

Databases

3. Multi-table queries

Arni Magnusson

United Nations University
Fisheries Training Programme

23–29 Oct 2015

Outline

What is a database

purpose, design, data types

Create database

software, import data

Query

get data, join tables, SQL language

Interface

connect to database from other program

Goals

After this five-day module, you should:

1. **Understand** what a database is, and how it works
2. Be able to **create** a simple database
3. Be able to **get data** from any database

Database design

How do we design tables?

Design rules

1. **Long format**, not crosstab
2. **Normalization**, by splitting tables

Design rules

1. **Long format**, not crosstab
2. **Normalization**, by splitting tables

In a nutshell:

Make tables as **narrow** as possible

Long format

1. Long format, not crosstab

Long format

Data tables like this:

Species	Year	Catch
Anchovy	2001	...
Anchovy	2002	...
Anchovy	2003	...
Barnacle	2001	...
Barnacle	2002	...
Barnacle	2003	...
Catfish	2001	...
Catfish	2002	...
Catfish	2003	...
Dogfish	2001	...
Dogfish	2002	...
Dogfish	2003	...

Not like this:

Year	Anchovy	Barnacle	Catfish	Dogfish
2001
2002
2003

Design rules

1. **Long format**, not crosstab
2. **Normalization**, by splitting tables

In a nutshell:

Make tables as narrow as possible

Normalization

2. Normalization, by splitting tables

Normalization

Remember our first table:

Name	Country	Capital	Siblings	Cars	Movie
...
...
...

Normalization

Remember our first table:

Name	Country	Capital	Siblings	Cars	Movie
...
...
...

How does it scale, if the table contains **7 billion** rows?

Normalization

Around 22 bytes per row:

Name	Country	Capital	Siblings	Cars	Movie
Short Text	Short Text	Short Text	Byte	Byte	Byte
~6	~7	~6	1	1	1

Normalization

Around 22 bytes per row:

Name	Country	Capital	Siblings	Cars	Movie
Short Text	Short Text	Short Text	Byte	Byte	Byte
~6	~7	~6	1	1	1

Our table is then 7 billion \times 22 \approx 150 GB

The names of countries and capitals are taking up too much space

Normalization

Split data into People and Countries:

Name	CountryID	Siblings	Cars	Movie
Short Text	Byte	Byte	Byte	Byte
~6	1	1	1	1

CountryID	Country	Capital
Byte	Short Text	Short Text
1	~7	~6

Normalization

Split data into People and Countries:

Name	CountryID	Siblings	Cars	Movie
Short Text	Byte	Byte	Byte	Byte
~6	1	1	1	1

CountryID	Country	Capital
Byte	Short Text	Short Text
1	~7	~6

7 billion \times 10 \approx 70 GB

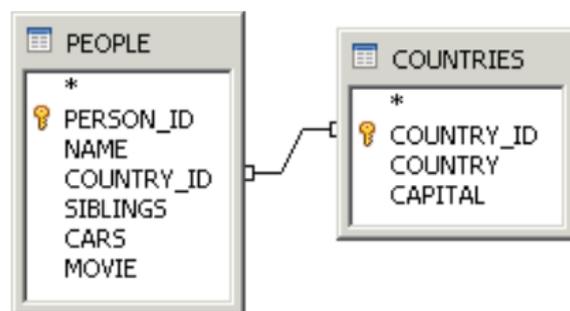
200 \times 14 = 0 GB

Normalization

One table



Joined tables



Normalization

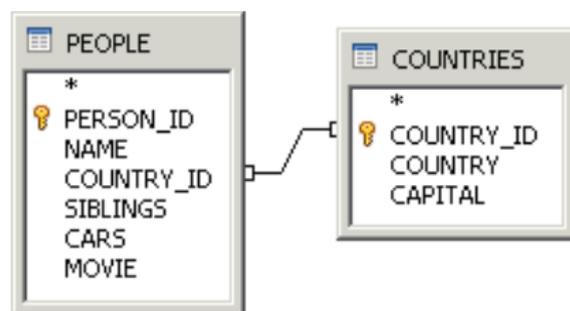
One table



redundant

risk of inconsistent data/mistakes
more work to enter data and modify
waste of storage
but convenient for tiny datasets

Joined tables



efficient

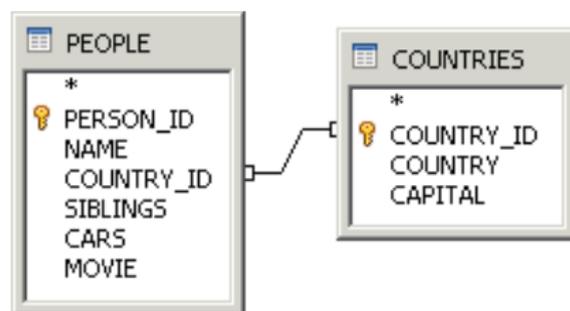
enforces consistent rules
less work to enter data and modify
compact storage
generally recommended

Normalization

One table



Joined tables



Splitting tables like this is called **normalizing**



An SQL query walks into a bar and sees two tables.

An SQL query walks into a bar and sees two tables.

He walks to them and says “Can I join you?”

Logbook data

Logbook data from Icelandic fisheries

Logbook data

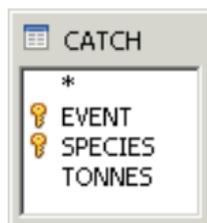


A screenshot of a database table named 'CATCH'. The table has three columns: 'EVENT', 'SPECIES', and 'TONNES'. The 'EVENT' and 'SPECIES' columns are marked with a key icon, indicating they are primary keys. The table is displayed in a window with a title bar that says 'CATCH'.

EVENT	SPECIES	TONNES
-------	---------	--------

Logbook data

```
SELECT sum(tonnes) AS total  
FROM catch
```



CATCH		
*		
EVENT		
SPECIES		
TONNES		

Logbook data



CATCH		
*		
EVENT		
SPECIES		
TONNES		

```
SELECT sum(tonnes) AS total  
FROM catch
```

```
SELECT species,  
       sum(tonnes) AS total  
FROM catch  
GROUP BY species  
ORDER BY species
```

Logbook data



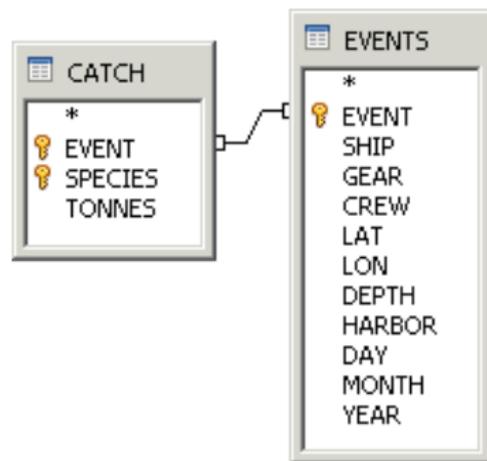
CATCH		
*		
EVENT		
SPECIES		
TONNES		

```
SELECT sum(tonnes) AS total  
FROM catch
```

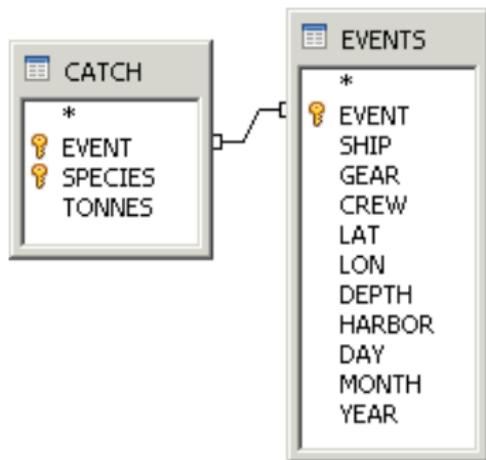
```
SELECT species,  
       sum(tonnes) AS total  
FROM catch  
GROUP BY species  
ORDER BY species
```

```
SELECT species,  
       max(tonnes) AS highscore  
FROM catch  
GROUP BY species  
ORDER BY species
```

Logbook data

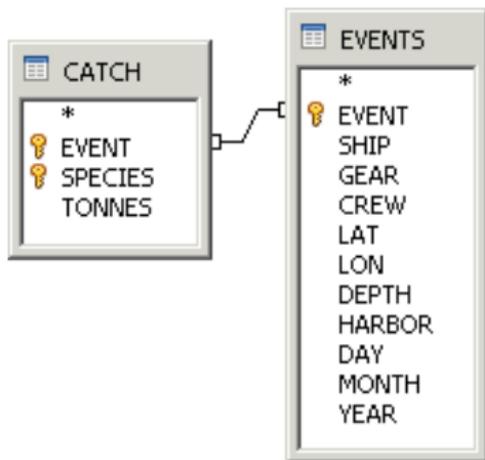


Logbook data



```
SELECT ship,  
       sum(tonnes) AS total  
FROM   catch c,  
       events e  
WHERE  c.event = e.event  
GROUP BY ship  
ORDER BY ship
```

Logbook data



```
SELECT ship,  
       sum(tonnes) AS total  
FROM   catch c,  
       events e  
WHERE  c.event = e.event  
GROUP BY ship  
ORDER BY ship
```

```
SELECT gear,  
       sum(tonnes) AS total  
FROM   catch c,  
       events e  
WHERE  c.event = e.event  
GROUP BY gear  
ORDER BY gear
```

Multi-table queries

How do we query many tables?

Equijoin

The expression

`WHERE table1.id = table2.id`

is an **equijoin**, which is the simplest join type

Equijoin

The expression

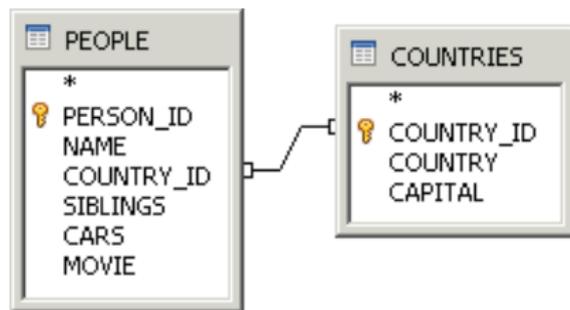
`WHERE table1.id = table2.id`

is an **equijoin**, which is the simplest join type

This is equivalent to

`WHERE table2.id = table1.id`

Table relationships



Most joins represent a
one-to-many table relationship
which is equivalent to **many-to-one**

This means that on one side of the join,
the column has only **unique** values

Table relationships



Table relationships

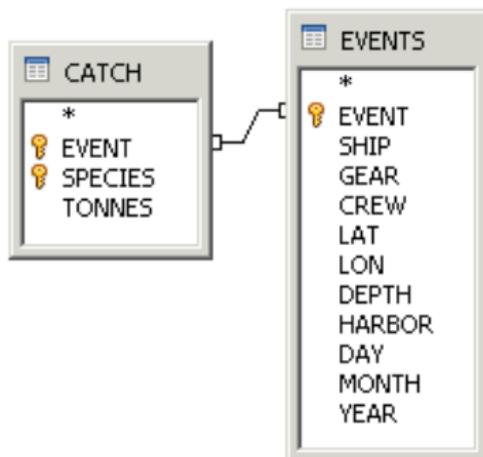
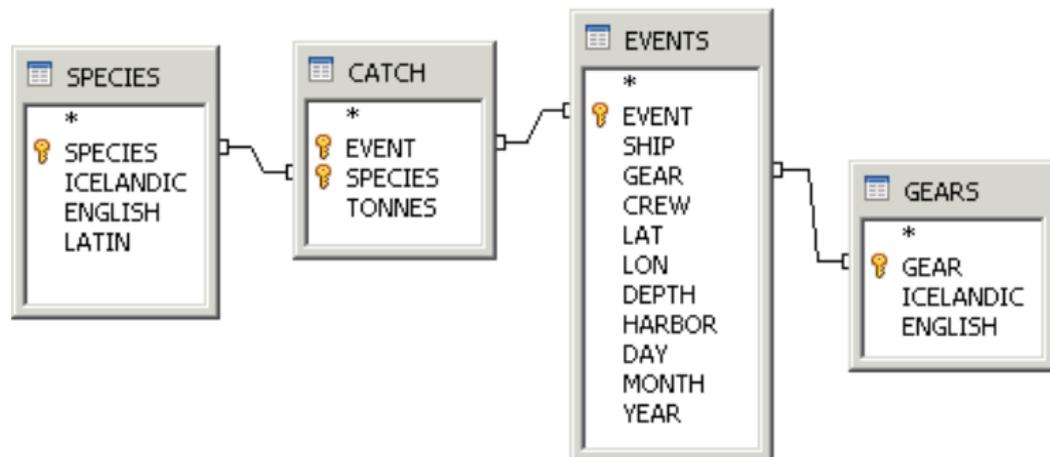


Table relationships



In what gear is saithe mainly caught?

SELECT

```
g.english AS gearname,  
sum(tonnes) AS total
```

FROM

```
catch c,  
events e,  
gears g,  
species s
```

WHERE

```
c.species = s.species AND  
c.event = e.event AND  
e.gear = g.gear AND  
s.english = 'Saithe'
```

GROUP BY

```
g.english
```

Table relationships

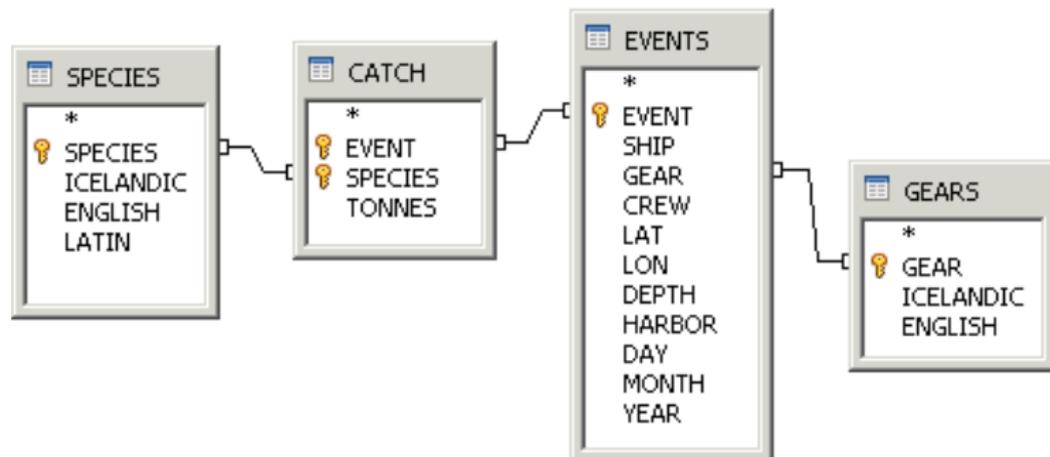
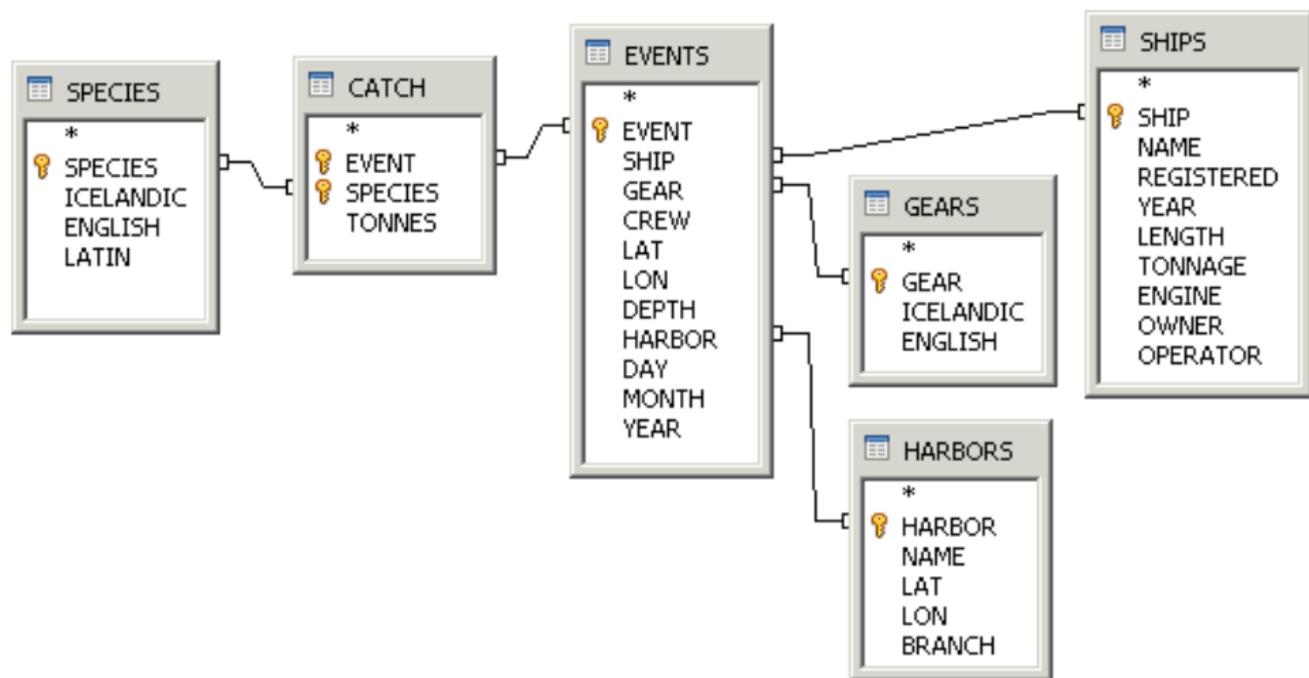


Table relationships



Postprocessing query results

What do we do with the query results?

Postprocessing query results

A query is just the first step

The next step is to **analyze**, create **plots** and summary **tables**

This is done **outside** the database, maybe in a spreadsheet or R

Postprocessing query results

A query is just the first step

The next step is to **analyze**, create **plots** and summary **tables**

This is done **outside** the database, maybe in a spreadsheet or R

It is often convenient to **run a simple query** and then do **calculations afterwards** in your preferred statistical software

Long format vs. crosstab

Data tables like this:

Species	Year	Catch
Anchovy	2001	...
Anchovy	2002	...
Anchovy	2003	...
Barnacle	2001	...
Barnacle	2002	...
Barnacle	2003	...
Catfish	2001	...
Catfish	2002	...
Catfish	2003	...
Dogfish	2001	...
Dogfish	2002	...
Dogfish	2003	...

Not like this:

Year	Anchovy	Barnacle	Catfish	Dogfish
2001
2002
2003

Crosstab

Year	Anchovy	Barnacle	Catfish	Dogfish
2001
2002
2003

Cross tabulation is great for **viewing**, but **not** for storing data

Crosstab

Year	Anchovy	Barnacle	Catfish	Dogfish
2001
2002
2003

Cross tabulation is great for **viewing**, but **not for storing data**

Not part of standard SQL, but query results can be crosstabbed afterwards:

- **Pivot table** in a spreadsheet
- **xtabs** in R

Avoid slow queries

A simple query can sometimes take a long time to compute

This should be avoided, especially on a **multi-user** database system

Avoid slow queries

A simple query can sometimes take a long time to compute

This should be avoided, especially on a **multi-user** database system

To make a query run fast, use

```
WHERE x = value AND  
      y LIKE '%pattern%' AND  
      z IN (value1,value2,value3)
```

to return only the subset that you're interested in