Week 2 cliff notes

Our first task is to get comfortable with the database nomenclature. So the first part of class will be discussing the following terms (those that are online, please share your thoughts/findings in the Cobra Discussion forum):

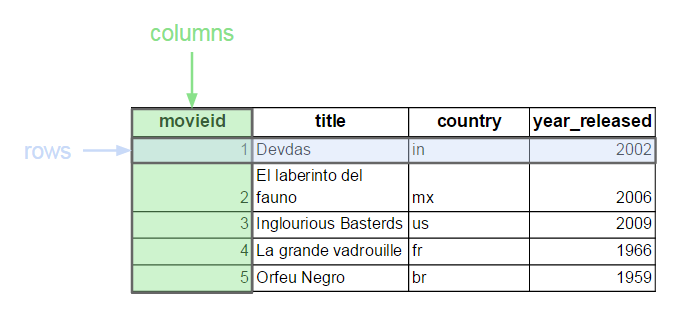
* database
* sql
* table
* row
* column
* field
* record
* constraint
* index
* relationships

Along the way, we will trip over things like RDBMS, and learn what role that plays in all this. The more you explore, the more you will learn. For those that like a quick overview, I have a Prezi demo:

<http://prezi.com/muerej0dzvqz/?utm_campaign=share&utm_medium=copy>

In my opinion, the easiest and simplest way to visualize a database table is to open a spreadsheet. Databases are of course more complex, but let’s start on square one.

I will reference both the [SQL Fiddle](http://sqlfiddle.com/#!9/fe3e8/1) I created for this lesson, copied to a [google spreadsheet](https://docs.google.com/spreadsheets/d/1OBsjTpm8ew0cvKbH88nWp_iEosfPKCFV6YBgVsYyodE/edit?usp=sharing); all the examples come from the [Konagora sandbox database](http://edu.konagora.com/SQLsandbox.php), so you can play around there as well. Let’s start with a table called “movies” - here is what a sample looks like:



This is a simple table.

It has 4 columns called “movieid”, “title”,”country”, “year\_released”. Note that the column names (ie, movieid) are not actually part of the data - they are just labels. Some people use the term “fields” to refer to columns. I do not.

This table also has 5 rows. Some people say “records” interchangeably with “rows”; for this class, I will always say “rows”. Again, the first “row” that contains the column names is not part of the data, it is merely shown here to explain what the column is. If you take the (real) first row cross reference with the column Movieid, you see the number 1. This is movieid 1. Given that, what is the title of the movie for movied 4? How many rows have a year\_released greater than 2000? (Guess what, I just asked you SQL questions. \*grin\*)

Let’s look at another table called “credits” (PS - if you right click, there are options for row, column and table)

|  |  |  |
| --- | --- | --- |
| **movieid** | **peopleid** | **credited\_as** |
| 1 | 75 | D |
| 1 | 229 | A |
| 1 | 470 | A |
| 1 | 719 | A |
| 2 | 969 | D |
| 2 | 552 | A |
| 2 | 884 | A |
| 2 | 48 | A |
| 3 | 903 | A |
| 3 | 698 | A |
| 3 | 858 | D |
| 3 | 486 | A |
| 3 | 502 | A |
| 4 | 966 | A |
| 4 | 91 | A |
| 4 | 671 | D |
| 5 | 116 | D |
| 5 | 606 | A |
| 5 | 205 | A |

This table has 3 columns? How many rows? You actually have to count them to figure out how many rows, right? If you are looking at the spreadsheet version, there are rownumbers on the right-hand side, so you know there are 19 rows. You will see that this time there are 4 rows for movieid=1. (How many rows for movieid=3?)

This table does not seem to make much sense all by itself. What is movieid? What is peopleid? What is credited\_as? The single most confusing thing about sql in modern relational databases are joining tables together (“R” in RDBMS is “relational”); this is a topic that will take a majority of our semester to master, so don’t worry too much about understanding it perfectly right now. The basic idea is that tables can be related to each other via a key, or a column with the same information. In this case, both the “movies” table and the “credits” table have a column called “movieid”. Thus you can figure out the information stored in both tables for any movie. For example, movieid=1:

|  |  |  |  |
| --- | --- | --- | --- |
| **movies** |  |  |  |
| **movieid** | **title** | **country** | **year\_released** |
| 1 | Devdas | in | 2002 |

|  |  |  |
| --- | --- | --- |
| **credits** |  |  |
| **movieid** | **peopleid** | **credited\_as** |
| 1 | 75 | D |
| 1 | 229 | A |
| 1 | 470 | A |
| 1 | 719 | A |

We can see that where the column “movieid” matches for each row (equals 1), the title is “Devdas”, in the country “in”, released in the year 2002, and has 4 peopleids (75, 229, 470 and 719), one of which is the “D” and 3 of which are “A”.

And yes, there is a “people” table that has names for those peopleids. We will go over that another week.

Think of how much easier all this would be if all that data were in one table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **movieid** | **peopleid** | **credited\_as** | **title** | **country** | **year\_released** |
| 1 | 75 | D | Devdas | in | 2002 |
| 1 | 229 | A | Devdas | in | 2002 |
| 1 | 470 | A | Devdas | in | 2002 |
| 1 | 719 | A | Devdas | in | 2002 |
| 2 | 969 | D | El laberinto del fauno | mx | 2006 |
| 2 | 552 | A | El laberinto del fauno | mx | 2006 |
| 2 | 884 | A | El laberinto del fauno | mx | 2006 |
| 2 | 48 | A | El laberinto del fauno | mx | 2006 |
| 3 | 903 | A | Inglourious Basterds | us | 2009 |
| 3 | 698 | A | Inglourious Basterds | us | 2009 |
| 3 | 858 | D | Inglourious Basterds | us | 2009 |
| 3 | 486 | A | Inglourious Basterds | us | 2009 |
| 3 | 502 | A | Inglourious Basterds | us | 2009 |
| 4 | 966 | A | La grande vadrouille | fr | 1966 |
| 4 | 91 | A | La grande vadrouille | fr | 1966 |
| 4 | 671 | D | La grande vadrouille | fr | 1966 |
| 5 | 116 | D | Orfeu Negro | br | 1959 |
| 5 | 606 | A | Orfeu Negro | br | 1959 |
| 5 | 205 | A | Orfeu Negro | br | 1959 |

There is a tradeoff that you have probably noticed by now. For instance, in the “one big table” approach, the columns called “title”, “country” and “year\_released” have a lot of duplicated data. So if you have a typo in the title of the movie, you have to change it 4 different places. And this is just a very simple example.

We have two more terms to explore before wrapping it all up with “database”. Both topics we cover in much more depth later in the semester.

**Index**

Most nonfiction books will have an index at the back to help you find which page a keyword appears on. Table indexes are exactly the same thing. If our “movies” table has a million rows (well, it actually only has 340), imagine trying to find movies released in the year 2000 by hand. One purpose of an index is to speed up searches (aka, queries). The other purpose is to constrain data.

**Constraint**

You can look up the dictionary.com [definition](http://dictionary.reference.com/browse/constraint), but to state it in “common language”, constraints make data fit into a box. This is extremely important for application development, so those of you wanting to use SQL in a job setting, take note. Constraints are required if you want to use a unique identifier for each row (typically called a “primary key” constraint, which is enforced by a unique index).

**Conclusion**

So what is a database? There are lots of “official” answers out on the interwebs, but let me paint you a picture.



[photo credit: <http://4.bp.blogspot.com/-m-ISjdQf7xs/UYHQlSpYeTI/AAAAAAAAIz0/z0ppP18PSwI/s1600/Playroom+Before+and+After.png>]

The purpose of a database is to make it easy to get data. So it includes all those things like tables, indexes and constraints to make it easy for you. And SQL is how you talk to a database.

RDBMS, by the way, stands for Relational Database Management System. The RDBMS manages all the other things wrapped around a database to make it sustainable; things like backups, controlling access and privileges, and other administrative tasks like changing table/index names.

**Bonus**

How would you draw a chart showing how many movies were released in each year? What tools would you use, what kind of image best conveys the most helpful information?

As one of many possible answers, I did a simple and quick Fusion Table:

<https://www.google.com/fusiontables/DataSource?docid=1pFVaP2xNOoy4os2IJLfz_yMMn93hk9mOKuMU86tZ>

( and yes, the data has a problem - can you spot it? )