sensitive and are used with text and dates as well as numbers.

Compound condition

When more than one logical operator (AND, OR, NOT) is used in the WHERE clause.

CONCAT function

Combines a list of strings into a single string.

COUNT function

The number of the non-null columns in the expression.

CURRENT_DATE function

Returns current local date.

CURRENT_TIME function

Returns current local time

DATE function

Extracts the date from date/time input. If time is included it is dropped.

DELETE clause

SQL clause that deletes data from a table.

DISTINCT clause

The DISTINCT clause removes duplicate rows from a query.

FLOOR function

Returns the next lowest whole number no matter what the decimal point.

FROM clause

Specifies the base table(s) from which results will be retrieved.

GROUP BY clause

Groups rows of a result set based on columns or expressions separated by commas.

HAVING clause

The HAVING CLAUSE allows you to use an aggregate function as a filter. This is not allowed in a WHERE clause.

IN operator

The IN operator tests whether an expression is equal to a value or values in a list of expressions. The order of the items in the list does not matter. You can use the NOT operator to test for items not in the list. The IN clause may be used with a subquery.

Indexes

A SQL index is like the index of a book. It speeds up the retrieval of a record. The relational database management system (RDBMS) can retrieve a record with the index key instead of having to perform a table scan.

INSERT clause

SQL Clause used to insert data into a table.

IS NULL function

Null values indicate an unknown or non-existent value and is different from an empty string (' '). To test for a null value you use the IS NULL clause. The test for a value use IS NOT NULL clause

JOIN (OUTER) clause

An outer join will return all the rows from one table and only the rows from the other table that match the join condition

JOIN clause

A JOIN clause allows you to access data from two or more tables in a query.

LEFT function

Returns a substring starting from the left side of the string.

LIKE operator

The LIKE keyword is used with the WHERE clause. The LIKE keyword and can use two symbols as wildcards. The percent (%) symbol matches any number of characters and the underscore (_) matches a single character.

LIMIT clause

Specifies the number of rows to be returned.

Logical Operators: AND, OR, NOT

Logical operators are used in the WHERE clause. You may use multiple logical operators in a WHERE clause to create a compound condition. The order of evaluation when multiple operators are used is shown in the table above.

LTRIM function

Removes leading spaces from a string.

MIN function

The lowest value of the non-null columns in the expression

NOW function

Returns current local date and time.

ORDER BY clause

SQL clause that orders a result set.

REGEXP operator

REGEXP operator allows you to do more complex pattern matching than a LIKE keyword. Some version of REGEXP exists in many computer languages. Refer to the "LIKE and REGEXP" handout for a full list of examples.

RIGHT function

Returns a substring starting from the right side of the string.

ROUND function

Rounds to the decimal specified.

RTRIM function

Removes trailing spaces from a string.

SELECT clause

Specifies the columns that will appear in a SQL query result set.

Subquery

A subquery is a SELECT statement coded within another SELECT statement.

SUM function

The total of the non-null columns in the expression.

Summary Query

A query that uses at least one aggregate function.

TRIM function

Removes leading and trailing spaces from a string.

TRUNCATE function

Returns the number truncated to the precision specified.

UNION clause

A UNION combines the results of two or more queries into a single result set.

UPDATE clause

SQL clause that updates data in a table.

UTC_DATE function

Returns current UTC date.

UTC_time function

Returns current UTC time.

VIEWS

A SQL view is a SELECT statement that is stored as a database object.

WHERE function

Specifies any conditions for the results set (filter).



This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/glossary.

Index



aggregate functions

Using the HAVING Clause

1. [Filtering **aggregate functions** With The HAVING Clause](#)

Aggregate Functions

1. [aggregate functions](#)
2. [Table 1. **aggregate functions** List](#)

and

The DELETE Clause

1. [... WHERE name = 'san felipe' **and** countrycode = 'chl';](#)
2. [WHERE name = 'san felipe' **and** countrycode = 'chl';](#)

Grouping Data

1. [Filtering With WHERE **and** HAVING](#)

Using the HAVING and WHERE Clauses Together

1. [... that includes both the HAVING **and** WHERE clause in the same SQL statement.](#)

COUNT(column_name) and COUNT(*)

1. [COUNT\(column_name\) **and** COUNT\(*\)](#)

Introduction

1. [... databases that you can download **and** install in your local MySQL instance....](#)
2. [... will include SQL design basics **and** guidance on how to install MySQL **and** MySQL...](#)

How to Retrieve Data From a Single Table

1. [LIKE **and** REGEXP Operators](#)
2. [and, OR, NOT Logical Operators](#)
3. [and](#)
4. [Separates two string patterns **and** matches either one](#)
5. [... countryWHERE region = 'caribbean'**and** population > 100000ORDER BY population...](#)
6. [... countryWHERE name BETWEEN "Aruba" **and** "Bahamas";](#)
7. [Table 4. Operators **and** precedence order](#)
8. [\(a **and** b\) –If both a **and** b are present, item is included](#)

LIKE and REGEXP Operators

1. [LIKE **and** REGEXP Operators](#)
2. [Separates two string patterns **and** matches either one](#)

Arithmetic Operators

1. [Table 4. Operators **and** precedence order](#)

IS NULL, BETWEEN, IN Operators

1. [... countryWHERE name BETWEEN "Aruba" **and** "Bahamas";](#)

AND, OR, NOT Logical Operators

1. [and, OR, NOT Logical Operators](#)
2. [and](#)
3. [... countryWHERE region = 'caribbean'**and** population > 100000ORDER BY population...](#)
4. [\(a **and** b\) –If both a **and** b are present, item is included](#)

The JOIN Clause

1. [... table aliases of co for country **and** ci for city are defined in the FROM clause...](#)

Date Functions

1. [* Returns current local date **and** time.](#)

String Functions

1. [LOCATE\(\), **and** LENGTH\(\) accept a string but return an integer. • SUBSTRING\(\)...](#)

arithmetic operators

How to Retrieve Data From a Single Table

1. [arithmetic operators](#)

Arithmetic Operators

1. [arithmetic operators](#)

avg

Simple GROUP BY Query

1. [USE bike;SELECT category_id, avg\(list_price\)FROM productGROUP BY category_id](#)
2. [SELECT category_id, avg\(list_price\):](#)

Improving the GROUP BY Query

1. [... category_name, __ CONCAT\('\\$', ROUND\(avg\(list_price\),2\)\) AS 'Average List...](#)
2. [__ CONCAT\('\\$', ROUND\(avg\(list_price\),2\)\) AS 'Average List Price'](#)

Using the HAVING Clause

1. [HAVING avg\(list_price\) > 1000](#)
2. [USE bike;SELECT category_id, avg\(list_price\)FROM productGROUP BY category_idHAVING...](#)

Using the HAVING and WHERE Clauses Together

1. [USE bike;SELECT category_id, avg\(list_price\)FROM productWHERE model_year =...](#)
2. [HAVING avg\(list_price\) > 1000](#)

Aggregate Functions

1. [USE bike;SELECT avg\(list_price\), SUM\(list_price\), MIN\(list_price\), __...](#)
2. [avg\(\[DISTINCT\] column_values\)](#)

between

How to Retrieve Data From a Single Table

1. [between Operators](#)
2. [... IndepYearFROM countryWHERE name between "Aruba" and "Bahamas";](#)

IS NULL, BETWEEN, IN Operators

1. [between Operators](#)
2. [... IndepYearFROM countryWHERE name between "Aruba" and "Bahamas";](#)

ceiling

Numeric Functions

1. [FLOOR, ceiling, TRUNCATE](#)
2. [... list_price, FLOOR\(list_price\), ceiling\(list_price\), TRUNCATE\(list_price,...](#)
3. [ceiling\(number\)](#)
4. [ceiling\(6.2\)](#)
5. [Table 6. FLOOR, ceiling, TRUNCATE functions](#)

column

How to Retrieve Data From a Single Table

1. [column Aliases](#)
2. [Show all columns](#)
3. [Comma separated list of column names](#)
4. [... previous example, we created a new column that was a calculated value. The problem...](#)
5. [column Name](#)
6. [Table 1. column Specifications](#)
7. [... then in quotes we put the new column alias of "People per square mile."...](#)

The Five Clauses of the SELECT Statement

1. [Show all columns](#)
2. [Comma-separated list of column names](#)
3. [column Name](#)
4. [Table 1. column Specifications](#)

Column Specifications

1. [column Specifications](#)
2. [Show all columns](#)
3. [column Name](#)
4. [Comma separated list of column names](#)
5. [column Specifications](#)

Column Aliases

1. [column Aliases](#)
2. [... previous example, we created a new column that was a calculated value. The problem...](#)
3. [... then in quotes we put the new column alias of "People per square mile."...](#)

The JOIN Clause

1. [... whole table name to qualify a column, you can use a table alias.](#)

The INSERT Clause With a Column List

1. [The INSERT Clause With a column List](#)
2. [... of an INSERT statement with a column list:](#)

The INSERT Clause Without a Column List

1. [The INSERT Clause Without a column List](#)

Grouping Data

1. [... rows of a result set based on **columns** or expressions separated by commas.](#)

COUNT(column_name) and COUNT(*)

1. [COUNT\(**column_name**\) and COUNT\(*\)](#)

Aggregate Functions

1. [SUM\(\[DISTINCT\] **column_values**\)](#)
2. [MIN\(\[DISTINCT\] **column_values**\)](#)
3. [The average of the non-null **columns** in the expression](#)
4. [MAX\(\[DISTINCT\] **column_values**\)](#)
5. [COUNT\(\[DISTINCT\] **column_values**\)](#)
6. [... highest value of the non-null **columns** in the expression](#)
7. [AVG\(\[DISTINCT\] **column_values**\)](#)
8. [The total of the non-null **columns** in the expression](#)
9. [... lowest value off the non-null **columns** in the expression](#)
10. [The number of the non-null **columns** in the expression](#)

column aliases

How to Retrieve Data From a Single Table

1. [column aliases](#)

Column Aliases

1. [column aliases](#)

column specifications

How to Retrieve Data From a Single Table

1. [Table 1. **column specifications**](#)

The Five Clauses of the SELECT Statement

1. [Table 1. **column specifications**](#)

Column Specifications

1. [column specifications](#)
2. [column specifications](#)

comparison operators

How to Retrieve Data From a Single Table

1. [comparison operators](#)
2. [Table 5. **comparison operators**](#)

Comparison Operators

1. [comparison operators](#)
2. [Table 5. comparison operators](#)

concat

String Functions

1. [concat](#)
2. [USE world;SELECT concat\(name,'_',continent\)FROM country;](#)

Improving the GROUP BY Query

1. [... bike;SELECT category_name,concat\('\\$',ROUND\(AVG\(list_price\),2\)\)AS 'Average...'](#)
2. [concat\('\\$',ROUND\(AVG\(list_price\),2\)\)AS 'Average List Price'](#)

count

The INSERT Clause With a Column List

1. [\(name,countryCode,district,population\)](#)
2. [... INTO city 3 \(name,countryCode,district,population\) 4 VALUES 5...](#)

The DELETE Clause

1. [... WHERE name = 'san felipe' AND countryCode = 'chl';](#)
2. [WHERE name = 'san felipe' AND countryCode = 'chl';](#)

COUNT(column_name) and COUNT(*)

1. [count\(column_name\) and count\(*\)](#)
2. [USE bike;SELECT count\(phone\),count\(*\) FROM CUSTOMER](#)

Using the DISTINCT Statement

1. [ExampleUSE bike;SELECT count\(list_price\),count\(DISTINCT list_price\) FROM product;](#)

The Subquery in an UPDATE statement

1. [\(SELECT countryCode FROM country|language WHERE population = 0\).](#)
2. [1 UPDATE country 2 SET GNPOld = 0.00 3 WHERE Code IN 4 ...](#)
3. [UPDATE country](#)

The Subquery In a Delete Statement

1. [\(SELECT code FROM country](#)
2. [... world;DELETE FROM city_bakWHERE countryCode IN \(SELECT code FROM country ...](#)
3. [WHERE countryCode IN](#)

Benefits of Using Views

1. [USE WORLD;CREATE VIEW city_country AS SELECT ci.name AS city_name, co.name AS...](#)
2. [CREATE VIEW city_country AS](#)
3. [... ci.name AS city_name, co.name AS country_name](#)
4. [... JOIN country co](#)
5. [... ON ci.countryCode = co.Code;](#)
6. [Results by selecting from the city_country view:](#)

Aggregate Functions

1. [count\(*\)](#)
2. [... MIN\(list_price\), ... MAX\(list_price\), count\(list_price\), count\(*\)FROM product;](#)
3. [count\(\[DISTINCT\] column_values\)](#)

The Subquery In a SELECT Statement

1. [WHERE countryCode IN](#)
2. [... FROM city4 WHERE countryCode IN 5 \(SELECT code 6 ...](#)
3. [FROM country](#)

How to Retrieve Data From a Single Table

1. [USE world;SELECT nameFROM countryWHERE name IN \('Aruba', 'Barbados', 'Cuba',...](#)
2. [... "People per square mile"FROM country;](#)
3. [SELECT name, IndepYearFROM countryWHERE IndepYear IS NULL;](#)
4. [USE world;SELECT nameFROM countryWHERE name REGEXP 'g\[o,u\]';](#)
5. [... world;SELECT name, populationFROM countryWHERE population > 1000000;](#)
6. [... world;SELECT name, populationFROM countryWHERE region = 'caribbean'AND population...](#)
7. [... "People per square mile"FROM country;](#)
8. [... world;SELECT name, IndepYearFROM countryWHERE name BETWEEN "Aruba" and "Bahamas";](#)
9. [... DISTINCT continent, nameFROM countryORDER BY continent;](#)
10. [USE world;SELECT nameFROM countryWHERE name LIKE 'A%'](#)

The Five Clauses of the SELECT Statement

1. [... name3 FROM city4 WHERE countryCode = "AFG"5 ORDER BY name6...](#)

LIKE and REGEXP Operators

1. [USE world;SELECT nameFROM countryWHERE name REGEXP 'g\[o,u\]';](#)
2. [USE world;SELECT nameFROM countryWHERE name LIKE 'A%'](#)

Arithmetic Operators

1. [... "People per square mile"FROM country;](#)

Column Aliases

1. [... "People per square mile"FROM country;](#)

Comparison Operators

1. [... world;SELECT name, populationFROM countryWHERE population > 1000000;](#)

IS NULL, BETWEEN, IN Operators

1. [USE world;SELECT nameFROM **country**WHERE name IN \('Aruba', 'Barbados', 'Cuba',...](#)
2. [SELECT name, IndepYearFROM **country**WHERE IndepYear IS NULL;](#)
3. [... world;SELECT name, IndepYearFROM **country**WHERE name BETWEEN "Aruba" and "Bahamas";](#)

AND, OR, NOT Logical Operators

1. [... world;SELECT name, populationFROM **country**WHERE region = 'caribbean'AND population...](#)

DISTINCT Clause

1. [... DISTINCT continent, nameFROM **country**ORDER BY continent;](#)

The JOIN Clause

1. [... "City Name", co.name AS "**country** Name"](#)
2. [JOIN **country** co](#)
3. [ON ci.**country**Code = co.Code;](#)
4. [... AS "City Name", 3 **country**.name AS "**country** Name" 4 FROM **country** 6...](#)
5. [... aliases. The table aliases of co for **country** and ci for city are defined in...](#)
6. [... Name", 3 co.name AS "**country** Name" 4 FROM city ci 5 ...](#)

Joining More Than Two Tables

1. [ON cl.**country**Code = ci.**country**Code;](#)
2. [JOIN **country**language cl.](#)
3. [... Name", 3 co.name AS "**country** Name", 4 cl.language AS...](#)

The OUTER JOIN Clause

1. [ON c.code = cl.**country**Code](#)
2. [FROM **country** c LEFT JOIN **country**language cl](#)
3. [... c.continent, cl.language3 FROM **country** c LEFT JOIN **country**language cl4 ON c.code...](#)

How to Code a UNION

1. [SELECT name, populationFROM **country**WHERE continent = 'Oceania'](#)
2. [... name, populationFROM cityWHERE **country**Code = 'AUS'](#)
3. [... name, population3 FROM city WHERE **country**Code = 'AUS'4 UNION5 SELECT name,...](#)

Date Functions

1. [... DATE_FORMAT\('2020-01-28', '%m/%d/%y'\)FROM **country**;](#)

Numeric Functions

1. [... ROUND\(LifeExpectancy\) FROM world.**country**;](#)

String Functions

1. [... CONCAT\(name, ',', continent\)FROM **country**;](#)

current_date

Date Functions

1. [current_date\(\)](#)
2. [current_date](#)
3. [... DATE\('2020-01-01'\) AS 'DATE\(\)', date only', current_date AS 'current_date', CURRENT_TIME...](#)

current_time

Date Functions

1. [current_time](#)
2. [... CURRENT_DATE AS 'CURRENT_DATE', current_time AS 'current_time', UTC_DATE...](#)
3. [current_time\(\)](#)

date

Date Functions

1. [date_FORMAT](#)
2. [date_ADD](#)
3. [dateDIFF](#)
4. [Current date/Time Functions](#)
5. [date, dateTIME](#)
6. [Table 1. Current date Functions](#)
7. [CURRENT_date\(\)](#)
8. [date](#)
9. [date\(date\)](#)
10. [CURRENT_date](#)
11. [Table 3. date_FORMAT Function](#)
12. [* Returns current local date](#)
13. [date_FORMAT](#)
14. [date/time](#)
15. [Table 2. date_ADD Function](#)
16. [date](#)
17. [SELECT NOW\(\) AS 'NOW\(\)', date\('2020-01-01'\) AS 'date\(\)', date only', ...](#)
18. [• dates must be enclosed in quotes • You can pass a date or dateTIME datatype...](#)
19. [SELECT dateDIFF\('2018-01-01', '2019-01-01'\) AS 'date Difference';](#)
20. [* extracts the date from input. If time is included, the time is dropped.](#)
21. [date_FORMAT\('2020-09-03', '%m/%d/%y'\)](#)
22. [... world;SELECT name, continent, date_FORMAT\('2020-01-28', '%m/%d/%y'\)FROM country;](#)
23. [* Returns current local date and time.](#)
24. [date/time](#)
25. [USE bike;SELECT order_date, date_ADD\(order_date, INTERVAL 1 DAY\) AS 'ORDER...](#)
26. [* Returns current UTC date.](#)
27. [date](#)
28. [date\('2020-01-01 11:31:31'\)](#)
29. [date_ADD\(date, interval expression unit\)](#)
30. [* Returns current UTC date.](#)
31. [UTC_date\(\)](#)
32. [UTC_date](#)
33. [date_ADD\('2020-01-01', INTERVAL 1 DAY\)](#)
34. [• Returns a date with a date or dateTIME value equal to the original value...](#)

The UPDATE Clause With a Column List

1. [The UPDATE Clause](#)
2. [UPDATE city](#)
3. [1 USE world; 2 UPDATE city 3 SET Population = 65000, district...](#)

The Subquery in an UPDATE statement

1. [The Subquery in an UPDATE statement](#)
2. [1 UPDATE country 2 SET GNPOld = 0.00 3 WHERE Code IN 4 ...](#)
3. [UPDATE country](#)

The Subquery In a Delete Statement

1. [... Before you can run a DELETE or UPDATE statement without a WHERE clause, you...](#)

Views That Allow UPDATE Statements

1. [... Views That Can Be Used With an UPDATE Statement](#)

Aggregate Functions

1. [numeric, date, string](#)
2. [numeric, date, string](#)

delete

The DELETE Clause

1. [The delete Clause](#)
2. [1 USE world;2 delete 3 FROM city 4 WHERE name = 'san felipe'...](#)
3. [delete](#)

The Subquery In a Delete Statement

1. [The Subquery in a delete statement](#)
2. [delete FROM city_bak](#)
3. [USE world;delete FROM city_bakWHERE CountryCode IN \(SELECT code FROM country ...](#)
4. [NOTE: Before you can run a delete or UPDATE statement without a WHERE clause,...](#)

distinct

Using the DISTINCT Statement

1. [Removing Duplicate Values With distinct](#)
2. [... bike;SELECT COUNT\(list_price\),COUNT\(distinct list_price\) FROM product;](#)

Aggregate Functions

1. [SUM\(\[distinct\] column_values\)](#)
2. [MIN\(\[distinct\] column_values\)](#)
3. [MAX\(\[distinct\] column_values\)](#)
4. [COUNT\(\[distinct\] column_values\)](#)
5. [AVG\(\[distinct\] column_values\)](#)

How to Retrieve Data From a Single Table

1. [distinct Keyword](#)
2. [distinct](#)
3. [Table 7. distinct Keyword](#)
4. [SELECT distinct continent, nameFROM countryORDER BY continent;](#)

DISTINCT Clause

1. [distinct Keyword](#)
2. [distinct](#)
3. [Table 7. distinct Keyword](#)
4. [SELECT distinct continent, name FROM country ORDER BY continent;](#)

floor

Numeric Functions

1. [floor, CEILING, TRUNCATE](#)
2. [floor\(7.7\)](#)
3. [USE bike; SELECT list_price, floor\(list_price\), CEILING\(list_price\), TRUNCATE\(list_price, ...](#)
4. [floor\(number\)](#)
5. [Table 6. floor, CEILING, TRUNCATE functions](#)

group by

Grouping Data

1. [Using the group by Clause](#)
2. [Table 1. group by Function](#)
3. [group by](#)

Simple GROUP BY Query

1. [group by category_id:](#)
2. [... category_id, AVG\(list_price\) FROM product group by category_id](#)

Improving the GROUP BY Query

1. [Improving the group by Query](#)
2. [... p.category_id = c.category_id group by category_name ORDER BY category_name;](#)
3. [group by category_name](#)

Using the HAVING Clause

1. [... category_id, AVG\(list_price\) FROM product group by category_id HAVING AVG\(list_price\) ...](#)

Using the HAVING and WHERE Clauses Together

1. [... product WHERE model_year = 2016 group by category_id HAVING AVG\(list_price\) > 1000](#)

having

Grouping Data

1. [Filtering With WHERE And having](#)

Using the HAVING Clause

1. [... Aggregate Functions With The **having** Clause](#)
2. [... **having** AVG\(list_price\) > 1000](#)
3. [... AVG\(list_price\)FROM productGROUP BY category_id**having** AVG\(list_price\) > 1000](#)
4. [... so we will focus solely on the **having** clause.](#)

Using the HAVING and WHERE Clauses Together

1. [... model_year = 2016GROUP BY category_id**having** AVG\(list_price\) > 1000](#)
2. [... **having** AVG\(list_price\) > 1000](#)
3. [... statement that includes both the **having** and WHERE clause in the same SQL statement.](#)

in

The INSERT Clause With a Column List

1. [The **in**SERT Clause With a Column List](#)
2. [Below is a basic example of an **in**SERT statement with a column list:](#)
3. [inSERT inTO city](#)
4. [Results of the insert:](#)
5. [1 USE world;2 inSERT inTO city 3 \(name, countryCode,...](#)

The INSERT Clause Without a Column List

1. [The **in**SERT Clause Without a Column List](#)
2. [1 USE world;2 inSERT inTO city 3 VALUES 4 \(DEFAULT,...](#)

Grouping Data

1. [Filtering With WHERE And HAVinG](#)
2. [Using the GROUP BY Clause](#)

Improving the GROUP BY Query

1. [Improving the GROUP BY Query](#)
2. [... List Price'FROM product p JOIN category c ON p.category_id = c.category_idGROUP...](#)
3. [... JOIN category c](#)

Using the HAVING Clause

1. [Filtering Aggregate Functions With The HAVinG Clause](#)
2. [HAVinG AVG\(list_price\) > 1000](#)
3. [... productGROUP BY category_idHAVinG AVG\(list_price\) > 1000](#)
4. [... previously discussed the preceding lines of code for this query so we will...](#)

Using the HAVING and WHERE Clauses Together

1. [... = 2016GROUP BY category_idHAVinG AVG\(list_price\) > 1000](#)
2. [HAVinG AVG\(list_price\) > 1000](#)
3. [... an example of a statement that includes both the HAVinG and WHERE clause...](#)

Using the DISTINCT Statement

1. [Removing Duplicate Values With DISTinCT](#)
2. [... COUNT\(list_price\), COUNT\(DISTinCT list_price\) FROM product;](#)

The Subquery in an UPDATE statement

1. [The Subquery in an UPDATE statement](#)
2. [... GNPOld = 0.00 3 WHERE Code in 4 \(SELECT CountryCode FROM countrylanguage...](#)
3. [WHERE Code in](#)

Create a Duplicate Table From An Existing Table

1. [... Duplicate Table from an Existing Table with a Select Statement](#)

The Subquery In a Delete Statement

1. [The Subquery in a DELETE statement](#)
2. [... FROM city_bak WHERE CountryCode in \(SELECT code FROM country ...](#)
3. [WHERE CountryCode in](#)
4. [... uncheck "Safe Updates" checkbox in MySQL Preference. Please see below.](#)

Benefits of Using Views

1. [Benefits of Using Views](#)
2. [... country_name FROM city ci JOIN country co ON ci.CountryCode = co.Code;](#)
3. [JOIN country co](#)
4. [Results by selecting from the city_country view:](#)

Views That Allow UPDATE Statements

1. [Creating Views That Can Be Used With an UPDATE Statement](#)

Clustered vs. Non-clustered Indexes

1. [Clustered vs. Non-clustered indexes](#)

Aggregate Functions

1. [SUM\(\[DISTinCT\] column_values\)](#)
2. [Min\(\[DISTinCT\] column_values\)](#)
3. [... average of the non-null columns in the expression](#)
4. [... AVG\(list_price\), SUM\(list_price\), Min\(list_price\), MAX\(list_price\),...](#)
5. [MAX\(\[DISTinCT\] column_values\)](#)
6. [COUNT\(\[DISTinCT\] column_values\)](#)
7. [... value of the non-null columns in the expression](#)
8. [numeric, date, string](#)
9. [AVG\(\[DISTinCT\] column_values\)](#)
10. [... total of the non-null columns in the expression](#)
11. [numeric, date, string](#)
12. [... value off the non-null columns in the expression](#)
13. [... number of the non-null columns in the expression](#)

The Subquery In a SELECT Statement

1. [The Subquery **in** a SELECT Statement](#)
2. [WHERE CountryCode **in**](#)
3. [... city 4 WHERE CountryCode **in** 5 \(SELECT code 6 ...](#)

Introduction

1. [Before You Begin](#)
2. [... databases that you can download and **install in** your local MySQL **instance**....](#)
3. [in a future edition, this book will **include** SQL design basics and guidance...](#)

How to Retrieve Data From a Single Table

1. [The **in** Keyword](#)
2. [DIST**in**CT Keyword](#)
3. [The clauses MUST appear **in** the order shown above.](#)
4. [Matches any **single** character with**in** the given range.](#)
5. [... world;SELECT nameFROM countryWHERE name **in** \('Aruba', 'Barbados', 'Cuba',...](#)
6. [DIST**in**CT](#)
7. [Matches any **single** character listed with**in** the brackets.](#)
8. [Match the pattern to the **beginning** of the value **being** tested.](#)
9. [Let us break the statement **line** by **line**:](#)
10. [... but b must NOT be present to be **included**](#)
11. [Elim**in**ates duplicate rows](#)
12. [Match any **string** of characters to the left of the symbol](#)
13. [SELECT name, **indep**YearFROM countryWHERE **indep**Year IS NULL;](#)
14. [Separates two **string** patterns and matches either one](#)
15. [Matches any **single** character.](#)
16. [USE world;SELECT name, **indep**YearFROM countryWHERE name BETWEEN "Aruba" and...](#)
17. [... pattern to the end of the value **being** tested.](#)
18. [integer Division](#)
19. [in the previous example, we created a new column that was a calculated value....](#)
20. [Table 7. DIST**in**CT Keyword](#)
21. [... a and b are present, item is **included**](#)
22. [SELECT DIST**in**CT **continent**, nameFROM countryORDER BY **continent**;](#)
23. [Match a **single** character](#)
24. [... either a OR b is present item is **included**](#)
25. [Modulo \(remain**der**\)](#)
26. [We used the AS keyword then **in** quotes we put the new column alias of "People...](#)

SQL Indexes Explained

1. [When to Create an **index**](#)
2. [SQL **indexes**](#)

The Five Clauses of the SELECT Statement

1. [The clauses MUST appear **in** the order shown above.](#)
2. [Let us break the statement **line** by **line**:](#)

LIKE and REGEXP Operators

1. [Matches any **single** character with**in** the given range.](#)
2. [Matches any **single** character listed with**in** the brackets.](#)
3. [Match the pattern to the **beginning** of the value **being** tested.](#)
4. [Match any **string** of characters to the left of the symbol](#)
5. [Separates two **string** patterns and matches either one](#)
6. [Matches any **single** character.](#)
7. [... pattern to the end of the value **being** tested.](#)
8. [Match a **single** character](#)

Arithmetic Operators

1. [integer Division](#)
2. [Modulo \(remainder\).](#)

Column Aliases

1. [in the previous example, we created a new column that was a calculated value....](#)
2. [We used the AS keyword then **in** quotes we put the new column alias of "People..."](#)

IS NULL, BETWEEN, IN Operators

1. [The **in** Keyword](#)
2. [... world;SELECT nameFROM countryWHERE name **in** \('Aruba', 'Barbados', 'Cuba',...](#)
3. [SELECT name, **indepYear**FROM countryWHERE **indepYear** IS NULL;](#)
4. [USE world;SELECT name, **indepYear**FROM countryWHERE name BETWEEN "Aruba" and...](#)

AND, OR, NOT Logical Operators

1. [... but b must NOT be present to be **included**](#)
2. [... a and b are present, item is **included**](#)
3. [... either a OR b is present item is **included**](#)

DISTINCT Clause

1. [DIST**in**CT Keyword](#)
2. [DIST**in**CT](#)
3. [Eliminates duplicate rows](#)
4. [Table 7. DIST**in**CT Keyword](#)
5. [SELECT DIST**in**CT continent, nameFROM countryORDER BY continent;](#)

The JOIN Clause

1. [The Join Clause](#)
2. [... write SQL statements more succ**in**ctly with an **inner join** clause using table...](#)
3. [JO**in** country co](#)
4. [Let us break the statement **line** by **line**:](#)
5. [... example of a SQL statement with an **inner join** clause using explicit syntax.](#)
6. [... FROM country 6 JO**in** city 5 ON city.CountryCode...](#)
7. [The results of the **join** query would yield the same results as shown below...](#)
8. [... FROM city ci 5 JO**in** country co 6 ON ci.CountryCode...](#)

Joining More Than Two Tables

1. [How to Join More than Two Tables](#)
2. [JOIN countrylanguage cl.](#)
3. [... FROM city ci6 JOIN country co 7 ON ci.CountryCode...](#)

The OUTER JOIN Clause

1. [The Outer Join Clause](#)
2. [... SQL statement with an outer join clause.](#)
3. [SELECT c.name, c.continent, cl.language](#)
4. [FROM country c LEFT JOIN countrylanguage cl](#)
5. [... world;2 SELECT c.name, c.continent, cl.language3 FROM country c LEFT JOIN...](#)

How to Code a UNION

1. [... populationFROM countryWHERE continent = 'Oceania'](#)
2. [... population6 FROM country7 WHERE continent = 'Oceania'8 ORDER BY name;_](#)

Date Functions

1. [• Dates must be enclosed in quotes • You can pass a DATE or DATETIME...](#)
2. [* extracts the date from input. If time is included, the time is dropped.](#)
3. [USE world;SELECT name, continent, DATE_FORMAT\('2020-01-28', '%m/%d/%y'\)FROM...](#)
4. [Minutes, numeric \(00..59\).](#)
5. [... order_date, DATE_ADD\(order_date, INTERVAL 1 DAY\) AS 'ORDER DATE PLUS...](#)
6. [DATE_ADD\(date, interval expression unit\).](#)
7. [DATE_ADD\('2020-01-01', INTERVAL 1 DAY\)](#)
8. [... DATETIME value equal to the original value plus the specified interval.](#)

Numeric Functions

1. [FLOOR, CEILING, TRUNCATE](#)
2. [... list_price, FLOOR\(list_price\), CEILING\(list_price\), TRUNCATE\(list_price,...](#)
3. [CEILING\(number\).](#)
4. [CEILING\(6.2\).](#)
5. [Table 6. FLOOR, CEILING, TRUNCATE functions](#)

String Functions

1. [LOCATE, LENGTH, SUBSTRinG](#)
2. [string](#)
3. [SUBSTRinG\(str,start\[,length\]\)](#)
4. [string](#)
5. [string](#)
6. [LEFT\(string, num. characters\)](#)
7. [TRIM\(string\)](#)
8. [string](#)
9. [... world;SELECT CONCAT\(name,',','continent'\)FROM country;](#)
10. [string](#)
11. [LTRIM\(string\)](#)
12. [string](#)
13. [string](#)
14. [RIGHT\(string, num. characters\)](#)
15. [... LENGTH\('salmon'\), __ SUBSTRinG\('salmon',3,999\);](#)
16. [Table 9. LOCATE, LENGTH, SUBSTRinG functions](#)
17. [LOCATE\(find,search\[,start\]\)](#)
18. [string](#)
19. [LOWER\(string\)](#)
20. [SUBSTRinG\('salmon',3,999\)](#)
21. [string](#)
22. [string](#)
23. [LOCATE\(\), and LENGTH\(\) accept a string but return an integer. • SUBSTRinG\(\)...](#)
24. [RTRIM\(string\)](#)
25. [string](#)
26. [UPPER\(string\)](#)

index

SQL Indexes Explained

1. [When to Create an index](#)
2. [SQL indexes](#)

Clustered vs. Non-clustered Indexes

1. [Clustered vs. Non-clustered indexes](#)

indexes

Clustered vs. Non-clustered Indexes

1. [Clustered vs. Non-clustered indexes](#)

SQL Indexes Explained

1. [SQL indexes](#)

insert

The INSERT Clause With a Column List

1. [The **insert** Clause With a Column List](#)
2. [Below is a basic example of an **insert** statement with a column list:](#)
3. [**insert** INTO city](#)
4. [Results of the **insert**:](#)
5. [1 USE world;2 **insert** INTO city 3 \(name, countryCode, district,...](#)

The INSERT Clause Without a Column List

1. [The **insert** Clause Without a Column List](#)
2. [1 USE world;2 **insert** INTO city 3 VALUES 4 \(DEFAULT,...](#)

is null

How to Retrieve Data From a Single Table

1. [is null](#)
2. [... IndepYearFROM countryWHERE IndepYear **is null**;](#)

IS NULL, BETWEEN, IN Operators

1. [is null](#)
2. [... IndepYearFROM countryWHERE IndepYear **is null**;](#)

join

The JOIN Clause

1. [The **join** Clause](#)
2. [... more succinctly with an inner **join** clause using table aliases. Instead of...](#)
3. [join country co](#)
4. [... a SQL statement with an inner **join** clause using explicit syntax.](#)
5. [... FROM country 6 join city 5 ON city.CountryCode...](#)
6. [The results of the **join** query would yield the same results as shown below whether...](#)
7. [... FROM city ci 5 join country co 6 ON ci.CountryCode...](#)

Joining More Than Two Tables

1. [How to **join** More than Two Tables](#)
2. [join countrylanguage cl](#)
3. [... FROM city ci6 join country co 7 ON ci.CountryCode...](#)

The OUTER JOIN Clause

1. [The Outer **join** Clause](#)
2. [... a SQL statement with an outer **join** clause.](#)
3. [FROM country c LEFT join countrylanguage cl](#)
4. [... cl.language3 FROM country c LEFT join countrylanguage cl4 ON c.code = cl.CountryCode5...](#)

Improving the GROUP BY Query

1. [... List Price'FROM product p **join** category c ON p.category_id = c.category_idGROUP...](#)
2. [...**join** category c](#)

Benefits of Using Views

1. [... country_nameFROM city ci **join** country co ON ci.CountryCode = co.Code;](#)
2. [...**join** country co](#)

left

String Functions

1. [... bike;SELECT category_name, **left**\(category_name,8\) AS 'First 8 Characters',...](#)
2. [...**left**\('Salmon',3\).](#)

How to Retrieve Data From a Single Table

1. [... any string of characters to the **left** of the symbol](#)

LIKE and REGEXP Operators

1. [... any string of characters to the **left** of the symbol](#)

The OUTER JOIN Clause

1. [FROM country c **left** JOIN countrylanguage cl](#)
2. [... c.continent, cl.language3 FROM country c **left** JOIN countrylanguage cl4 ON...](#)

String Functions

1. [RIGHT, **left**](#)
2. [...**left**\(string, num. characters\)](#)
3. [...**left**\('Salmon_'\)](#)
4. [SELECT LTRIM\(' Salmon_'\) AS "**left** Trim", RTRIM\(' Salmon_'\) AS...](#)
5. [Table 7. RIGHT, **left** functions](#)

like

How to Retrieve Data From a Single Table

1. [... **like** and REGEXP Operators](#)
2. [... **like** Symbol](#)
3. [Table 2. **like** Keyword](#)
4. [... world;SELECT nameFROM countryWHERE name **like** 'A%'](#)

LIKE and REGEXP Operators

1. [... **like** and REGEXP Operators](#)
2. [... **like** Symbol](#)
3. [Table 2. **like** Keyword](#)
4. [... world;SELECT nameFROM countryWHERE name **like** 'A%'](#)

limit

How to Retrieve Data From a Single Table

1. [limit 5;](#)

The Five Clauses of the SELECT Statement

1. [... "AFG"5 ORDER BY name6 limit 3](#)
2. [limit 5;](#)

The Subquery In a SELECT Statement

1. [... ORDER BY population 9 limit 5;](#)

logical operators

How to Retrieve Data From a Single Table

1. [AND, OR, NOT logical operators](#)
2. [Table 6. logical operators](#)

AND, OR, NOT Logical Operators

1. [AND, OR, NOT logical operators](#)
2. [Table 6. logical operators](#)

ltrim

String Functions

1. [TRIM, ltrim, RTRIM](#)
2. [ltrim\(string\)](#)
3. [SELECT ltrim\(' Salmon '\) AS "Left Trim", RTRIM\(' Salmon '\) AS "Right..."](#)

min

Aggregate Functions

1. [min\(\[DISTINCT\] column_values\)](#)
2. [... AVG\(list_price\), SUM\(list_price\), min\(list_price\), MAX\(list_price\),...](#)

How to Retrieve Data From a Single Table

1. [Eliminates duplicate rows](#)

DISTINCT Clause

1. [Eliminates duplicate rows](#)

Date Functions

1. [minutes, numeric\(00..59\)](#)

now

How to Retrieve Data From a Single Table

1. [... is that the column header is **now** population / SurfaceArea. However we can...](#)

Column Aliases

1. [... is that the column header is **now** population / SurfaceArea. However, we can...](#)

Date Functions

1. [now\(\)](#).
2. [SELECT now\(\) AS 'now\(\)', DATE\('2020-01-01'\) AS 'DATE\(\)', date only',...](#)
3. [now\(\)](#).

null

How to Retrieve Data From a Single Table

1. [IS null](#)
2. [... IndepYearFROM countryWHERE IndepYear IS null;](#)

IS NULL, BETWEEN, IN Operators

1. [IS null](#)
2. [... IndepYearFROM countryWHERE IndepYear IS null;](#)

Aggregate Functions

1. [The average of the non-**null** columns in the expression](#)
2. [The highest value of the non-**null** columns in the expression](#)
3. [The total of the non-**null** columns in the expression](#)
4. [The lowest value off the non-**null** columns in the expression](#)
5. [The number of the non-**null** columns in the expression](#)

or not

The JOIN Clause

1. [... results as shown below whether **or not** table names are completely written out...](#)

order by

How to Retrieve Data From a Single Table

1. [... 'Barbados', 'Cuba', 'Bahamas'\)**order by** population ASC;](#)
2. [order by name](#)
3. [... 'caribbean'AND population > 100000**order by** population ASC;](#)
4. [... DISTINCT continent, nameFROM country**order by** continent;](#)

The Five Clauses of the SELECT Statement

1. [... CountryCode = "AFG"5 **order by** name6 LIMIT 3](#)
2. [order by name](#)

IS NULL, BETWEEN, IN Operators

1. [... 'Barbados', 'Cuba', 'Bahamas'\)order by population ASC;](#)

AND, OR, NOT Logical Operators

1. [... 'caribbean'AND population > 100000order by population ASC;](#)

DISTINCT Clause

1. [... DISTINCT continent, nameFROM countryorder by continent;](#)

The OUTER JOIN Clause

1. [... cl4 ON c.code = cl.CountryCode5 **order by** cl.language ASC;](#)

How to Code a UNION

1. [order by name;](#)
2. [... WHERE continent = 'Oceania'8 **order by** name;](#)

Improving the GROUP BY Query

1. [order by category_name;](#)
2. [... c.category_idGROUP BY category_nameorder by category_name;](#)

The Subquery In a SELECT Statement

1. [... region = 'Caribbean'\) 8 **order by** population 9 LIMIT 5;](#)

outer join

The OUTER JOIN Clause

1. [The **outer join** Clause](#)
2. [... snippet of a SQL statement with an **outer join** clause.](#)

regexp

How to Retrieve Data From a Single Table

1. [LIKE and **regexp** Operators](#)
2. [... world;SELECT nameFROM countryWHERE name **regexp** 'g\[o,u\]';](#)
3. [regexp Characters](#)

LIKE and REGEXP Operators

1. [LIKE and **regexp** Operators](#)
2. [... world;SELECT nameFROM countryWHERE name **regexp** 'g\[o,u\]';](#)
3. [regexp Characters](#)

right

String Functions

1. [right, LEFT](#)
2. [right\(string, num. characters\)](#)
3. [right\('Salmon', 3\)](#)
4. [right\(' Salmon'\)](#)
5. [... RTRIM\(' Salmon '\) AS "right Trim", TRIM\(' Salmon '\) AS "Trim";](#)
6. [Table 7. right, LEFT functions](#)
7. [... AS 'First 8 Characters', right\(category_name, 8\) AS 'Last 8 Characters'FROM...](#)

round

Improving the GROUP BY Query

1. [... category_name, CONCAT\('\\$', round\(AVG\(list_price\),2\)\) AS 'Average List...](#)
2. [... CONCAT\('\\$', round\(AVG\(list_price\),2\)\) AS 'Average List Price'](#)

Numeric Functions

1. [round](#)
2. [Table 5. round function](#)
3. [round\(13.37, 1\)](#)
4. [... world;SELECT name, LifeExpectancy, round\(LifeExpectancy\) FROM world.country;](#)
5. [round\(number\[, length\]\)](#)

rtrim

String Functions

1. [TRIM, LTRIM, rtrim](#)
2. [... Salmon '\) AS "Left Trim", rtrim\(' Salmon '\) AS "Right Trim", ...](#)
3. [rtrim\(string\)](#)

select

String Functions

1. [select UPPER\('Salmon'\), LOWER\('Salmon'\);](#)
2. [USE world;select CONCAT\(name, '_', continent\)FROM country;](#)
3. [select FORMAT\(list_price,2\) FROM bike.product;](#)
4. [select LOCATE\('al',salmon,1\), LENGTH\('salmon'\), SUBSTRING\('salmon',3,999\);](#)
5. [select LTRIM\(' Salmon '\) AS "Left Trim", RTRIM\(' Salmon '\) AS "Right...](#)
6. [USE bike;select category_name, LEFT\(category_name, 8\) AS 'First 8 Characters', ...](#)

Simple GROUP BY Query

1. [USE bike;select category_id, AVG\(list_price\)FROM productGROUP BY category_id](#)
2. [select category_id, AVG\(list_price\);](#)

Improving the GROUP BY Query

1. [USE bike;select category_name, CONCAT\('\\$', ROUND\(AVG\(list_price\),2\)\) AS...](#)
2. [select category_name,](#)

Using the HAVING Clause

1. [USE bike;select category_id, AVG\(list_price\)FROM productGROUP BY category_idHAVING...](#)

Using the HAVING and WHERE Clauses Together

1. [USE bike;select category_id, AVG\(list_price\)FROM productWHERE model_year = 2016GROUP...](#)

COUNT(column_name) and COUNT(*)

1. [USE bike;select COUNT\(phone\), COUNT\(*\) FROM CUSTOMER](#)

Using the DISTINCT Statement

1. [ExampleUSE bike;select COUNT\(list_price\), COUNT\(DISTINCT list_price\) FROM product;](#)

The Subquery in an UPDATE statement

1. [\(select CountryCode FROM countrylanguage WHERE population = 0\).](#)
2. [... 0.00 3 WHERE Code IN 4 \(select CountryCode FROM countrylanguage...](#)

Create a Duplicate Table From An Existing Table

1. [... from an Existing Table with a select Statement](#)
2. [CREATE TABLE city_bak AS select * FROM city;](#)
3. [... CREATE TABLE city_bak AS select * FROM city;](#)

The Subquery In a Delete Statement

1. [\(select code FROM country](#)
2. [... city_bakWHERE CountryCode IN \(select code FROM country WHERE...](#)

Benefits of Using Views

1. [... WORLD;CREATE VIEW city_country ASselect ci.name AS city_name, co.name AS country_nameFROM...](#)
2. [select ci.name AS city_name, co.name AS country_name](#)
3. [Results by selecting from the city_country view:](#)

Aggregate Functions

1. [USE bike;select AVG\(list_price\), SUM\(list_price\), MIN\(list_price\), MAX\(list_price\),...](#)

The Subquery In a SELECT Statement

1. [The Subquery in a select Statement](#)
2. [1 USE world;2 select name, population 3 FROM city 4 WHERE...](#)
3. [select name, population](#)
4. [\(select code](#)

How to Retrieve Data From a Single Table

1. [The Five Clauses of the **select** statement](#)
2. [USE world;**select** nameFROM countryWHERE name IN \('Aruba','Barbados','Cuba',...](#)
3. [select name, population / SurfaceArea AS "People per square mile"FROM...](#)
4. [select name, IndepYearFROM countryWHERE IndepYear IS NULL;](#)
5. [USE world;**select** nameFROM countryWHERE name REGEXP 'g\[o,u\]';](#)
6. [USE world;**select** name, populationFROM countryWHERE population > 1000000;](#)
7. [USE world;**select** name, populationFROM countryWHERE region = 'caribbean'AND population...](#)
8. [USE world;**select** name, population / SurfaceAreaAS "People per square mile"FROM...](#)
9. [USE world;**select** name, IndepYearFROM countryWHERE name BETWEEN "Aruba" and "Bahamas";](#)
10. [select name](#)
11. [select DISTINCT continent, nameFROM countryORDER BY continent;](#)
12. [USE world;**select** nameFROM countryWHERE name LIKE 'A%'](#)

The Five Clauses of the SELECT Statement

1. [The Five Clauses of the **select** statement](#)
2. [... Example:1 USE world;2 select name3 FROM city4 WHERE CountryCode...](#)
3. [select name](#)

LIKE and REGEXP Operators

1. [USE world;**select** nameFROM countryWHERE name REGEXP 'g\[o,u\]';](#)
2. [USE world;**select** nameFROM countryWHERE name LIKE 'A%'](#)

Arithmetic Operators

1. [USE world;**select** name, population / SurfaceAreaAS "People per square mile"FROM...](#)

Column Aliases

1. [select name, population / SurfaceArea AS "People per square mile"FROM...](#)

Comparison Operators

1. [USE world;**select** name, populationFROM countryWHERE population > 1000000;](#)

IS NULL, BETWEEN, IN Operators

1. [USE world;**select** nameFROM countryWHERE name IN \('Aruba','Barbados','Cuba',...](#)
2. [select name, IndepYearFROM countryWHERE IndepYear IS NULL;](#)
3. [USE world;**select** name, IndepYearFROM countryWHERE name BETWEEN "Aruba" and "Bahamas";](#)

AND, OR, NOT Logical Operators

1. [USE world;**select** name, populationFROM countryWHERE region = 'caribbean'AND population...](#)

DISTINCT Clause

1. [select DISTINCT continent, nameFROM countryORDER BY continent;](#)

The JOIN Clause

1. [select ci.name AS "City Name", co.name AS "Country Name"](#)
2. [1 USE world;2 select city.name AS "City Name", 3 country.name...](#)
3. [... clause and referenced in the select and ON clause:](#)
4. [1 USE world;2 select ci.name AS "City Name", 3 co.name...](#)

Joining More Than Two Tables

1. [1 USE world;2 select ci.name AS "City Name",3 co.name...](#)

The OUTER JOIN Clause

1. [select c.name, c.continent, cl.language](#)
2. [1 USE world;2 select c.name, c.continent, cl.language3 FROM country c LEFT JOIN...](#)

How to Code a UNION

1. [select name, populationFROM countryWHERE continent = 'Oceania'](#)
2. [select name, populationFROM cityWHERE CountryCode = 'AUS'](#)
3. [1 USE world;2 select name, population3 FROM city WHERE CountryCode = 'AUS'4 UNION5...](#)

Date Functions

1. [select NOW\(\) AS 'NOW\(\)', DATE\('2020-01-01'\) AS 'DATE\(\)', date only', CURRENT_DATE...](#)
2. [select DATEDIFF\('2018-01-01', '2019-01-01'\) AS 'Date Difference';](#)
3. [USE world;select name, continent, DATE_FORMAT\('2020-01-28', '%m/%d/%y'\)FROM country;](#)
4. [USE bike;select order_date, DATE_ADD\(order_date, INTERVAL 1 DAY\) AS 'ORDER...](#)

Numeric Functions

1. [USE bike;select list_price, FLOOR\(list_price\), CEILING\(list_price\), TRUNCATE\(list_price,...](#)
2. [USE world;select name, LifeExpectancy, ROUND\(LifeExpectancy\) FROM world.country;](#)

sql indexes

SQL Indexes Explained

1. [sql indexes](#)

sql view

SQL View Explained

1. [sql views](#)

sql views

SQL View Explained

1. [sql views](#)

subquery

The Subquery in an UPDATE statement

1. [The **subquery** in an UPDATE statement](#)

The Subquery In a Delete Statement

1. [The **subquery** in a DELETE statement](#)

The Subquery In a SELECT Statement

1. [The **subquery** in a SELECT Statement](#)

sum

Aggregate Functions

1. [sum\(\[DISTINCT\] column_values\)](#)
2. [... bike;SELECT AVG\(list_price\), **sum**\(list_price\), MIN\(list_price\), MAX\(list_price\),...](#)

trim

String Functions

1. [trim, Ltrim, Rtrim](#)
2. [trim\(string\)](#)
3. [trim\(' _ Salmon _'\)](#)
4. [Ltrim\(string\)](#)
5. [Table 8. trim functions](#)
6. [SELECT Ltrim\(' _ Salmon _'\) AS "Left trim", Rtrim\(' _ Salmon _'\) AS...](#)
7. [Rtrim\(string\)](#)

truncate

Numeric Functions

1. [FLOOR, CEILING, truncate](#)
2. [truncate\(7.9\)](#)
3. [... FLOOR\(list_price\), CEILING\(list_price\), truncate\(list_price, 0\)FROM product;](#)
4. [Table 6. FLOOR, CEILING, truncate functions](#)
5. [truncate\(NUMBER, length\)](#)

union

How to Code a UNION

1. [How to Code a **union**](#)
2. [union](#)
3. [... city WHERE CountryCode = 'AUS'4 **union**5 SELECT name,,population6 FROM country7...](#)

update

The UPDATE Clause With a Column List

1. [The **update** Clause](#)
2. [**update** city](#)
3. [1 USE world; 2 **update** city 3 SET Population = 65000, district...](#)

The Subquery in an UPDATE statement

1. [The Subquery in an **update** statement](#)
2. [1 **update** country 2 SET GNPOld = 0.00 3 WHERE Code IN 4 \(SELECT...](#)
3. [**update** country](#)

The Subquery In a Delete Statement

1. [... Before you can run a DELETE or **update** statement without a WHERE clause, you...](#)

Views That Allow UPDATE Statements

1. [... Views That Can Be Used With an **update** Statement](#)

utc_date

Date Functions

1. [... CURRENT_TIME AS 'CURRENT_TIME', **utc_date** AS '**utc_date**', UTC_TIME AS...](#)
2. [**utc_date**\(\)](#)
3. [**utc_date**](#)

utc_time

Date Functions

1. [**utc_time**](#)
2. [**utc_time**\(\)](#)
3. [... UTC_DATE AS 'UTC_DATE', **utc_time** AS '**utc_time**';](#)

view

Benefits of Using Views

1. [Benefits of Using **views**](#)
2. [USE WORLD; CREATE **view** city_country AS SELECT ci.name AS city_name, co.name AS...](#)
3. [CREATE **view** city_country AS](#)
4. [... selecting from the city_country **view**;](#)

Views That Allow UPDATE Statements

1. [Creating **views** That Can Be Used With an UPDATE Statement](#)

SQL View Explained

1. [SQL **views**](#)

views

Benefits of Using Views

1. [Benefits of Using **views**](#)

Views That Allow UPDATE Statements

1. [Creating **views** That Can Be Used With an UPDATE Statement](#)

SQL View Explained

1. [SQL **views**](#)

where

The DELETE Clause

1. [... DELETE 3 FROM city 4 **where** name = 'san felipe' AND countrycode...](#)
2. [where name = 'san felipe' AND countrycode = 'chl';](#)

Grouping Data

1. [Filtering With **where** And HAVING](#)

Using the HAVING and WHERE Clauses Together

1. [where model_year = 2016](#)
2. [... category_id, AVG\(list_price\)FROM productwhere model_year = 2016GROUP BY category_idHAVING...](#)
3. [... includes both the HAVING and **where** clause in the same SQL statement.](#)

The Subquery in an UPDATE statement

1. [... CountryCode FROM countrylanguage **where** population = 0\)](#)
2. [... SET GNPOld = 0.00 3 **where** Code IN 4 \(SELECT CountryCode FROM...](#)
3. [where Code IN](#)

The Subquery In a Delete Statement

1. [USE world;DELETE FROM city_bakwhere CountryCode IN \(SELECT code FROM country...](#)
2. [where region = 'Central Africa'\);](#)
3. [where CountryCode IN](#)
4. [... or UPDATE statement without a **where** clause, you must uncheck "Safe Updates"...](#)

The Subquery In a SELECT Statement

1. [where CountryCode IN](#)
2. [... population 3 FROM city 4 **where** CountryCode IN 5 \(SELECT...](#)
3. [where region = 'Caribbean'\)](#)

How to Retrieve Data From a Single Table

1. [... world;SELECT nameFROM countrywhere name IN \('Aruba', 'Barbados', 'Cuba', 'Bahamas'\)ORDER...](#)
2. [SELECT name, IndepYearFROM countrywhere IndepYear IS NULL;](#)
3. [... world;SELECT nameFROM countrywhere name REGEXP 'g\[o,u\]';](#)
4. [... world;SELECT name, populationFROM countrywhere population > 1000000;](#)
5. [... world;SELECT name, populationFROM countrywhere region = 'caribbean'AND population...](#)
6. [... world;SELECT name, IndepYearFROM countrywhere name BETWEEN "Aruba" and "Bahamas";](#)
7. [... world;SELECT nameFROM countrywhere name LIKE 'A%'](#)

The Five Clauses of the SELECT Statement

1. [... SELECT name3 FROM city4 where CountryCode = "AFG"5 ORDER...](#)

LIKE and REGEXP Operators

1. [... world;SELECT nameFROM countrywhere name REGEXP 'g\[o,u\]';](#)
2. [... world;SELECT nameFROM countrywhere name LIKE 'A%'](#)

Comparison Operators

1. [... world;SELECT name, populationFROM countrywhere population > 1000000;](#)

IS NULL, BETWEEN, IN Operators

1. [... world;SELECT nameFROM countrywhere name IN \('Aruba', 'Barbados', 'Cuba', 'Bahamas'\)ORDER...](#)
2. [SELECT name, IndepYearFROM countrywhere IndepYear IS NULL;](#)
3. [... world;SELECT name, IndepYearFROM countrywhere name BETWEEN "Aruba" and "Bahamas";](#)

AND, OR, NOT Logical Operators

1. [... world;SELECT name, populationFROM countrywhere region = 'caribbean'AND population...](#)

How to Code a UNION

1. [SELECT name, populationFROM countrywhere continent = 'Oceania'](#)
2. [SELECT name, populationFROM citywhere CountryCode = 'AUS'](#)
3. [... SELECT name, population3 FROM city where CountryCode = 'AUS'4 UNION5 SELECT...](#)

Date Functions

1. [Year for the week where Sunday is the first day of the week, numeric, four digits;...](#)
2. [Week \(01..53\), where Sunday is the first day of the week; WEEK\(\) mode 2; used...](#)
3. [Week \(00..53\), where Sunday is the first day of the week; WEEK\(\) mode 0](#)
4. [Week \(00..53\), where Monday is the first day of the week; WEEK\(\) mode 1](#)
5. [Week \(01..53\), where Monday is the first day of the week; WEEK\(\) mode 3; used...](#)
6. [Year for the week, where Monday is the first day of the week, numeric, four...](#)





This content is provided to you freely by BYU-I Books.

Access it online or download it at https://books.byui.edu/learning_mysql/index.