
Before we start...

This is the **Introduction to Databases
Queries and Analytics** workshop

Download material: dartgo.org/db-query

More info: rc.dartmouth.edu



DARTMOUTH

Information, Technology and Consulting

Hands-on Introduction to MySQL Queries and Analytics

Christian Darabos, PhD

Material download: dartgo.org/db-query

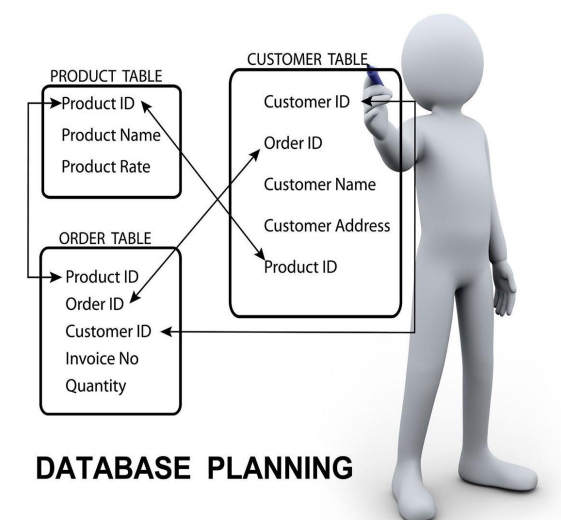
<http://www.mysqltutorial.org/>

Overview

- Introduction to Databases and this workshop
- Development vs. production environments
- tools (admin, browse, query, etc.)
- Queries and Analytics in SQL — case study
- Concluding remarks

Definition of a Relational Database (SQL)

- a database type structured to recognize relations among stored items of information
- designed to store text, dates/times, integers, floating-point number
- implemented as a series of tables



SQL vs. NoSQL

SQL	NoSQL
Relational Databases	distributed database
table based	document based, key-value pairs, graph databases or wide-column stores
predefined schema	dynamic schema for unstructured data
vertically scalable (more powerful hardware)	horizontally scalable (more hardware)
SQL (structured query language)	proprietary language
MySql, Oracle, Sqlite, Postgres, MariaDB, ...	MongoDB, BigTable, Redis, RavenDb, Cassandra, ...

Why use a Relational Database

- concurrent (simultaneous) read and **write**
 - **powerful selecting, filtering and sorting**
cross-referencing tables
 - large quantity of structured storage and standardized distribution
 - minimize post-processing (simple analytics tools pre-implemented)
 - automate using any scripting and programming languages (R, Matlab, Python, C++, Java, PHP)
 - web-proof
-

Development Environment

- **MAMP / MAMP Pro**
- Windows & Mac OS X
- Web servers: Apache, Ngnix
- Data Base server: MySQL
- Web/DB admin tools: phpMyAdmin, phpLiteAdmin
- Scripting: PHP, Python, Perl
- User friendly: “Web Start”



Production Environment

- Research Computing
 - Institutional Information Systems & Services
 - Cloud services (AWS, Azure, Google Cloud, etc.)
 - Google: free database hosting :)
-

Accessing the DB

First, navigate to:

<http://dartgo.org/pma>

- Log in via DUO/SSO

Then:

- username: **workshop**
password: **learndb**
-

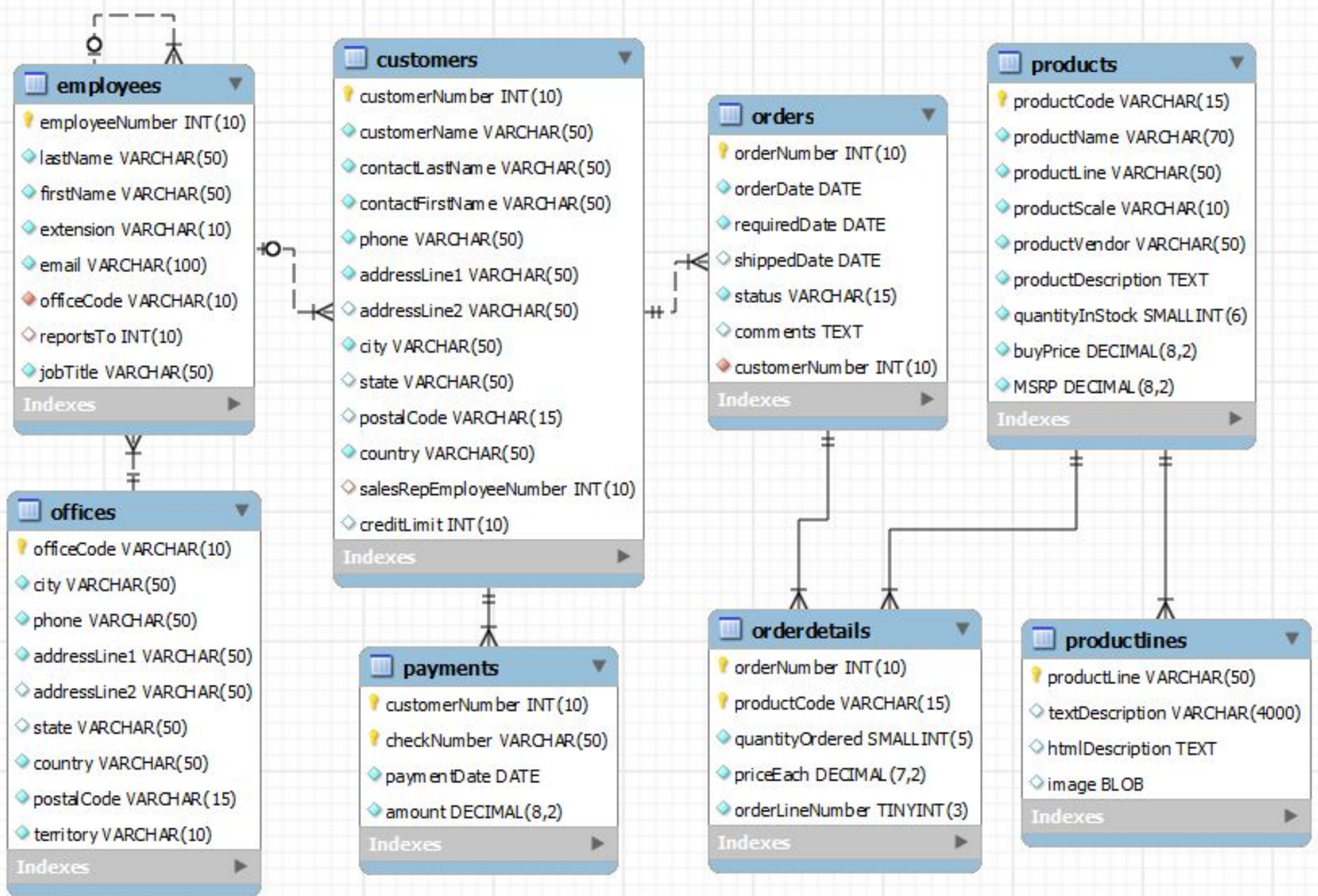
What is a QUERY?

- A query is a question:
 - How many clients are named Paul?
 - What is the sales peoples' average sales sum in December?
 - The answer is given in the form of a table
-

Case Study

- A toy classic car company keeps track of:
 - Employees, Offices and Customers
 - Orders and Payment methods
 - Products and Product Lines
-

Case Study DB



Case Study DB

Server: localhost:8889 » Database: retail_example

Structure SQL Search Query Export Import Operations Privileges Routines Events Triggers Designer

	Table ▲	Action							Rows ?	Type	Collation	Size	Overhead
<input type="checkbox"/>	customers	★	Browse	Structure	Search	Insert	Empty	Drop	122	InnoDB	latin1_swedish_ci	32 KiB	-
<input type="checkbox"/>	employees	★	Browse	Structure	Search	Insert	Empty	Drop	23	InnoDB	latin1_swedish_ci	48 KiB	-
<input type="checkbox"/>	offices	★	Browse	Structure	Search	Insert	Empty	Drop	7	InnoDB	latin1_swedish_ci	16 KiB	-
<input type="checkbox"/>	orderdetails	★	Browse	Structure	Search	Insert	Empty	Drop	2,996	InnoDB	latin1_swedish_ci	240 KiB	-
<input type="checkbox"/>	orders	★	Browse	Structure	Search	Insert	Empty	Drop	326	InnoDB	latin1_swedish_ci	64 KiB	-
<input type="checkbox"/>	payments	★	Browse	Structure	Search	Insert	Empty	Drop	273	InnoDB	latin1_swedish_ci	16 KiB	-
<input type="checkbox"/>	productlines	★	Browse	Structure	Search	Insert	Empty	Drop	7	InnoDB	latin1_swedish_ci	16 KiB	-
<input type="checkbox"/>	products	★	Browse	Structure	Search	Insert	Empty	Drop	110	InnoDB	latin1_swedish_ci	80 KiB	-
8 tables		Sum							3,864	InnoDB	latin1_swedish_ci	512 KiB	0 B

⬆ ☐ Check All With selected: ▼

- browse content, edit structure, search, insert, empty or drop
- feel free to click around and explore the UI

Browsing the Content

Using the UI:

- select the “classic_model_cars” database
- select the “employee” table
- click on the “Browse” tab

```
SELECT * FROM `employees`
```

UI Search Form

Using the UI's "Search" tab, search for:

- all employees with last name "Firrelli"
 - all employees whose first name is NOT "Leslie"
 - all employees whose job title contains "sale"
-

UI Search Form

Using the UI's "Search" tab, search for:

- all employees with last name "Firrelli"

```
SELECT * FROM `employees` WHERE `lastName` LIKE 'firrelli'
```

- all employees whose first name is NOT "Leslie"

```
SELECT * FROM `employees` WHERE `firstName` NOT LIKE 'leslie'
```

- all employees whose job title contains "sale"

```
SELECT * FROM `employees` WHERE `jobTitle` LIKE '%sale%'
```

Search Operators

Operator	
✓ =	
>	
>=	
<	
<=	
!=	
LIKE	
LIKE %...%	
NOT LIKE	
IN (...)	
NOT IN (...)	
BETWEEN	
NOT BETWEEN	
IS NULL	
IS NOT NULL	

= : equals

> : greater than

>= : greater than or equals

< : smaller than

<= : smaller than or equals

!= : not equals

LIKE / NOT LIKE: case insensitive comparison

LIKE %...% : contains

IN / NOT IN (...) : equals one of the values in (...)

BETWEEN / NOT BETWEEN: between 2 values

IS NULL / IS NOT NULL : value is "NULL"

UI Limitations

The search form in the UI is limited:

- one table at a time
 - one value at a time
 - no arithmetic
 - no grouping
-

SQL Query in UI

- at the database level
 - select the database
 - SQL (or Query tab for more advanced users)
 - at the table level
 - select the database
 - select the table
 - SQL tab
 - hit “SELECT *” button, if table name is missing
-

SELECT statement

- The SELECT statement controls which columns and rows that you want to see of the tables specified in the FROM section of the statement.
- The result(s) of a SELECT statement is always a table
- SELECT * shows ALL the columns

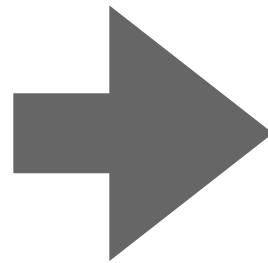
```
1 SELECT
2     lastname, firstname, jobtitle
3 FROM
4     employees;
```

```
1 SELECT * FROM employees;
```

Eliminate Duplicates

In order to remove these duplicate rows, you use the DISTINCT clause in the SELECT statement.

```
1 SELECT
2     lastname
3 FROM
4     employees
5 ORDER BY lastname;
```



```
1 SELECT DISTINCT
2     lastname
3 FROM
4     employees
5 ORDER BY lastname;
```

You can use the DISTINCT clause with more than one column.

```
1 SELECT DISTINCT
2     state, city
3 FROM
4     customers
```

Solution

- `SELECT DISTINCT city FROM offices`
-

Filtering

The WHERE clause allows you to specify exact rows to select based on a particular filtering expression or condition.

```
1  SELECT
2      lastname, firstname, jobtitle
3  FROM
4      employees
5  WHERE
6      jobtitle = 'Sales Rep';
```

Logical Operators

You can have more than one condition by using logical operator like AND and OR.

AND : both conditions have to be satisfied

OR : at least one condition has to be satisfied

```
1  SELECT
2      lastname, firstname, jobtitle
3  FROM
4      employees
5  WHERE
6      jobtitle = 'Sales Rep' AND officeCode = 1;
```

Get Dirty

Operator	
✓	=
	>
	>=
	<
	<=
	!=
	LIKE
	LIKE %...%
	NOT LIKE
	IN (...)
	NOT IN (...)
	BETWEEN
	NOT BETWEEN
	IS NULL
	IS NOT NULL

- list all the unique employee first names
 - who reports to employee #1102?
 - which sales rep report to #1088?
 - whose phone extension starts with a 4?
 - whose phone extension contains a 3 or a 5?
-

Get Dirty

Operator

✓ =

>

>=

<

<=

!=

LIKE

LIKE %...%

NOT LIKE

IN (...)

NOT IN (...)

BETWEEN

NOT BETWEEN

IS NULL

IS NOT NULL

- list all the unique employee first names

```
SELECT DISTINCT firstName FROM `employees`
```
- who reports to employee #1102?

```
SELECT * FROM employees WHERE reportsTo = 1102
```
- which sales rep report to #1088?

```
SELECT * FROM employees WHERE reportsTo = 1088  
AND jobTitle LIKE 'sales rep'
```
- whose phone extension starts with a 4?

```
SELECT lastname, extension FROM employees  
WHERE extension LIKE 'x4%'
```
- whose phone extension contains a 3 or a 5

```
SELECT lastname, firstname, extension FROM employees  
WHERE extension LIKE '%3%' OR extension LIKE '%5%'
```

Sorting

When you use the `SELECT` statement to query data from a table, the result set is not sorted in any orders. To sort the result set, you use the `ORDER BY` clause. The `ORDER BY` clause allows you to:

- Sort a result set by a single column or multiple columns.
 - Sort a result set by different columns in ascending (ASC) or descending order (DESC).
-

Sorting

```
1 SELECT
2   contactLastname,
3   contactFirstname
4 FROM
5   customers
6 ORDER BY
7   contactLastname;
```

```
1 SELECT
2   contactLastname,
3   contactFirstname
4 FROM
5   customers
6 ORDER BY
7   contactLastname DESC;
```

```
1 SELECT
2   contactLastname,
3   contactFirstname
4 FROM
5   customers
6 ORDER BY
7   contactLastname DESC,
8   contactFirstname ASC;
```

Aliasing

- To give a column a descriptive name, you use a column alias.

```
1 SELECT
2   CONCAT_WS(' ', lastName, firstname) `Full name`
3 FROM
4   employees
5 ORDER BY
6   `Full name`;
```

Grouping

- The GROUP BY clause, which is an optional part of the SELECT statement, groups a set of rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row for each group.
 - We often use the GROUP BY clause with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT. The aggregate function that appears in the SELECT clause provides the information about each group.
-

Aggregate functions

Name	Description
<u>AVG ()</u>	Return the average value of the argument
<u>BIT_AND ()</u>	Return bitwise AND
<u>BIT_OR ()</u>	Return bitwise OR
<u>BIT_XOR ()</u>	Return bitwise XOR
<u>COUNT ()</u>	Return a count of the number of rows returned
<u>COUNT (DISTINCT)</u>	Return the count of a number of different values
<u>GROUP_CONCAT ()</u>	Return a concatenated string
<u>MAX ()</u>	Return the maximum value
<u>MIN ()</u>	Return the minimum value
<u>STD ()</u>	Return the population standard deviation
<u>STDDEV ()</u>	Return the population standard deviation
<u>STDDEV_POP ()</u>	Return the population standard deviation
<u>STDDEV_SAMP ()</u>	Return the sample standard deviation
<u>SUM ()</u>	Return the sum
<u>VAR_POP ()</u>	Return the population standard variance
<u>VAR_SAMP ()</u>	Return the sample variance
<u>VARIANCE ()</u>	Return the population standard variance

GROUP BY

```
1 SELECT
2     status, COUNT(*)
3 FROM
4     orders
5 GROUP BY status;
```

```
1 SELECT
2     orderNumber,
3     SUM(quantityOrdered * priceEach) AS total
4 FROM
5     orderdetails
6 GROUP BY orderNumber;
```


Name
<u>AVG ()</u>
<u>BIT_AND ()</u>
<u>BIT_OR ()</u>
<u>BIT_XOR ()</u>
<u>COUNT ()</u>
<u>COUNT (DISTINCT)</u>
<u>GROUP_CONCAT ()</u>
<u>MAX ()</u>
<u>MIN ()</u>
<u>STD ()</u>
<u>STDDEV ()</u>
<u>STDDEV_POP ()</u>
<u>STDDEV_SAMP ()</u>
<u>SUM ()</u>
<u>VAR_POP ()</u>
<u>VAR_SAMP ()</u>
<u>VARIANCE ()</u>

Get Dirty

- the total quantity and average item price for each order
 - the number, in descending order, of different product for each order
 - the cheapest product
 - how many products are in the “Vintage Cars” product line
-

Name

AVG()

BIT_AND()

BIT_OR()

BIT_XOR()

COUNT()

COUNT(DISTINCT)

GROUP_CONCAT()

MAX()

MIN()

STD()

STDDEV()

STDDEV_POP()

STDDEV_SAMP()

SUM()

VAR_POP()

VAR_SAMP()

VARIANCE()

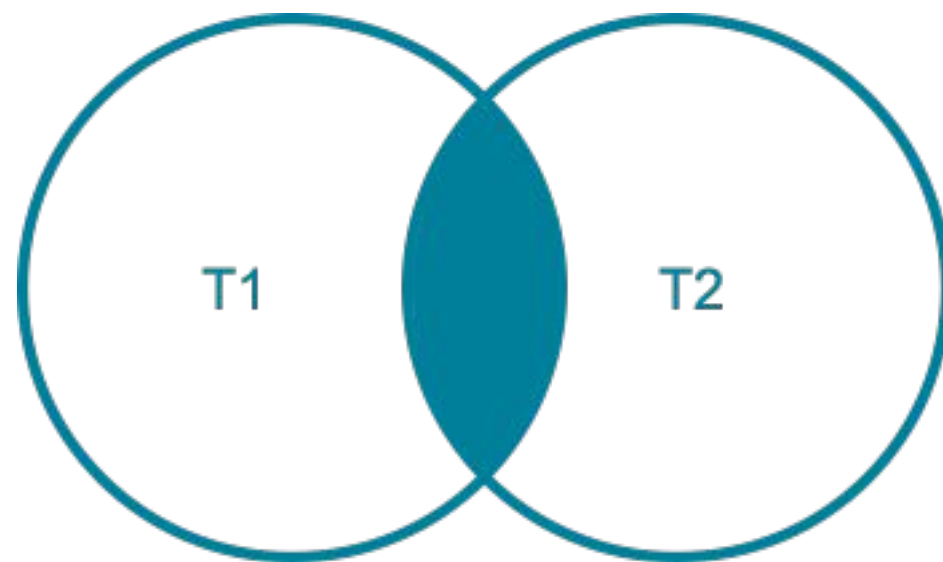
Get Dirty

- ```
SELECT orderNumber, SUM(quantityOrdered), AVG(priceEach)
FROM orderdetails
GROUP BY orderNumber
LIMIT 0 , 30
```
- ```
SELECT orderNumber, COUNT( * ) AS nbProducts
FROM orderdetails
GROUP BY orderNumber
ORDER BY nbProducts DESC
```
- ```
SELECT *
FROM `products`
ORDER BY MSRP
LIMIT 1
```

st product
- ```
SELECT COUNT( * )
FROM `products`
WHERE productLine LIKE 'vintage cars'
GROUP BY productLine
```

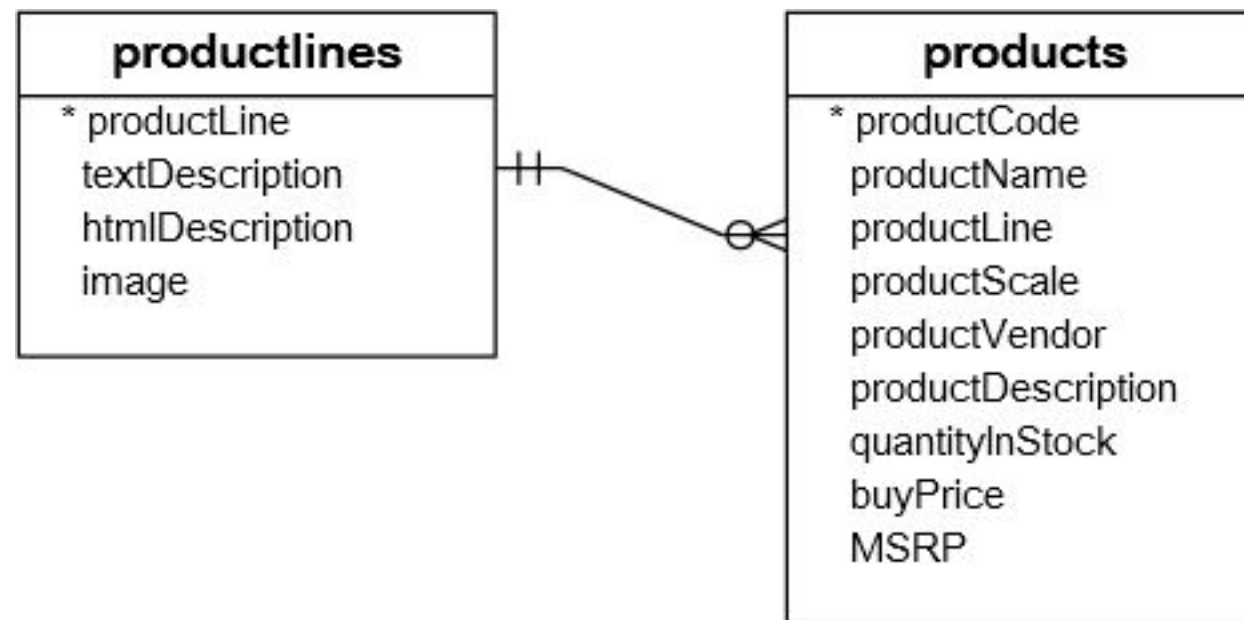
the “Vintage

Joining



- The MySQL INNER JOIN clause matches rows in one table with rows in other tables and allows you to query rows that contain columns from both tables.
-

Join example 1



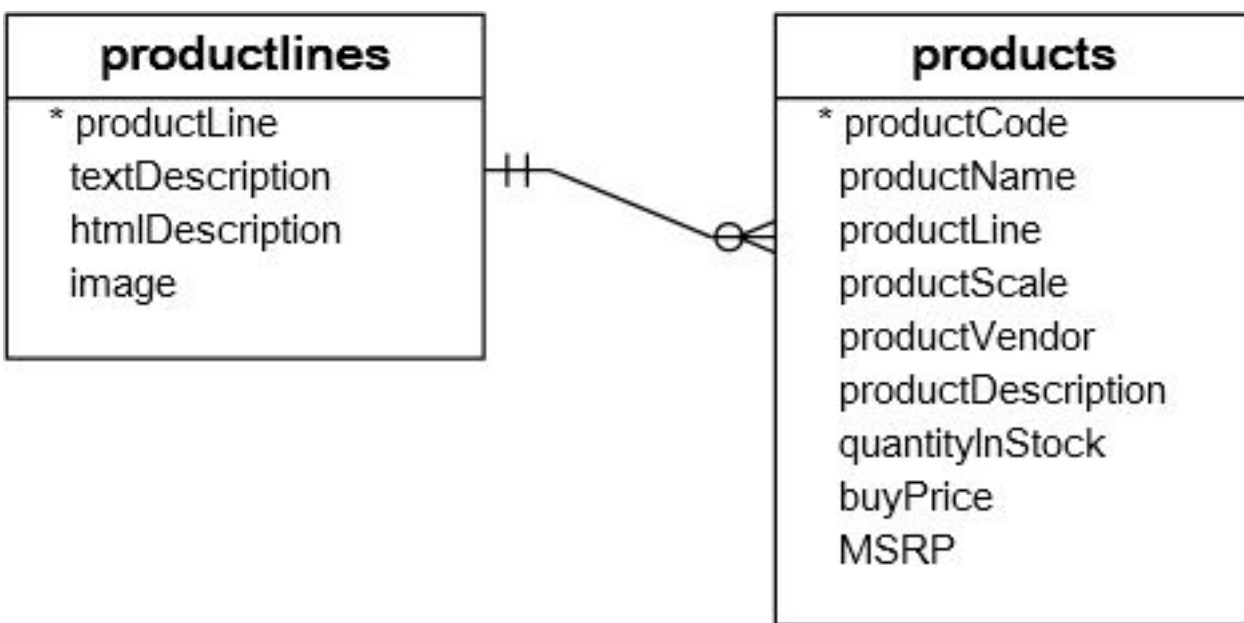
How to get...

- The product code and product name from the products table.

AND

- The text description of product lines from the productlines table.
-

Join example 1



```
1 SELECT
2   productCode,
3   productName,
4   textDescription
5 FROM
6   products T1
7 INNER JOIN productlines T2 ON T1.productline = T2.productline;
```

How to get...

- The product code and product name from the products table.

AND

- The text description of product lines from the productlines table.
-

Join example 1

	productCode	productName	textDescription
	S10_1949	1952 Alpine Renault 1300	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S10_4757	1972 Alfa Romeo GTA	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S10_4962	1962 LanciaA Delta 16V	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S12_1099	1968 Ford Mustang	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S12_1108	2001 Ferrari Enzo	Attention car enthusiasts: Make your wildest car ownership dreams come true.

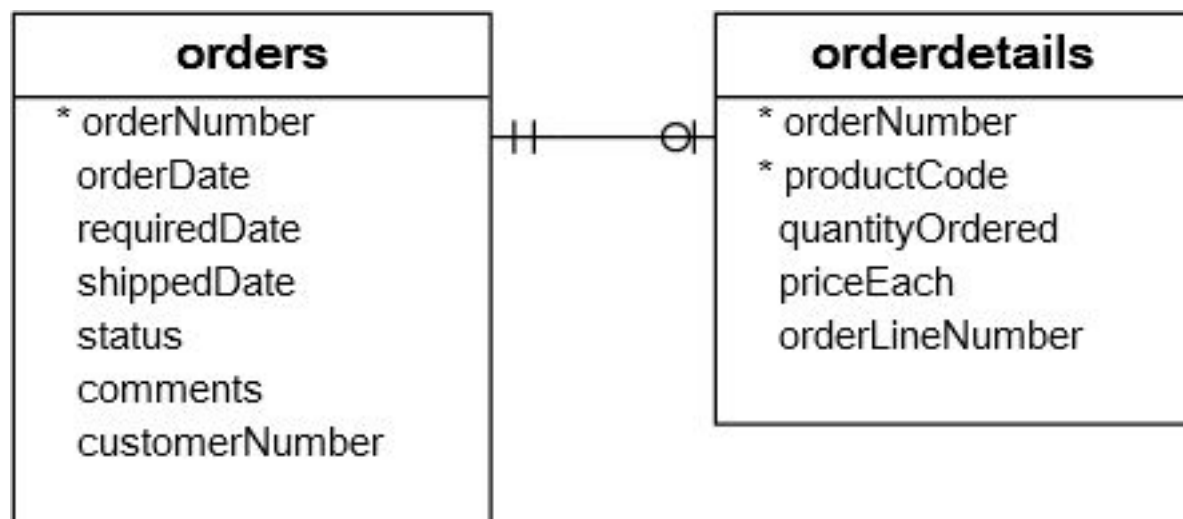
How to get...

- The product code and product name from the products table.

AND

- The text description of product lines from the productlines table.
-

Join example 2



- We can get the order number, order status and total sales from the *orders* and *orderdetails* tables using the INNER JOIN clause with the GROUP BY clause as follows:

```
1 SELECT
2   T1.orderNumber,
3   STATUS,
4   SUM(quantityOrdered * priceEach) total
5 FROM
6   orders AS T1
7 INNER JOIN orderdetails AS T2 ON T1.orderNumber = T2.orderNumber
8 GROUP BY
9   orderNumber;
```

	orderNumber	status	total
	10100	Shipped	10223.83
	10101	Shipped	10549.01
	10102	Shipped	5494.78
	10103	Shipped	50218.95
	10104	Shipped	40206.20

Get Dirty

- get a list of employees names and the city of their office. First and last name may remain separate columns.

full name	city
Diane Murphy	San Francisco
Mary Patterson	San Francisco
Jeff Firrelli	San Francisco
Anthony Bow	San Francisco
Leslie Jennings	San Francisco
Leslie Thompson	San Francisco
Julie Firrelli	Boston
Steve Patterson	Boston
Foon Yue Tseng	NYC
George Vanauf	NYC
Gerard Bondur	Paris
Loui Bondur	Paris
Gerard Hernandez	Paris
Pamela Castillo	Paris
Martin Gerard	Paris
Mami Nishi	Tokyo
Yoshimi Kato	Tokyo
William Patterson	Sydney
Andy Fixter	Sydney
Peter Marsh	Sydney
Tom King	Sydney
Larry Bott	London
Barry Jones	London

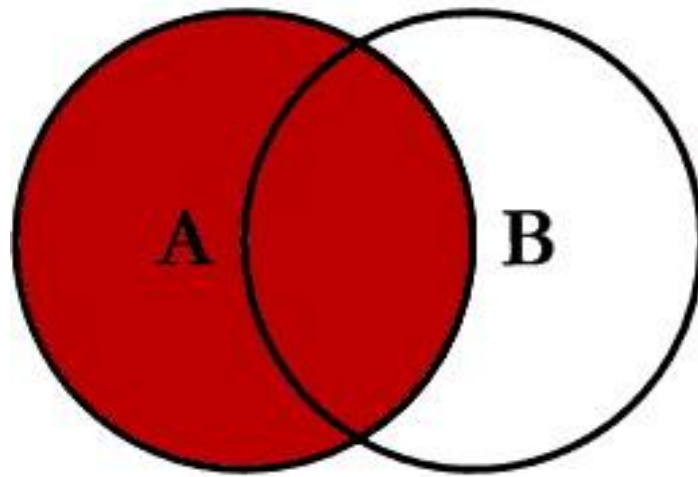
Get Dirty

- get a list of employees names and the city of their office

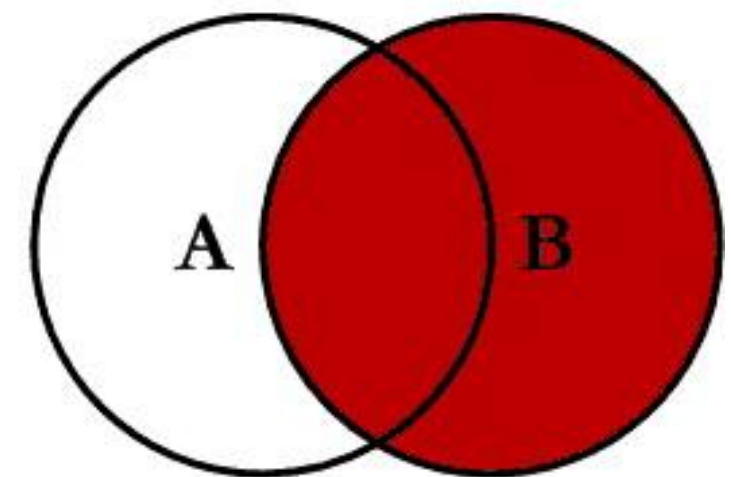
```
SELECT CONCAT_WS( ' ', firstName, lastName ) AS 'full name', city
FROM `employees` T1
INNER JOIN offices T2 ON T1.officeCode = T2.officeCode
```

full name	city
Diane Murphy	San Francisco
Mary Patterson	San Francisco
Jeff Firrelli	San Francisco
Anthony Bow	San Francisco
Leslie Jennings	San Francisco
Leslie Thompson	San Francisco
Julie Firrelli	Boston
Steve Patterson	Boston
Foon Yue Tseng	NYC
George Vanauf	NYC
Gerard Bondur	Paris
Loui Bondur	Paris
Gerard Hernandez	Paris
Pamela Castillo	Paris
Martin Gerard	Paris
Mami Nishi	Tokyo
Yoshimi Kato	Tokyo
William Patterson	Sydney
Andy Fixter	Sydney
Peter Marsh	Sydney
Tom King	Sydney
Larry Bott	London
Barry Jones	London

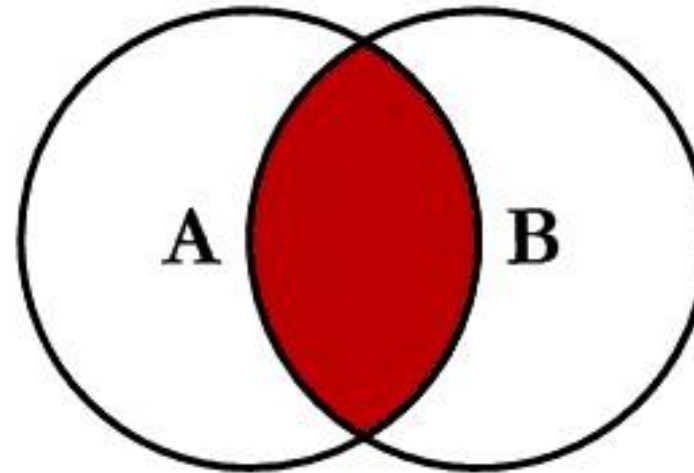
SQL JOINS



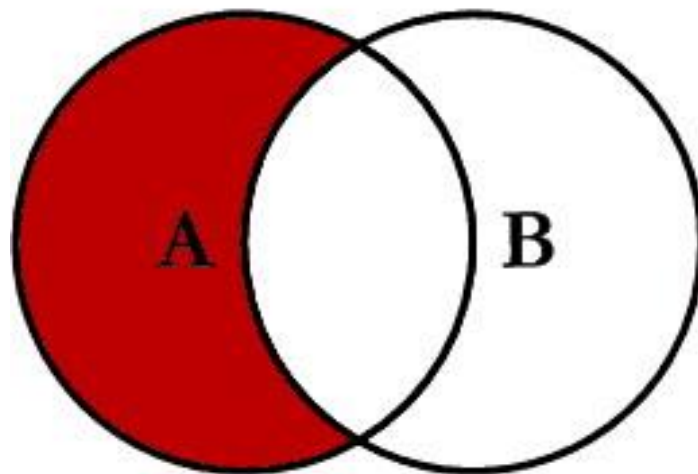
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



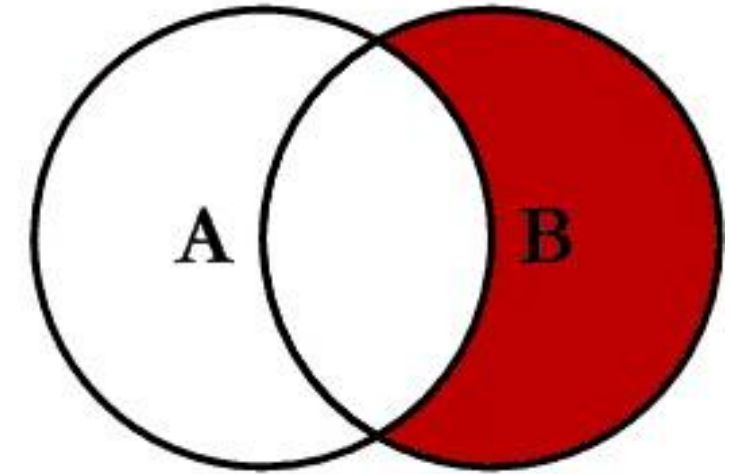
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



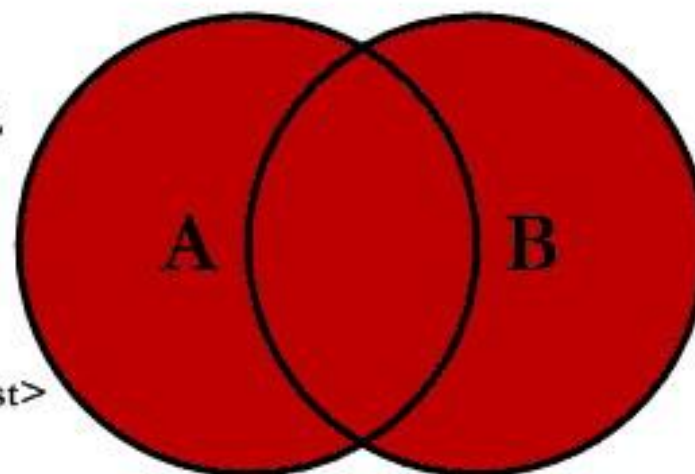
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



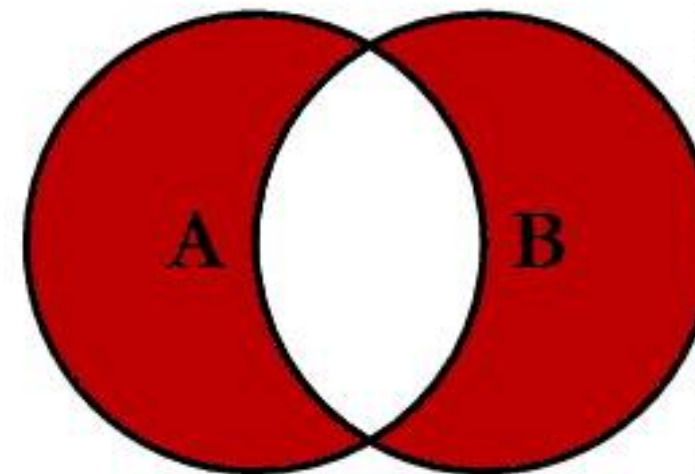
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```


String Functions

- <https://dev.mysql.com/doc/refman/5.7/en/string-functions.html>

Name	Description
<u>ASCII()</u>	Return numeric value of left-most character
<u>BIN()</u>	Return a string containing binary representation of a number
<u>BIT_LENGTH()</u>	Return length of argument in bits
<u>CHAR()</u>	Return the character for each integer passed
<u>CHAR_LENGTH()</u>	Return number of characters in argument
<u>CHARACTER_LENGTH()</u>	Synonym for CHAR_LENGTH()
<u>CONCAT()</u>	Return concat enated string
<u>CONCAT_WS()</u>	Return concat enate with separator
<u>ELT()</u>	Return string at index number
<u>EXPORT_SET()</u>	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
<u>FIELD()</u>	Return the index (position) of the first argument in the subsequent arguments
<u>FIND_IN_SET()</u>	Return the index position of the first argument within the second argument
<u>FORMAT()</u>	Return a number formatted to specified number of decimal places
<u>FROM_BASE64()</u>	Decode to a base-64 string and return result
<u>HEX()</u>	Return a hexadecimal representation of a decimal or string value
<u>INSERT()</u>	Insert a substring at the specified position up to the specified number of characters
<u>INSTR()</u>	Return the index of the first occurrence of substring
<u>LCASE()</u>	Synonym for LOWER()
<u>LEFT()</u>	Return the leftmost number of characters as specified
<u>LENGTH()</u>	Return the length of a string in bytes
<u>LIKE</u>	Simple pattern matching
<u>LOAD_FILE()</u>	Load the named file
<u>LOCATE()</u>	Return the position of the first occurrence of substring
<u>LOWER()</u>	Return the argument in lowercase
<u>LPAD()</u>	Return the string argument, left-padded with the specified string
<u>LTRIM()</u>	Remove leading spaces
<u>MAKE_SET()</u>	Return a set of comma-separated strings that have the corresponding bit in bits set
<u>MATCH</u>	Perform full-text search
<u>MID()</u>	Return a substring starting from the specified position
<u>NOT LIKE</u>	Negation of simple pattern matching
<u>NOT REGEXP</u>	Negation of REGEXP
<u>OCT()</u>	Return a string containing octal representation of a number
<u>OCTET_LENGTH()</u>	Synonym for LENGTH()
<u>ORD()</u>	Return character code for leftmost character of the argument
<u>POSITION()</u>	Synonym for LOCATE()
<u>QUOTE()</u>	Escape the argument for use in an SQL statement
<u>REGEXP</u>	Pattern matching using regular expressions
<u>REPEAT()</u>	Repeat a string the specified number of times
<u>REPLACE()</u>	Replace occurrences of a specified string
<u>REVERSE()</u>	Reverse the characters in a string
<u>RIGHT()</u>	Return the specified rightmost number of characters
<u>RLIKE</u>	Synonym for REGEXP
<u>RPAD()</u>	Append string the specified number of times
<u>RTRIM()</u>	Remove trailing spaces
<u>SOUNDEX()</u>	Return a soundex string
<u>SOUNDS LIKE</u>	Compare sounds
<u>SPACE()</u>	Return a string of the specified number of spaces
<u>STRCMP()</u>	Compare two strings
<u>SUBSTR()</u>	Return the substring as specified
<u>SUBSTRING()</u>	Return the substring as specified
<u>SUBSTRING_INDEX()</u>	Return a substring from a string before the specified number of occurrences of the delimiter
<u>TO_BASE64()</u>	Return the argument converted to a base-64 string
<u>TRIM()</u>	Remove leading and trailing spaces
<u>UCASE()</u>	Synonym for UPPER()
<u>UNHEX()</u>	Return a string containing hex representation of a number
<u>UPPER()</u>	Convert to uppercase
<u>WEIGHT_STRING()</u>	Return the weight string for a string

Get Dirty

- get a list of employees names and the city of their office
- and the number of customers they work with
- order by the descending nb of customers

full name	city	nb customers ▼
Pamela Castillo	Paris	10
Barry Jones	London	9
Larry Bott	London	8
George Vanauf	NYC	8
Gerard Hernandez	Paris	7
Foon Yue Tseng	NYC	7
Leslie Thompson	San Francisco	6
Loui Bondur	Paris	6
Steve Patterson	Boston	6
Leslie Jennings	San Francisco	6
Martin Gerard	Paris	6
Julie Firrelli	Boston	6
Mami Nishi	Tokyo	5
Peter Marsh	Sydney	5
Andy Fixter	Sydney	5

Get Dirty

- get a list of employees names and the city of their office
- and the number of customers they work with
- order by the descending nb of customers

full name	city	nb customers
Pamela Castillo	Paris	10
Barry Jones	London	9
Larry Bott	London	8
George Vanauf	NYC	8
Gerard Hernandez	Paris	7
Foon Yue Tseng	NYC	7
Leslie Thompson	San Francisco	6
Loui Bondur	Paris	6
Steve Patterson	Boston	6
Leslie Jennings	San Francisco	6
Martin Gerard	Paris	6
Julie Firrelli	Boston	6
Mami Nishi	Tokyo	5
Peter Marsh	Sydney	5
Andy Fixter	Sydney	5

```
SELECT CONCAT_WS( ' ', firstName, lastName ) AS 'full name', T2.city, COUNT( DISTINCT T3.customerNumber ) AS 'nb customers'
FROM `employees` T1
INNER JOIN offices T2 ON T1.officeCode = T2.officeCode
INNER JOIN customers T3 ON T1.employeeNumber = T3.salesRepEmployeeNumber
GROUP BY T1.employeeNumber
ORDER BY `nb customers` DESC
```

Get Dirty

- get a list of employees names and the city of their office, the number of customers they work with, order by the descending nb of customers

- how many sales in 2004
- for how much total money

full name	city	nb customers ▾	nb sales	total \$\$
Pamela Castillo	Paris	9	14	409910.07
Barry Jones	London	9	14	388872.38
Larry Bott	London	7	9	303470.32
George Vanauf	NYC	7	12	401758.60
Foon Yue Tseng	NYC	6	9	237255.26
Steve Patterson	Boston	6	10	327602.21
Gerard Hernandez	Paris	6	15	418367.27
Martin Gerard	Paris	5	7	207828.89
Leslie Jennings	San Francisco	5	10	291693.96
Julie Firrelli	Boston	5	7	129916.12
Loui Bondur	Paris	5	8	254002.97
Mami Nishi	Tokyo	4	6	151761.45
Peter Marsh	Sydney	4	7	247176.25
Leslie Thompson	San Francisco	4	6	185038.40
Andy Fixter	Sydney	3	5	172377.82

Get Dirty

- get a list of employees names and the city of their office, the number of customers they work with, order by the descending nb of customers

```
SELECT CONCAT_WS(' ', firstName,lastName) AS 'full name', T2.city, COUNT(DISTINCT
T3.customerNumber) AS 'nb customers', count(distinct T4.orderNumber) AS 'nb sales'
FROM `employees` T1
INNER JOIN offices T2 ON T1.officeCode = T2.officeCode
INNER JOIN customers T3 ON T1.employeeNumber = T3.salesRepEmployeeNumber
INNER JOIN orders T4 ON T3.customerNumber = T4.customerNumber
GROUP BY T1.employeeNumber
ORDER BY `nb customers` DESC
```

- and how many sales

```
WHERE T4.orderDate BETWEEN CAST('2004-01-01' AS DATE) AND CAST('2004-12-01' AS DATE)
```

- how many sales in 2004

```
1 SELECT CONCAT_WS(' ', firstName,lastName) AS 'full name', T2.city, COUNT(DISTINCT
T3.customerNumber) AS 'nb customers', count(distinct T4.orderNumber),
SUM(T5.priceEach*T5.quantityOrdered) AS 'total revenu'
2 FROM `employees` T1
3 INNER JOIN offices T2 ON T1.officeCode = T2.officeCode
4 INNER JOIN customers T3 ON T1.employeeNumber = T3.salesRepEmployeeNumber
5 INNER JOIN orders T4 ON T3.customerNumber = T4.customerNumber
6 INNER JOIN orderdetails T5 ON T4.orderNumber = T5.orderNumber
7 WHERE T4.orderDate BETWEEN CAST('2004-01-01' AS DATE) AND CAST('2004-12-01' AS DATE)
8 GROUP BY T1.employeeNumber
9 ORDER BY `nb customers` DESC
```

Insert, Update, Replace, Delete and So Much more...

- mysqltutorial.org — “Basic MySQL Tutorial”
 - lynda.dartmouth.edu — search “SQL” or “Database”
 - dev.mysql.com/doc/
-

Announcements

More RC workshops:

- <https://rc.dartmouth.edu> > Training

Support:

- christian.darabos@dartmouth.edu
- research.computing@dartmouth.edu