

# Relational Databases

*using MySQL*

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

March 17, 2016



# Agenda

## Introduction

- Pracmatic Approach

## Using MySQL

- Getting and Installing

- Example database

## Creating and populating the database

- Two ways

- Creating the database

- Creating tables

- Entering values

- Updating

- Foreign keys

## Manipulating data with SQL

- SELECT and FROM

- WHERE

- ORDER BY

# INTRODUCTION TO DATABASES

# Relational databases

- ▶ An important purpose of computers have long been to store and manipulate data.
  - ▶ Not the only purpose, of course, but important in many applications.
- ▶ Data can be stored in many ways (flat file, using hierarical structures, network and more).
- ▶ In the beginning of the 70s, Edgar Codd at IBM created the *relational database*, based on the mathematical concept of relations.
  - ▶ This means that data is structured in a way consistent with first-order predicate logic.
- ▶ This makes it possible to define both data and queries with a *declarative* language.
- ▶ The most popular, and most standadised, way of doing that today is using SQL – Structured Query Language.

# Database Management Systems

- ▶ Database Management Systems, or DBMS for short, are software applications that store and manipulate data.
- ▶ The most common DBMS today is a relational one where the largest, in terms of userbase, are:
  - ▶ Oracle, now at version 12c.
  - ▶ DB2 by IBM.
  - ▶ SQL Server from Microsoft.
- ▶ There are also many free and open alternatives, among the most known are:
  - ▶ MySQL – originally from a Swedish company, later sold to Sun Microsystems which in turn was sold to Oracle.
  - ▶ PostgreSQL – a long living open alternative originally based on Ingres.
  - ▶ SQLite – smaller, not using the client-server model of the others. Easy to embed.

# Databases in this course

- ▶ You can fill hours with theory on how databases work and how to use them.
- ▶ In this course we are going to use a very pragmatic approach, we are going to *use* databases.
- ▶ Each database will consist of a number of *tables*.
- ▶ Each table has a number of columns, each column holds a specific kind of data.
  - ▶ A string representing a name for example.
- ▶ All values of a column represent the same thing (name or whatever it might be).
- ▶ One row in the table represent an *entity*, something of value.
  - ▶ For example all the data we are intrested in for a person.

## An example

- ▶ The following could be a table for the entity “Person”:

| ID | Name            | Shoe size |
|----|-----------------|-----------|
| 1  | Charles-Clemens | 47        |
| 2  | Bo-Bertil       | 34        |
| 3  | Nisse P         | 42        |

- ▶ One (or more) columns represent the *key*, a value that uniquely identifies the row.
- ▶ To connect to other tables, the key from one column can be used as a *foreign key* in another table.

| ID | Shoe name   | Buyer |
|----|-------------|-------|
| X  | Ecco Rider  | 1     |
| Y  | Tretorn Hal | 1     |
| Z  | Foppatoffla | 2     |

## USING MySQL



# MySQL

- ▶ The database manager we recommend that you use in this course is *MySQL*.
  - ▶ In reality it doesn't matter which one (as long as it is a client-server DBMS), but to make things easy use MySQL.
- ▶ MySQL was originally Swedish, but now in the hands of Oracle.
- ▶ It is available in both free and commercial versions, in this course the free is more than good enough.
- ▶ The name comes from the daughter of one of the original developers.
  - ▶ As he was not happy with Oracle as owner (through the acquisition of Sun Microsystems) he developed a new database manager, named after his next daughter – MariaDB...

# Getting MySQL

- ▶ MySQL is available for all larger platforms – Linux, MacOS X and Windows.
- ▶ The free version is available from <http://dev.mysql.com>
- ▶ At the time of writing the latest stable version is 5.7.11.
  - ▶ Use this or the older version 5.6 but stay away from the beta releases...
- ▶ When installing, you will be prompted for a root password – do not forget this!
  - ▶ If you do, you need to completely remove MySQL and begin from the beginning!
- ▶ If you use one of the installers provided by MySQL, remember to install MySQL Workbench as well.
- ▶ MySQL Workbench is also available at <http://dev.mysql.com/downloads/tools/workbench/>

## Example database

- ▶ Below is a view of the database we are going to create in this first part.
- ▶ The name of the tables is in blue and primary keys are underlined.

**Dinosaur** DName, *DClade*, *DPeriod*, Discovered,  
Description

**Period** PName, Begin, End

**Clade** CName, Diet, Stance

- ▶ *Clade* means “a group of organisms believed to have evolved from a common ancestor”.
- ▶ *Stance* means “the way in which someone stands”.

## CREATING AND POPULATING THE DATABASE

## Two ways

- ▶ The database can be created and populated in two different ways as we have two clients.
- ▶ In a terminal, type `mysql -u root -p` and press Return.
  - ▶ Enter the password you created earlier.
- ▶ If all goes well, you will have a prompt saying `mysql >`
- ▶ To create a database, simply type:  
`create database dinosaurDB;`
- ▶ To show all available databases, type `verb|show databases;`
- ▶ Notice that all lines end with a semicolon.

```
tobias@tobias-VirtualBox:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.5.35-0ubuntu0.13.10.2 (Ubuntu)
```

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

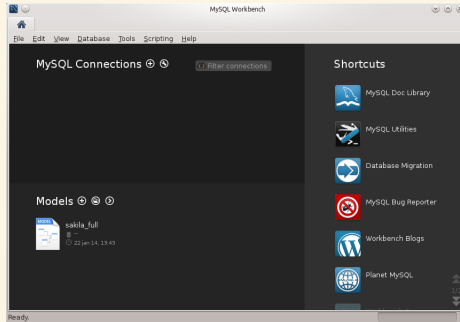
```
mysql> create database dinosaurDB;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| dinosaurDB |
| mysql |
| performance_schema |
| test |
+-----+
5 rows in set (0.00 sec)

mysql>
```

# Visually with Workbench

- It is also possible to do it using Workbench.



- Press the plus sign at “MySQL Connections” to create a connection to a database server.

# New connection

- Name the connection as you like and enter the URL to connect to it.

Setup New Connection

Connection Name:  Type a name for the connection

Connection Method:  Method to use to connect to the RDBMS

Parameters ☒ SSL ☐ Advanced

Hostname:  Port:  Name or IP address of the server host - TCP/IP port.

Username:  Name of the user to connect with.

Password:  The user's password. Will be requested later if it's not set.

Default Schema:  The schema to use as default schema. Leave blank to select it later.



# Create the database

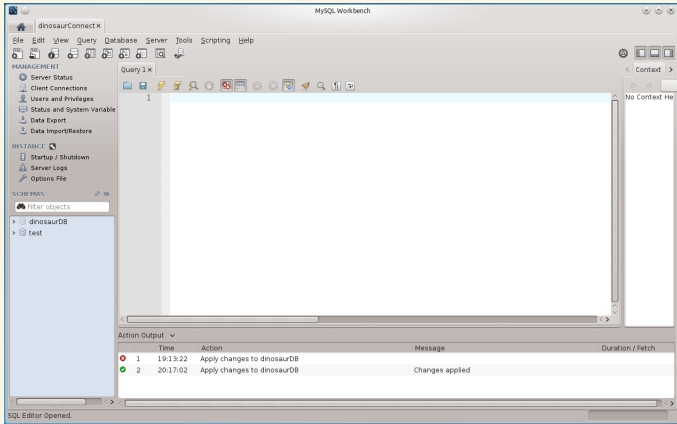
- ▶ When the connection is done it is possible to use it to create the database itself.
- ▶ In the toolbar there are many icons, one which is named “Create new schema...” which creates the database.



- ▶ In the dialogue that appears, name the database dinosaurDB.
- ▶ When pressing “Apply” the code for creating the database is shown on screen.

# The workspace

- This image shows the workspace when a database has been newly created.

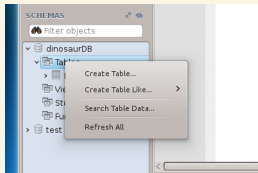


# Creating tables

- ▶ The database should contain tables, where each table has a number of columns of different data types.
- ▶ To do it in the terminal, type:

```
CREATE TABLE Dinosaur (DName varchar(50) not null,  
    DClade nvarchar(20),  
    DPeriod nvarchar(20),  
    Discovered int,  
    Description nvarchar(4000),  
    primary key(DName));
```

- ▶ The same can be done in Workbench by right clicking the database and select “Create Table” and filling in the values.

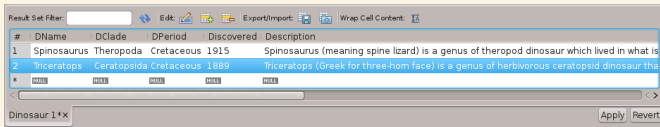


# Entering values

- ▶ When the tables are in place, fill them with values using INSERT INTO:

```
INSERT INTO Dinosaur
-> VALUES ('Spinosaurus', 'Theropoda', 'Cretaceous', 1915,
--- cut for space --- The best known species is S. aegypti
maroccanus has been recovered from Morocco.');
```

- ▶ Or in Workbench by right clicking the table, selecting “Select Rows” and entering i table format:



- ▶ Remember to press “Apply”, otherwise no values will be saved.

# Updating table data

- ▶ To update the contents of a table, use UPDATE which has a WHERE clause that defines the conditions for the update.
- ▶ For example, say we like to update the time for the jurassic period from 66 million years ago to 65, type:

```
UPDATE Period  
SET End = 65  
WHERE PName = 'Cretaceous';
```

- ▶ The same can be done in Workbench by simply editing the data after right clicking and selecting “Select rows”.
- ▶ If data is to be removed, there is also a DELETE command.

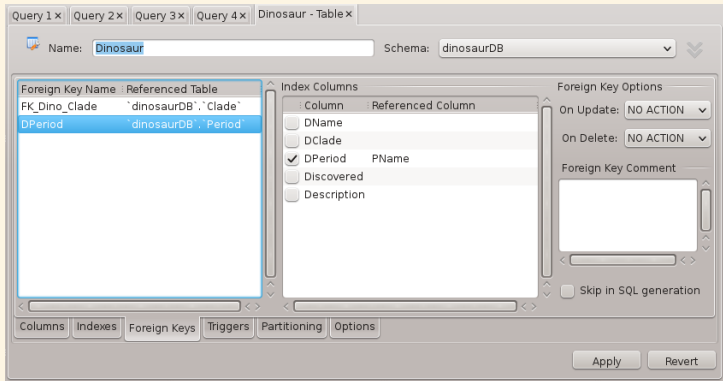
# Foreign keys

- ▶ Foreign keys can be defined either at the creation of the table or afterwards.
- ▶ To do it later means to use ALTER TABLE, which can be used for all sorts of changes of the table.
- ▶ To set DClade as a foreign key referring to the name in the table Clade, type:

```
mysql> ALTER TABLE Dinosaur
      -> ADD CONSTRAINT FK_Dino_Clade
      -> FOREIGN KEY (DClade)
      -> REFERENCES Clade (CName);
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

# Foreign keys in Workbench

- ▶ To set a foreign key in Workbench, right click the table and select “Alter table...”
- ▶ In the tab named “Foreign keys”, name and select the keys that should be connected:



## MANIPULATING DATA WITH SQL

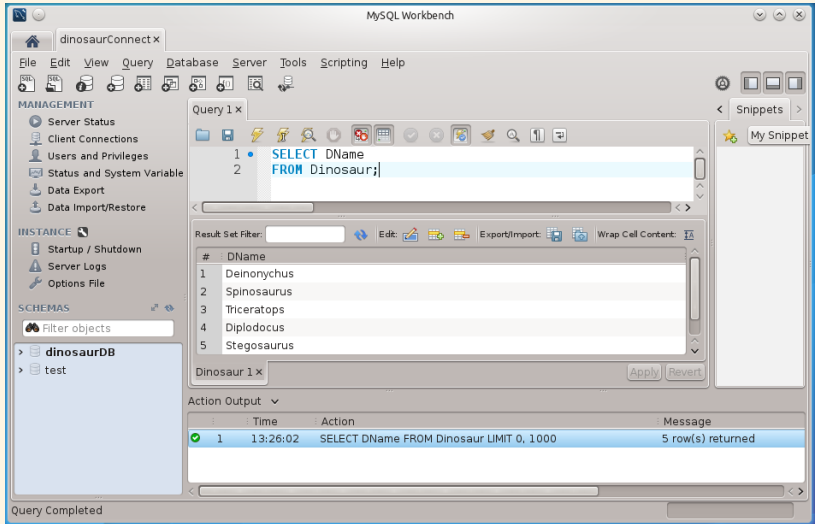


# Working with SQL

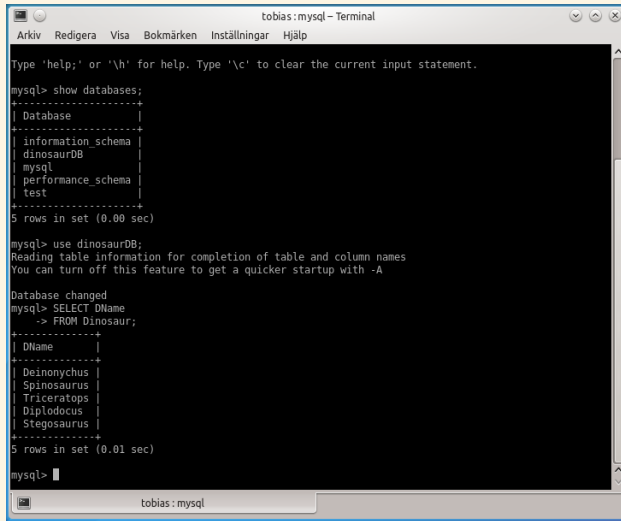
- ▶ SQL, Structured Query Language, is the primary way of asking questions to the database.
- ▶ When stating a query, it is done *in declarative way* using clauses.
- ▶ The two first, and simplest, clauses are SELECT and FROM.
- ▶ With SELECT you state *what* should be collected from the database.
  - ▶ One or more columns are selected, or all of them using a \*.
- ▶ To decide what tables to get the data from, use FROM.
- ▶ This can be done both in the terminal and in Workbench with our first example being shown with both:

```
SELECT DName  
FROM Dinosaur;
```

# In Workbench



# In the terminal



The screenshot shows a terminal window titled "tobias: mysql - Terminal". The terminal displays the output of the following MySQL commands:

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| dinosaurDB |
| mysql |
| performance_schema |
| test |
+-----+
5 rows in set (0.00 sec)

mysql> use dinosaurDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT DName
   -> FROM Dinosaur;
+-----+
| DName |
+-----+
| Deinonychus |
| Spinosaurus |
| Triceratops |
| Diplodocus |
| Stegosaurus |
+-----+
5 rows in set (0.01 sec)

mysql>
```

The terminal window has a menu bar with "Arkiv", "Redigera", "Visa", "Bokmärken", "Inställningar", and "Hjälp". The status bar at the bottom shows "tobias : mysql".

# More examples

- Show name and period for all dinosaurs:

```
SELECT DName, DPeriod
FROM Dinosaur;
```

- Result:

|             |            |
|-------------|------------|
| Deinonychus | Cretaceous |
| Spinosaurus | Cretaceous |
| Triceratops | Cretaceous |
| Diplodocus  | Jurassic   |
| Stegosaurus | Jurassic   |

- Show all information from all dinosaurs:

```
SELECT *
FROM Dinosaur;
```

- Result (shortend):

|   |                 |            |      |                    |
|---|-----------------|------------|------|--------------------|
| Deinonychus   | Theropoda       | Cretaceous | 1969 | There is one descr |
| 108 million years ago (from the mid-Aptian to early Albian stages). F |                 |            |      |                    |
| Diplodocus  | Sauropodomorpha | Jurassic   | 1878 | This genus of dir  |
| Spinosaurus   | Theropoda       | Cretaceous | 1915 | Spinosaurus (mean  |
| Stegosaurus   | Ornithischia    | Jurassic   | 1877 | They lived during  |
| Triceratops   | Ceratopsidae    | Cretaceous | 1889 | Triceratops (Gre   |

## WHERE clause

- ▶ With WHERE a predicate is added to the search.
  - ▶ This defines a condition that is either true or false.
- ▶ This defines a filter or selection on the query to make it more exact.
- ▶ All common relational operators are possible to use in the WHERE clause.
  - ▶ For example =, <, >, !=, & (AND), || (OR)
- ▶ An example is to find the name of the dinosaur called Spinosaurus:

```
SELECT DName
FROM Dinosaur
WHERE DName = 'Spinosaurus';
```

- ▶ Result:  
Spinosaurus

## Additional examples

- Show the name of the dinosaurs living during the jurassic time period:

```
SELECT DName
FROM Dinosaur
WHERE DPeriod = 'Jurassic';
```

**Result:**

|             |
|-------------|
| Diplodocus  |
| Stegosaurus |

- Show the name of the dinosaurs found after the year 1900:

```
SELECT DName, Discovered
FROM Dinosaur
WHERE Discovered > 1900;
```

**Result:**

|             |      |
|-------------|------|
| Deinonychus | 1969 |
| Spinosaurus | 1915 |

- Show name and time for dinosaurs found before 1877 and lived in jurassic time.

```
SELECT DName, Discovered
FROM Dinosaur
WHERE Discovered <= 1877
      AND DPeriod = 'Jurassic';
```

**Resultat:**

|             |      |
|-------------|------|
| Stegosaurus | 1877 |
|-------------|------|

# Finding patterns

- ▶ Sometimes it is hard to know exactly what you are looking for, but you know some of it.
- ▶ In WHERE it is possible to add LIKE and define a pattern using wildcards.
  - ▶ % for one or more characters
  - ▶ \_ for exactly one character.
- ▶ An example could be to search for all dinosaurs with names ending with 'saurus':

```
SELECT DName, DPeriod  
FROM Dinosaur  
WHERE DName LIKE '%saurus';
```

|             |            |
|-------------|------------|
| Spinosaurus | Cretaceous |
| Stegosaurus | Jurassic   |

## Ordering output

- ▶ The order by which the data is presented is sometimes of no importance, but sometimes very important.
- ▶ The clause `ORDER BY` specifies which column data should be sorted on.
- ▶ Order can be either ascending (`ASC`) or descending (`DESC`).
- ▶ An example, show name and time period for all dinosaurs descending on name:

```
SELECT DName, DPeriod  
FROM Dinosaur  
ORDER BY DNAME DESC;
```

- ▶ Result:

|             |            |
|-------------|------------|
| Triceratops | Cretaceous |
| Stegosaurus | Jurassic   |
| Spinosaurus | Cretaceous |
| Diplodocus  | Jurassic   |
| Deinonychus | Cretaceous |



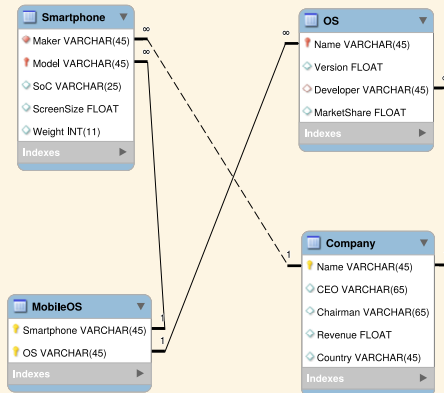
## MORE ON QUERING

## More on querying

- ▶ The previous was the bare minimum that needs to be known for SQL.
  - ▶ In fact, you will probably get very far in the project using only that.
- ▶ To fully utilise a database, there are many more things that can be done.
- ▶ This section is introducing many of those, some of which will be very handy for you to use.
- ▶ First, though, we will introduce a new database!
  - ▶ About mobile phones and unfortunately it is beginning to show some of its age...

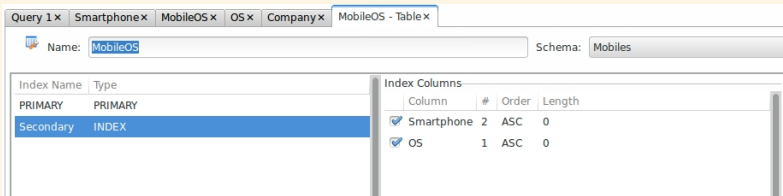
# Structure

- The following is an ER-diagram of the database where the straight lines are keys made of primary keys and dotted are referring to non-keys.



## A short note...

- ▶ If you like to create the database yourselves (or have something similar), you might run in to a problem with indexes.
- ▶ The table MobileOS contains two primary keys from two tables where those keys are primary keys as well.
- ▶ Because of this, an indexes need to be set for both keys in this table.
  - ▶ Otherwise the error 1215 will be shown when creating the table.
- ▶ In Workbench this can be done by altering the table and on the tab “Indexes” mark both columns.



# Aggregate functions

- ▶ SQL has a large number of functions that perform calculations on columns.
- ▶ Some of them are:
  - ▶ `AVG()` for average.
  - ▶ `COUNT()` for calculating the number of rows.
  - ▶ `MIN()` and `MAX()` for smallest and largest values.
- ▶ The function is used as part of the `SELECT` clause.
- ▶ The result can later be filtered or grouped.
- ▶ Observe that rows with `NULL` values are not calculated, they are disregarded totally.

# Examples

- What is the average screen size for the smartphones?

```
select AVG(ScreenSize)
from Smartphone;
```

**Result:**

'4.96349995136261'

- Show the average revenue for companies in USA:

```
select AVG(Revenue)
from Company
where Country = 'USA';
```

**Result:**

'83.94724986329675'

- Show the name and marked share of the largest OS.

```
select Name, MAX(MarketShare)
from OS;
```

**Result:**

'Android', '84.4000015258789'

- How many smartphones are using the Snapdragon 810?

```
select COUNT(SoC)
from Smartphone
where SoC = 'Snapdragon 810';
```

**Result:**

'3'

# Using multiple tables

- ▶ It is possible to draw data from more than one table in a single query.
- ▶ In fact, it is very common to have several, comma separated tables listed in FROM.
- ▶ In WHERE columns from the different tables are combined with AND to filter out the data.
- ▶ An example, what is the name of the CEO for the company that makes the HTC One M9?

```
select CEO
from Company, Smartphone
where Company.Name = Smartphone.Maker
      and Smartphone.Maker = 'HTC'
      and Smartphone.Model = 'One M9';
```

- ▶ Result:  
'Fred Liu'

## Another example

- ▶ What smartphones are using “Android” as the OS?

```
select Smartphone.Maker, Smartphone.Model, MobileOS.OS
from Smartphone, MobileOS
where Smartphone.Model = MobileOS.Smartphone
      and MobileOS.OS = 'Android';
```

- ▶ Svar:

```
'LG', 'G Flex 2', 'Android'
'LG', 'G3', 'Android'
'Samsung', 'Galaxy S6 Edge', 'Android'
'Meizu', 'MX4', 'Android'
'LG', 'Nexus 5', 'Android'
'Motorola', 'Nexus 6', 'Android'
'HTC', 'One (M8)', 'Android'
'HTC', 'One M9', 'Android'
'Sony', 'Xperia Z3', 'Android'
'Sony', 'Xperia Z3 Compact', 'Android'
'Sony', 'Xperia Z4', 'Android'
```



## New example

- Show the name on the manufacturer, model, OS and version of the smartphones that can run “Ubuntu Touch”.

```
select Smartphone.Maker, Smartphone.Model, MobileOS.OS,  
        OS.Version  
from Smartphone, MobileOS, OS  
where Smartphone.Model = MobileOS.Smartphone  
    and MobileOS.OS = 'Ubuntu Touch'  
    and OS.Name = 'Ubuntu Touch';
```

- Result:

```
'bq', 'Aquaris E4.5', 'Ubuntu Touch', '14.1'  
'Meizu', 'MX4', 'Ubuntu Touch', '14.1'  
'LG', 'Nexus 5', 'Ubuntu Touch', '14.1'
```

## Yet another example

- ▶ Phones from HTC are sometimes made to run both Android and Windows Phone. Show what companies that make the OS for the different phones of HTC.

```
select Smartphone.Model, Company.Name
from Company, OS, MobileOS, Smartphone
where Smartphone.Maker = 'HTC'
      and MobileOS.Smartphone = Smartphone.Model
      and MobileOS.OS = OS.Name and OS.developer=Company.Name;
```

- ▶ Result:

```
'One (M8)', 'Google'
'One (M8)', 'Microsoft'
'One M9', 'Google'
```

# Joining tables

- ▶ The clause JOIN is used to combine two or more tables.
  - ▶ Which is what we have done in the previous slides, but this way is sometimes easier to follow.
- ▶ The tables are joined with one or more columns.
  - ▶ Choose one column in table one, and compare with all columns in table two.
  - ▶ This is also called *left join* and returns a new table (result set).
- ▶ There are several types of joins available, depending on how you like to compare the columns.
  - ▶ Apart from left, also right, inner and full join.

# An example

- What OS is each smartphone running?

```
select Maker, Model, MobileOS.OS
from Smartphone
join MobileOS on Smartphone.Model = MobileOS.Smartphone;
```

- Result (shortend due to space):

```
'Alcatel', 'Fire C', 'Firefox OS'
'Apple', 'iPhone 6', 'iOS'
'Apple', 'iPhone 6 Plus', 'iOS'
'Blackberry', 'Z10', 'Blackberry'
'Blackberry', 'Z30', 'Blackberry'
'bq', 'Aquaris E4.5', 'Ubuntu Touch'
'HTC', 'One (M8)', 'Android'
'HTC', 'One (M8)', 'Windows Phone'
'HTC', 'One M9', 'Android'
'Jolla', 'Phone', 'Sailfish'
'LG', 'G Flex 2', 'Android'
'LG', 'G3', 'Android'
'LG', 'Nexus 5', 'Android'
'LG', 'Nexus 5', 'Firefox OS'
'LG', 'Nexus 5', 'Ubuntu Touch'
```

## Additional examples

- Show the CEO of the company that sells Jolla Phone (which is the same as we did with HTC previously).

```
select CEO
from Company
join Smartphone on Smartphone.Maker = Company.Name
where Smartphone.Maker = 'Jolla'
      and Smartphone.Model = 'Phone';
```

- Result:  
'Tomi Pienimäki'

## Another example

- What phones can run “Ubuntu Touch”?

```
select Maker, Model
from Smartphone
join MobileOS on Smartphone.Model = MobileOS.Smartphone
join OS on OS.Name = MobileOS.OS
where MobileOS.OS = 'Ubuntu Touch'
      and OS.Name = 'Ubuntu Touch';
```

- Svar:

```
'bq', 'Aquaris E4.5'
'Meizu', 'MX4'
'LG', 'Nexus 5'
```

## A rewrite of the previous query

- Show all smartphones that run “Android”.

```
select Smartphone.Maker, Smartphone.Model
from MobileOS
join Smartphone on Smartphone.Model = MobileOS.Smartphone
join OS on OS.Name = MobileOS.OS
where OS.Name = 'Android';
```

- Result:

```
'LG', 'G Flex 2'
'LG', 'G3'
'Samsung', 'Galaxy S6 Edge'
'Meizu', 'MX4'
'LG', 'Nexus 5'
'Motorola', 'Nexus 6'
'HTC', 'One (M8)'
'HTC', 'One M9'
'Sony', 'Xperia Z3'
'Sony', 'Xperia Z3 Compact'
'Sony', 'Xperia Z4'
```

## Nested questions

- ▶ It is possible to have the result from one query as the input to another query.
- ▶ This is a nested query or subquery.
- ▶ The result from the *inner* query is either a single value (scalar), a single row or a new table.
  - ▶ For the last type, the predicate IN must be used.
- ▶ An example, show all phones from manufacturers with a revenue larger than 100 USD.

```
select Smartphone.Model
from Smartphone
where Smartphone.Maker in (
    select Company.Name
    from Company
    where revenue > 100);
```

### Result:

```
'iPhone 6'
'iPhone 6 Plus'
'G Flex 2'
'G3'
'Nexus 5'
'Galaxy S6 Edge'
```



## Another example

- ▶ This subquery only returns a single value, a scalar value, that can be compared as usually.
- ▶ Show all smartphones with a screen size larger than average.

```
select Smartphone.Model
from Smartphone
where Smartphone.ScreenSize > (
    select avg(Smartphone.ScreenSize)
    from Smartphone);
```

- ▶ Result (shortend due to space):

```
'G Flex 2'
'G3'
'Galaxy S6 Edge'
'iPhone 6 Plus'
'Lumia 535'
'Lumia 930'
'MX4'
'Nexus 6'
'One (M8)'
'One M9'
```

## Yet another example

- ▶ It is also possible to negate the inner query.
- ▶ Show the name of the companies that do not create smartphones (that is, the companies creating the OS).

```
select Company.Name  
from Company  
where Company.Name not in (  
    select Smartphone.Maker  
    from Smartphone);
```

- ▶ Result:  
    'Canonical'  
    'Google'  
    'Mozilla'

UNCATEGORISED

## Some additional things...

- ▶ There are a few additional things that are good and useful to know of.
- ▶ They are presented here in no particular way...
- ▶ Also noticed that there are many, many, many things that we have not touched that are part of both SQL and using databases.
  - ▶ You can find most of it on the Internet.
  - ▶ w3schools is a good starting point.
- ▶ Now, here are the final points.

# Filtering out doubles

- ▶ Use DISTINCT with SELECT to filter out any doubles that might show in the result.
- ▶ Show what operating systems are available.

```
select distinct OS  
from OS;
```

- ▶ Result:

```
'Firefox OS'  
'iOS'  
'Blackberry'  
'Ubuntu Touch'  
'Android'  
'Windows Phone'  
'Sailfish'
```

# Renaming columns

- ▶ Sometimes the names of the columns are not the easiest to read.
- ▶ Use AS in SELECT to rename a column.
- ▶ For example, show manufacturer, chip and screen size in a readable way for all phones larger than 5.3”.

```
select Maker, Model, SoC as 'System on Chip',
       ScreenSize AS 'Screen Size'
from Smartphone as sp
where sp.ScreenSize > 5.3;
```

| # | Maker    | Model         | System on Chip  | Screen Size |
|---|----------|---------------|-----------------|-------------|
| 1 | LG       | G Flex 2      | Snapdragon 810  | 5.5         |
| 2 | LG       | G3            | Snapdragon 801  | 5.5         |
| 3 | Apple    | iPhone 6 Plus | Apple A8        | 5.5         |
| 4 | Meizu    | MX4           | MediaTek MT6595 | 5.36        |
| 5 | Motorola | Nexus 6       | Snapdragon 805  | 5.96        |

# Concatenating columns

- ▶ To make it even easier to read, it is possible to concatenate (merge) columns and give them a new name.
- ▶ Use CONCAT for the merging and add an AS to rename it.
- ▶ An example, show all Windows Phones in one column with both manufacturer and model.

```
select concat(Maker, ' ', Model) as Phone
from Smartphone
join MobileOS on MobileOS.Smartphone = Smartphone.Model
where MobileOS.OS = 'Windows Phone';
```

| # | Phone               |
|---|---------------------|
| 1 | Microsoft Lumia 535 |
| 2 | Microsoft Lumia 930 |
| 3 | HTC One (M8)        |

# Grouping

- ▶ When using aggregate functions the result is often a number and often it is interesting to group by this number.
- ▶ An example could be to show the number of phones each operating system has in the database:

```
select MobileOS.OS, count(MobileOS.OS)
from MobileOS
join Smartphone on Smartphone.Model = MobileOS.Smartphone
group by MobileOS.OS;
```

- ▶ Result:

```
'Android', '12'
'Blackberry', '2'
'Firefox OS', '2'
'iOS', '2'
'Sailfish', '1'
'Ubuntu Touch', '4'
'Windows Phone', '3'
```



## Even more filtering

- ▶ The keyword HAVING is often used together with GROUP BY to filter even more.
- ▶ HAVING must always be placed after GROUP BY as this defines what needs to be grouped.
- ▶ To filter the previous question to only show operating systems with more than three phones, write:

```
select MobileOS.OS, count(MobileOS.OS)
from MobileOS
join Smartphone on Smartphone.Model = MobileOS.Smartphone
group by MobileOS.OS
having count(*) >= 3;
```

- ▶ Result:

```
'Android', '12'
'Ubuntu Touch', '4'
'Windows Phone', '3'
```

# The End

- ▶ This marks the end for this fast paced introduction to databases.
- ▶ For the projects you will need to use a database to store data, but it will probably not be too complex.
- ▶ As shown, the queries to a database can be fairly complex and you might feel compelled to just fetch the data and work with it in the programming language.
- ▶ In general, this is a bad idea – try to leave as much of the sorting and filtering to the database.
  - ▶ It is really good at it!
- ▶ So, later we will see how to connect to the database and send queries to it using Java.