A (short) Introduction to Databases

(v.145 2015-04-22)

Nicola Bernardini
n.bernardini@conservatoriosantacecilia.it

"S.Cecilia" Conservatory - Rome

April 22 2015
Service Systems Design Master
Aalborg University in Copenhagen
Copenhagen, Denmark

Topics we will cover

- Just a touch of history
- Why do we need databases
- What are they, anyway?
- How do they work (in a very basic form)
- Exercising database design

Topics we will cover

- Just a touch of history
- Why do we need databases
- What are they, anyway?
- How do they work (in a very basic form)
- Exercising database design

Topics we will cover

- Just a touch of history
- Why do we need databases
- What are they, anyway?
- How do they work (in a very basic form)
- Exercising database design

Topics we will cover

- Just a touch of history
- Why do we need databases
- What are they, anyway?
- How do they work (in a very basic form)
- Exercising database design

Topics we will cover

- Just a touch of history
- Why do we need databases
- What are they, anyway?
- How do they work (in a very basic form)
- Exercising database design

Topics we will cover

- Just a touch of history
- Why do we need databases
- What are they, anyway?
- How do they work (in a very basic form)
- Exercising database design

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
    - search algorithms
    - data indexing
    - complex multi-table queries
    - sophisticated database functions
    - . . .

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
  - search algorithms
  - data indexing
  - complex multi-table queries
  - sophisticated database functions
  - ...

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
  - search algorithms
  - data indexing
  - complex multi–table queries
  - sophisticated database functions
  - ...

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
    - search algorithms
    - data indexing
    - complex multi–table queries
    - sophisticated database functions
    - ...

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
  - search algorithms
  - data indexing
  - complex multi–table queries
  - sophisticated database functions
  - ...

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
  - search algorithms
  - data indexing
  - complex multi–table queries
  - sophisticated database functions
  - . . .

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
  - search algorithms
  - data indexing
  - complex multi–table queries
  - sophisticated database functions
  - . . .

Topics we will *NOT* cover

(but feel free to ask about them if you feel so inclined, by any means :-))

- Advanced topics, such as:
  - search algorithms
  - data indexing
  - complex multi–table queries
  - sophisticated database functions
  - ...

What are computers for? (1)

Figure: A Leibnitz computing machine

- Back in the old days (I mean the eighteenth century). . .
- . . . computers were conceived and built for calculus and computation
- Mathematicians found out that reality was quite more complicate than the abstract models they had conceived. . .
- . . . so they invented "computing technologies", that is **computers**

What are computers for? (1)

Figure: A Leibnitz computing machine

- Back in the old days (I mean the eighteenth century). . .
- . . . computers were conceived and built for calculus and computation
- Mathematicians found out that reality was quite more complicate than the abstract models they had conceived. . .
- . . . so they invented “computing technologies”, that is computers

What are computers for? (1)

Figure: A Leibnitz computing machine

- Back in the old days (I mean the eighteenth century). . .
- . . . computers were conceived and built for calculus and computation
- Mathematicians found out that reality was quite more complicate than the abstract models they had conceived. . .
- . . . so they invented "computing technologies", that is computers

What are computers for? (1)

Figure: A Leibnitz computing machine

- Back in the old days (I mean the eighteenth century). . .
- . . . computers were conceived and built for calculus and computation
- Mathematicians found out that reality was quite more complicate than the abstract models they had conceived. . .
- . . . so they invented “computing technologies”, that is computers

What are computers for? (1)

Figure: A Leibnitz computing machine

- Back in the old days (I mean the eighteenth century). . .
- . . . computers were conceived and built for calculus and computation
- Mathematicians found out that reality was quite more complicate than the abstract models they had conceived. . .
- . . . so they invented “computing technologies”, that is computers

What are computers for? (1)

Figure: A Leibnitz computing machine

- Back in the old days (I mean the eighteenth century). . .
- . . . computers were conceived and built for calculus and computation
- Mathematicians found out that reality was quite more complicate than the abstract models they had conceived. . .
- . . . so they invented "computing technologies", that is **computers**

What are computers for? (2)

Figure: An IBM 7094 (1962)

- The last time computers were used *solely* for computing purposes was during World War II
- At the end of WW II, computing technology was mature enough to enter the industrial world. . .
- . . . and indeed, the computing capability of computers faded into the background

What are computers for? (2)

Figure: An IBM 7094 (1962)

- The last time computers were used *solely* for computing purposes was during World War II
- At the end of WW II, computing technology was mature enough to enter the industrial world...
- ...and indeed, the computing capability of computers faded into the background

What are computers for? (2)

Figure: An IBM 7094 (1962)

- The last time computers were used *solely* for computing purposes was during World War II
- At the end of WW II, computing technology was mature enough to enter the industrial world. . .
- . . . and indeed, the computing capability of computers faded into the background

What are computers for? (2)

Figure: An IBM 7094 (1962)

- The last time computers were used *solely* for computing purposes was during World War II
- At the end of WW II, computing technology was mature enough to enter the industrial world. . .
- . . . and indeed, the computing capability of computers faded into the background

What are computers for? (2)

Figure: An IBM 7094 (1962)

- The last time computers were used *solely* for computing purposes was during World War II
- At the end of WW II, computing technology was mature enough to enter the industrial world. . .
- . . . and indeed, the computing capability of computers faded into the background

What are computers for? (3)

- Today, the most useful function of computers is. . .
- . . . their ability to **store and organize data**
- and this is where **databases** kick in.
- Computation is still used of course, but it is no longer the core function (it appears in several specific fields in what are now called "number crunching applications")

What are computers for? (3)

- Today, the most useful function of computers is. . .
- . . . their ability to **store and organize data**
- and this is where databases kick in.
- Computation is still used of course, but it is no longer the core function (it appears in several specific fields in what are now called "number crunching applications")

What are computers for? (3)

- Today, the most useful function of computers is. . .
- . . . their ability to **store and organize data**
- and this is where **databases kick in.**
- Computation is still used of course, but it is no longer the core function (it appears in several specific fields in what are now called "number crunching applications")

What are computers for? (3)

- Today, the most useful function of computers is. . .
- . . . their ability to **store and organize data**
- and this is where **databases kick in.**
- Computation is still used of course, but it is no longer the core function (it appears in several specific fields in what are now called "number crunching applications")

What are computers for? (3)

- Today, the most useful function of computers is. . .
- . . . their ability to **store and organize data**
- and this is where **databases kick in.**
- Computation is still used of course, but it is no longer the core function (it appears in several specific fields in what are now called "number crunching applications")

What are computers for? (3)

- Today, the most useful function of computers is. . .
- . . . their ability to **store and organize data**
- and this is where **databases kick in.**
- Computation is still used of course, but it is no longer the core function (it appears in several specific fields in what are now called "number crunching applications")

What are computers for? (3)

- Today, the most useful function of computers is. . .
- . . . their ability to **store and organize data**
- and this is where **databases kick in.**
- Computation is still used of course, but it is no longer the core function (it appears in several specific fields in what are now called "number crunching applications")

So why do we need databases? (1)

```
int i = 42;
```

- As you all know, computers structure (and store) information in RAM memory
- Data maps into memory straight memory
- Memory is characterized by a content and an address (pointer)
- An address can also be considered some form of content

So why do we need databases? (1)

```
int i = 42;
```

- As you all know, computers structure (and store) information in RAM memory
- Data maps into memory straight memory
- Memory is characterized by a content and an address (pointer)
- An address can also be considered some form of content

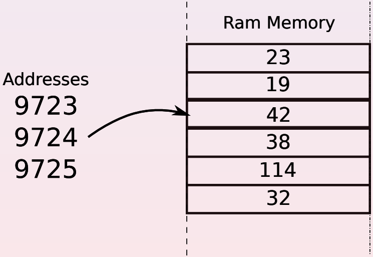
So why do we need databases? (1)

```
int i = 42;
```

- As you all know, computers structure (and store) information in RAM memory
- Data maps into memory straight memory
- Memory is characterized by a content and an address (pointer)
- An address can also be considered some form of content

So why do we need databases? (1)

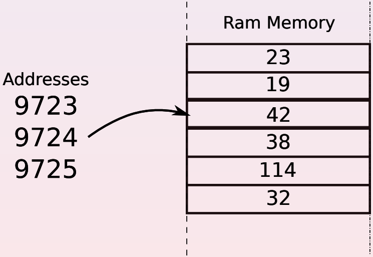
```
int i = 42;
```



- As you all know, computers structure (and store) information in RAM memory
- Data maps into memory straight memory
- Memory is characterized by a content and an address (pointer)
- An address can also be considered some form of content

So why do we need databases? (1)

```
int i = 42;
```



Ram Memory

23
19
42
38
114
32

Addresses
9723
9724
9725

- As you all know, computers structure (and store) information in RAM memory
- Data maps into memory straight memory
- Memory is characterized by a content and an address (pointer)
- An address can also be considered some form of content

So why do we need databases? (2)

```
struct birthday {
  int day;
  int month;
  int year;
};
```

- Furthermore, data in memory can be structured
- That is, it can be organized so that complex structures are kept physically together in RAM memory
- In the past few decades, these structures have become *object*, in object–oriented programming lingo

So why do we need databases? (2)

```
struct birthday {
  int day;
  int month;
  int year;
};
```

- Furthermore, data in memory can be structured
- That is, it can be organized so that complex structures are kept physically together in RAM memory
- In the past few decades, these structures have become *object*, in object–oriented programming lingo

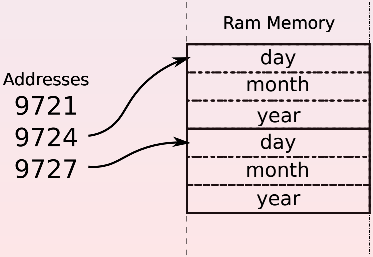
So why do we need databases? (2)

```
struct birthday {
  int day;
  int month;
  int year;
};
```

- Furthermore, data in memory can be structured
- That is, it can be organized so that complex structures are kept physically together in RAM memory
- In the past few decades, these structures have become *object*, in object–oriented programming lingo

So why do we need databases? (2)

```
struct birthday {
  int day;
  int month;
  int year;
};
```



- Furthermore, data in memory can be structured
- That is, it can be organized so that complex structures are kept physically together in RAM memory
- In the past few decades, these structures have become *object*, in object–oriented programming lingo

So why do we need databases? (3)

- However, data in RAM memory is not persistent
- When the computer shuts off, the data is gone
- So: data needs to be *persisted*
- Which translates into: databases

So why do we need databases? (3)

- However, data in RAM memory is not persistent
- When the computer shuts off, the data is gone
- So: data needs to be *persisted*
- Which translates into:

So why do we need databases? (3)

- However, data in RAM memory is not persistent
- When the computer shuts off, the data is gone
- So: data needs to be *persisted*
- Which translates into: databases

So why do we need databases? (3)

- However, data in RAM memory is not persistent
- When the computer shuts off, the data is gone
- So: data needs to be *persisted*
- Which translates into: databases

So why do we need databases? (3)

- However, data in RAM memory is not persistent
- When the computer shuts off, the data is gone
- So: data needs to be *persisted*
- Which translates into: databases

So why do we need databases? (3)

- However, data in RAM memory is not persistent
- When the computer shuts off, the data is gone
- So: data needs to be *persisted*
- Which translates into: **databases**

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text
    (f. ex. Nicola;Bernardini;Rome;14/08/1956;+4512345678)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text (f. ex. Nicola;Bernardini;Rome;14/08/1956;+4512345678)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text (f. ex. Nicola;Bernardini;Rome;14/08/1956;+4512345678)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text (f. ex. Nicola;Bernardini;Rome;14/08/1956;+4512345678)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text (f. ex. `Nicola;Bernardini;Rome;14/08/1956;+4512345678`)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text
    (f. ex. `Nicola;Bernardini;Rome;14/08/1956;+4512345678`)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text (f. ex. `Nicola;Bernardini;Rome;14/08/1956;+4512345678`)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

How many different kinds of databases are there?

- You can make databases (that is: save your data) with just about anything
  - log files
  - line oriented delimited text (f. ex. `Nicola;Bernardini;Rome;14/08/1956;+4512345678`)
  - spreadsheets
  - disk file systems
  - proper relational database
- only the last two do not save only *data* but also *relations*

What are relations for? (1)

- If you start collecting data you soon realize that
  - a) either you repeat parts of the data endlessly
  - b) or you have to save *data relations* along with the *data* itself
- and here we get to the **golden rule of databases**, namely. . .

What are relations for? (1)

- If you start collecting data you soon realize that

  a) either you repeat parts of the data endlessly
  b) or you have to save *data relations* along with the *data* itself

- and here we get to the **golden rule of databases**, namely. . .

What are relations for? (1)

- If you start collecting data you soon realize that
  a) either you repeat parts of the data endlessly
  b) or you have to save *data relations* along with the *data* itself
- and here we get to the **golden rule of databases**, namely. . .

What are relations for? (1)

- If you start collecting data you soon realize that
  a) either you repeat parts of the data endlessly
  b) or you have to save *data relations* along with the *data* itself
- and here we get to the **golden rule of databases**, namely. . .

What are relations for? (1)

- If you start collecting data you soon realize that
  a) either you repeat parts of the data endlessly
  b) or you have to save *data relations* along with the *data* itself
- and here we get to the **golden rule of databases**, namely...

The Golden Rule of Databases

DO NOT REPLICATE DATA.

Rationale for the golden rule

- You don't want to replicate data because you might need to change it in the future: if your data is replicated all over, you need to change it all over (very error-prone)
- while if you keep relationships instead, a single change will be sufficient for each piece of datum you have

Rationale for the golden rule

- You don't want to replicate data because you might need to change it in the future: if your data is replicated all over, you need to change it all over (very error-prone)
- while if you keep relationships instead, a single change will be sufficient for each piece of datum you have

Rationale for the golden rule

- You don’t want to replicate data because you might need to change it in the future: if your data is replicated all over, you need to change it all over (very error-prone)
- while if you keep relationships instead, a single change will be sufficient for each piece of datum you have

Rationale for the golden rule

- You don't want to replicate data because you might need to change it in the future: if your data is replicated all over, you need to change it all over (very error-prone)
- while if you keep relationships instead, a single change will be sufficient for each piece of datum you have

What are relations for? (2)

- This is what relations are for. For example:
  - If you have one *or more* telephone numbers for each person in your agenda (which is likely, or make it e-mail addresses, or whatever)
  - You need to be able to express something like “a person can *have many phone numbers*” or “this number *belongs to* this person”
  - Both sentences basically establish a *one–way relation* between persons and phone numbers

What are relations for? (2)

- This is what relations are for. For example:
  - If you have one *or more* telephone numbers for each person in your agenda (which is likely, or make it e-mail addresses, or whatever)
  - You need to be able to express something like *“a person can have many telphone numbers”* or like *“this number belongs to this person”*
  - Both sentences basically establish *a one–way relation* between persons and phone numbers

What are relations for? (2)

- This is what relations are for. For example:
  - If you have one *or more* telephone numbers for each person in your agenda (which is likely, or make it e-mail addresses, or whatever)
  - You need to be able to express something like "a person *can have many* telphone numbers" or like "this number *belongs to* this person"
  - Both sentences basically establish *a one–way relation* between persons and phone numbers

What are relations for? (2)

- This is what relations are for. For example:
  - If you have one *or more* telephone numbers for each person in your agenda (which is likely, or make it e-mail addresses, or whatever)
  - You need to be able to express something like “a person *can have many* telphone numbers” or like “this number *belongs to* this person”
  - Both sentences basically establish *a one–way relation* between persons and phone numbers

What are relations for? (2)

- This is what relations are for. For example:
  - If you have one *or more* telephone numbers for each person in your agenda (which is likely, or make it e-mail addresses, or whatever)
  - You need to be able to express something like "a person *can have many* telphone numbers" or like "this number *belongs to* this person"
  - Both sentences basically establish *a one–way relation* between persons and phone numbers

What are relations for? (2)

- This is what relations are for. For example:
  - If you have one *or more* telephone numbers for each person in your agenda (which is likely, or make it e-mail addresses, or whatever)
  - You need to be able to express something like “a person *can have many* telphone numbers” or like “this number *belongs to* this person”
  - Both sentences basically establish *a one–way relation* between persons and phone numbers

What are relations for? (2)

- This is what relations are for. For example:
  - If you have one *or more* telephone numbers for each person in your agenda (which is likely, or make it e-mail addresses, or whatever)
  - You need to be able to express something like “a person *can have many* telphone numbers” or like “this number *belongs to* this person”
  - Both sentences basically establish *a one–way relation* between persons and phone numbers

How do you represent relationships inside computers? (1)

```
struct person {
  char first_name[256];
  char last_name[256];
};

struct number {
  struct person *owner; /* <- */
  int number;
};

struct person persons[1000];
struct number numbers[10000];
```

- When your data is stored in RAM. . .
- . . . you represent relationships by *memory pointers* (also known as *references*) (check the arrow)
- but what happens when you want to *persist* the relationship? (== save it to disk)
- this (and some more) is what relational databases are for

How do you represent relationships inside computers? (1)

```
struct person {
  char first_name[256];
  char last_name[256];
};

struct number {
  struct person *owner; /* <- */
  int number;
};

struct person persons[1000];
struct number numbers[10000];
```

- When your data is stored in RAM. . .
- . . . you represent relationships by *memory pointers* (also known as *references*) (check the arrow)
- but what happens when you want to *persist* the relationship? (== save it to disk)
- this (and some more) is what relational databases are for

How do you represent relationships inside computers? (1)

```
struct person {
  char first_name[256];
  char last_name[256];
};

struct number {
  struct person *owner; /* <- */
  int number;
};

struct person persons[1000];
struct number numbers[10000];
```

- When your data is stored in RAM. . .
- . . . you represent relationships by *memory pointers* (also known as *references*) (check the arrow)
- but what happens when you want to *persist* the relationship? (== save it to disk)
- this (and some more) is what relational databases are for

How do you represent relationships inside computers? (1)

```
struct person {
  char first_name[256];
  char last_name[256];
};

struct number {
  struct person *owner; /* <- */
  int number;
};

struct person persons[1000];
struct number numbers[10000];
```

- When your data is stored in RAM...
- ...you represent relationships by *memory pointers* (also known as *references*) (check the arrow)
- but what happens when you want to *persist* the relationship? (== save it to disk)
- this (and some more) is what relational databases are for

How do you represent relationships inside computers? (1)

```
struct person {
  char first_name[256];
  char last_name[256];
};

struct number {
  struct person *owner; /* <- */
  int number;
};

struct person persons[1000];
struct number numbers[10000];
```

- When your data is stored in RAM. . .
- . . . you represent relationships by *memory pointers* (also known as *references*) (check the arrow)
- but what happens when you want to *persist* the relationship? (== save it to disk)
- this (and some more) is what relational databases are for

How do you represent relationships inside computers? (2)

Before we go on, a question for you.

How would you create a persistent telephone book. . .
with a filesystem?

How do you represent relationships inside computers? (2)

Before we go on, a question for you.

How would you create a persistent telephone book. . .
with a filesystem?

How do you represent relationships inside computers? (2)

Before we go on, a question for you.

How would you create a persistent telephone book...
with a filesystem?

How do you represent relationships inside computers? (3)

- Current relational database can do much more than that
- Within databases, you
  - use a progressive number [illegible]
  - this number is guaranteed (by the database engine) to be unique
  - it is called the primary key
- Primary keys act like persistent pointers

How do you represent relationships inside computers? (3)

- Current relational database can do much more than that
- Within databases, you
  - add a progressive number to each data structure
  - this number is guaranteed (by the database engine) to be unique
  - it is called the primary key
- Primary keys act like persistent pointers

How do you represent relationships inside computers? (3)

- Current relational database can do much more than that
- Within databases, you
  - add a progressive number to each data structure
  - this number is guaranteed (by the database engine) to be unique
  - it is called the *primary key*
- Primary keys act like persistent pointers

How do you represent relationships inside computers? (3)

- Current relational database can do much more than that
- Within databases, you
  - add a progressive number to each data structure
  - this number is guaranteed (by the database engine) to be unique
  - it is called the *primary key*
- Primary keys act like persistent pointers

How do you represent relationships inside computers? (3)

- Current relational database can do much more than that
- Within databases, you
  - add a progressive number to each data structure
  - this number is guaranteed (by the database engine) to be unique
  - it is called the *primary key*
- Primary keys act like persistent pointers

How do you represent relationships inside computers? (3)

- Current relational database can do much more than that
- Within databases, you
  - add a progressive number to each data structure
  - this number is guaranteed (by the database engine) to be unique
  - it is called the *primary key*
- Primary keys act like persistent pointers

How do you represent relationships inside computers? (3)

- Current relational database can do much more than that
- Within databases, you
  - add a progressive number to each data structure
  - this number is guaranteed (by the database engine) to be unique
  - it is called the *primary key*
- Primary keys act like persistent pointers

How do you represent relationships inside computers? (4)

- Relationships are expressed with further indexes, called *foreign keys*
- Theoreticians have established that with only *three* kinds of relationships you can describe any type of relation among data
- These relationships are. . .

How do you represent relationships inside computers? (4)

- Relationships are expressed with further indexes, called *foreign keys*
- Theoreticians have established that with only *three* kinds of relationships you can describe any type of relation among data
- These relationships are. . .

How do you represent relationships inside computers? (4)

- Relationships are expressed with further indexes, called *foreign keys*
- Theoreticians have established that with only *three* kinds of relationships you can describe any type of relation among data
- These relationships are. . .

How do you represent relationships inside computers? (4)

- Relationships are expressed with further indexes, called *foreign keys*
- Theoreticians have established that with only *three* kinds of relationships you can describe any type of relation among data
- These relationships are. . .

How do you represent relationships inside computers? (5)

- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many
- many–to–many

How do you represent relationships inside computers? (5)

- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many (f.ex: *a football team has many players*)
- many–to–many (f.ex: *a student has many teachers*)

How do you represent relationships inside computers? (5)

- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many (f.ex: *a football team has many players*)
- many–to–many

How do you represent relationships inside computers? (5)

- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many (f.ex: *a football team has many players*)
- many–to–many (f.ex: *a student has many subjects*)

How do you represent relationships inside computers? (5)

- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many (f.ex: *a football team has many players*)
- many–to–many (f.ex: *a student has many subjects*)

How do you represent relationships inside computers? (5)

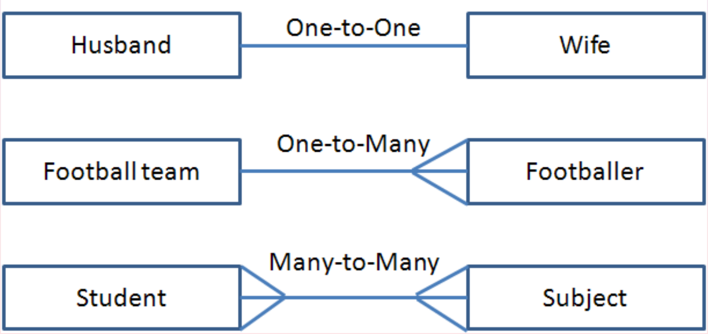
- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many (f.ex: *a football team has many players*)
- many–to–many (f.ex: *a student has many subjects*)

How do you represent relationships inside computers? (5)

- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many (f.ex: *a football team has many players*)
- many–to–many (f.ex: *a student has many subjects*)

How do you represent relationships inside computers? (5)

- one–to–one (f.ex: *a husband has only one wife*)
- one–to–many (f.ex: *a football team has many players*)
- many–to–many (f.ex: *a student has many subjects*)



One–to–one relationships

- you establish one–to–one relationships by:
  - creating a first table with an auto–generated primary key
  - creating a second table with a (*not* auto–generated) foreign key which points to an instance of the first table

One–to–one relationships

- you establish one–to–one relationships by:
    - creating a first table with an auto–generated primary key
    - creating a second table with a (*not* auto–generated) foreign key which points to an instance of the first table

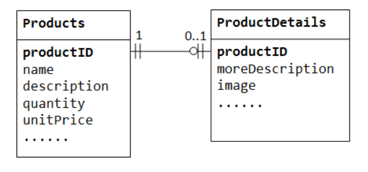
One–to–one relationships

- you establish one–to–one relationships by:
  - creating a first table with an auto–generated primary key
  - creating a second table with a (*not* auto–generated) foreign key which points to an instance of the first table

One–to–one relationships

- you establish one–to–one relationships by:
  - creating a first table with an auto–generated primary key
  - creating a second table with a (*not* auto–generated) foreign key which points to an instance of the first table

One–to–one relationships



Products

productID
name
description
quantity
unitPrice
......

ProductDetails

productID
moreDescription
image
......

- you establish one–to–one relationships by:
  - creating a first table with an auto–generated primary key
  - creating a second table with a (*not* auto–generated) foreign key which points to an instance of the first table

One–to–many relationships

- you establish one–to–many relationships by:
  - creating a first table with an auto–generated primary key
  - creating a second table with an auto–generated primary key *and* a foreign key which points to an instance of the first table

One–to–many relationships

- you establish one–to–many relationships by:
  - creating a first table with an auto–generated primary key
  - creating a second table with an auto–generated primary key *and* a foreign key which points to an instance of the first table

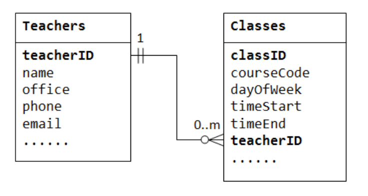
One–to–many relationships

- you establish one–to–many relationships by:
  - creating a first table with an auto–generated primary key
  - creating a second table with an auto–generated primary key *and* a foreign key which points to an instance of the first table

One–to–many relationships

- you establish one–to–many relationships by:
    - creating a first table with an auto–generated primary key
    - creating a second table with an auto–generated primary key *and* a foreign key which points to an instance of the first table

One–to–many relationships



- you establish one–to–many relationships by:
    - creating a first table with an auto–generated primary key
    - creating a second table with an auto–generated primary key *and* a foreign key which points to an instance of the first table

Many–to–many relationships

- you establish many–to–many relationships by:
    - creating two separate tables with an auto–generated primary keys
    - creating a *third* table with *two* foreign keys each pointing to one instance of one of the first two tables

Many–to–many relationships

- you establish many–to–many relationships by:
  - creating two separate tables with an auto–generated primary keys
  - creating *a third* table with *two* foreign keys each pointing to one instance of one of the first two tables

Many–to–many relationships

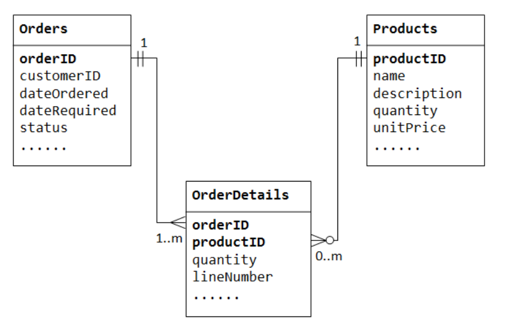
- you establish many–to–many relationships by:
    - creating two separate tables with an auto–generated primary keys
    - creating *a third* table with *two* foreign keys each pointing to one instance of one of the first two tables

Many–to–many relationships

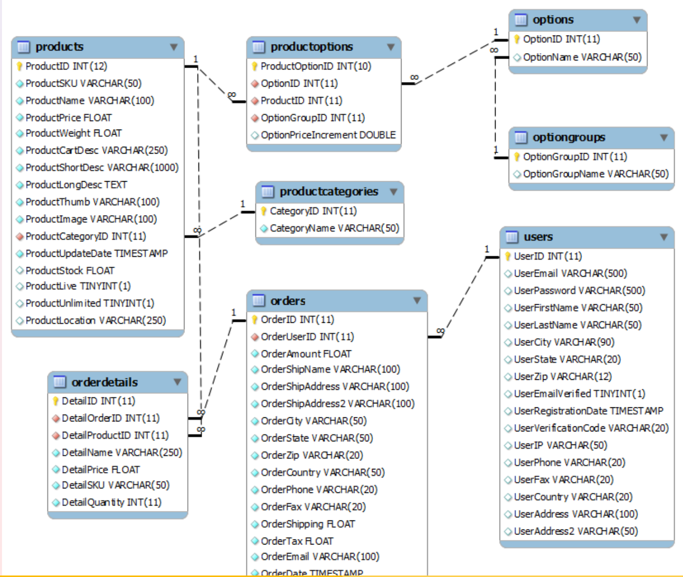
- you establish many–to–many relationships by:
  - creating two separate tables with an auto–generated primary keys
  - creating *a third* table with *two* foreign keys each pointing to one instance of one of the first two tables

Many–to–many relationships

- you establish many–to–many relationships by:
  - creating two separate tables with an auto–generated primary keys
  - creating *a third* table with *two* foreign keys each pointing to one instance of one of the first two tables



A real–world example



products
- ProductID INT(12)
- ProductSKU VARCHAR(50)
- ProductName VARCHAR(100)
- ProductPrice FLOAT
- ProductWeight FLOAT
- ProductCartDesc VARCHAR(250)
- ProductShortDesc VARCHAR(1000)
- ProductLongDesc TEXT
- ProductThumb VARCHAR(100)
- ProductImage VARCHAR(100)
- ProductCategoryID INT(11)
- ProductUpdateDate TIMESTAMP
- ProductStock FLOAT
- ProductLive TINYINT(1)
- ProductUnlimited TINYINT(1)
- ProductLocation VARCHAR(250)

productoptions
- ProductOptionID INT(10)
- OptionID INT(11)
- ProductID INT(11)
- OptionGroupID INT(11)
- OptionPriceIncrement DOUBLE

options
- OptionID INT(11)
- OptionName VARCHAR(50)

optiongroups
- OptionGroupID INT(11)
- OptionGroupName VARCHAR(50)

productcategories
- CategoryID INT(11)
- CategoryName VARCHAR(50)

users
- UserID INT(11)
- UserEmail VARCHAR(500)
- UserPassword VARCHAR(500)
- UserFirstName VARCHAR(50)
- UserLastName VARCHAR(50)
- UserCity VARCHAR(90)
- UserState VARCHAR(20)
- UserZip VARCHAR(12)
- UserEmailVerified TINYINT(1)
- UserRegistrationDate TIMESTAMP
- UserVerificationCode VARCHAR(20)
- UserIP VARCHAR(50)
- UserPhone VARCHAR(20)
- UserFax VARCHAR(20)
- UserCountry VARCHAR(20)
- UserAddress VARCHAR(100)
- UserAddress2 VARCHAR(50)

orders
- OrderID INT(11)
- OrderUserID INT(11)
- OrderAmount FLOAT
- OrderShipName VARCHAR(100)
- OrderShipAddress VARCHAR(100)
- OrderShipAddress2 VARCHAR(100)
- OrderCity VARCHAR(50)
- OrderState VARCHAR(50)
- OrderZip VARCHAR(20)
- OrderCountry VARCHAR(50)
- OrderPhone VARCHAR(20)
- OrderFax VARCHAR(20)
- OrderShipping FLOAT
- OrderTax FLOAT
- OrderEmail VARCHAR(100)
- OrderDate TIMESTAMP

orderdetails
- DetailID INT(11)
- DetailOrderID INT(11)
- DetailProductID INT(11)
- DetailName VARCHAR(250)
- DetailPrice FLOAT
- DetailSKU VARCHAR(50)
- DetailQuantity INT(11)

A structured query language (SQL) (1)

- Of course, structuring data is not enough to be able to use it efficiently
- You also need to have a way to create, browse, view, update, delete data
- Such a language exists: it is called *SQL* (spelled: *sequel* – which stands precisely for *Structured Query Language*)
- With minor changes, SQL can be used over a multitude of different databases (`sqlite`, `MySql`, `PostgreSQL`, `Oracle`, `Informix`, etc.)

A structured query language (SQL) (1)

- Of course, structuring data is not enough to be able to use it efficiently
- You also need to have a way to create, browse, view, update, delete data
- Such a language exists: it is called *SQL* (spelled: *sequel* – which stands precisely for *Structured Query Language*)
- With minor changes, SQL can be used over a multitude of different databases (`sqlite`, `MySql`, `PostgreSQL`, `Oracle`, `Informix`, etc.)

A structured query language (SQL) (1)

- Of course, structuring data is not enough to be able to use it efficiently
- You also need to have a way to create, browse, view, update, delete data
- Such a language exists: it is called *SQL* (spelled: *sequel* – which stands precisely for *Structured Query Language*)
- With minor changes, SQL can be used over a multitude of different databases (`sqlite`, `MySql`, `PostgreSQL`, `Oracle`, `Informix`, etc.)

A structured query language (SQL) (1)

- Of course, structuring data is not enough to be able to use it efficiently
- You also need to have a way to create, browse, view, update, delete data
- Such a language exists: it is called *SQL* (spelled: *sequel* – which stands precisely for *Structured Query Language*)
- With minor changes, SQL can be used over a multitude of different databases (`sqlite`, `MySql`, `PostgreSQL`, `Oracle`, `Informix`, etc.)

A structured query language (SQL) (1)

- Of course, structuring data is not enough to be able to use it efficiently
- You also need to have a way to create, browse, view, update, delete data
- Such a language exists: it is called *SQL* (spelled: *sequel* – which stands precisely for *Structured Query Language*)
- With minor changes, SQL can be used over a multitude of different databases (`sqlite`, `MySql`, `PostgreSQL`, `Oracle`, `Informix`, etc.)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, [illegible]
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”,
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE” and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE”, and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE” and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE” and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE” and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE” and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- "SELECT", "INSERT", "UPDATE", "DELETE", "CREATE" and "DROP"
- "SELECT" is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE” and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

A structured query language (SQL) (2)

- SQL is a text–based language: you type some instruction on a database console and the database will reply with an answer
- Of course it can also be scripted and put into a file
- SQL has several commonly used statements:
- “SELECT”, “INSERT”, “UPDATE”, “DELETE”, “CREATE” and “DROP”
- “SELECT” is the most commonly used, as it is the statement that allows you to make queries (it does not change the database)

The “SELECT” statement (1)

Here is what the “SELECT” statement looks like:

```
select "column1"
  [,"column2",etc]
  from "tablename"
  [where "condition"];
```

The “SELECT” statement (1)

Here is what the “SELECT” statement looks like:

```
select "column1"
  [,"column2",etc]
  from "tablename"
  [where "condition"];
```

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with 'Er' and display first name, last name and city of employee
  - `select * from employees where first = 'Eric';` find all first names that match the name 'Eric' exactly and display
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age and have a last name that ends in 's' and display last name, city and age
  - . . . let's do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
    - `select first, last, city from employees where first like 'Er%';` find all first names that begin with 'Er' and display first name, last name and city of operation
    - `select * from employees where first = 'Eric';` find all first names that match the name 'Eric' exactly and display [illegible]
    - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
    - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age and have a last name that ends in 's' and display last name, city and age
    - ...let's do some examples live...

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age and have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age and have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees who are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees who are older than 30 years of age and have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age and have a last name that ends with ’s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age and have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like ’Er%’;` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = ’Eric’;` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like ’%s’);` find all employees which are older than 30 years of age *and* have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age *and* have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age *and* have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The “SELECT” statement (2)

- Some “SELECT” examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with ‘Er’ and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name ‘Eric’ exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age *and* have a last name that ends in ‘s’ and display last name, city and age
  - . . . let’s do some examples live. . .

The "SELECT" statement (2)

- Some "SELECT" examples:
  - `select first, last, city from employees where first like 'Er%';` find all first names that begin with 'Er' and display first name, last name and city of operation
  - `select * from employees where first = 'Eric';` find all first names that match the name 'Eric' exactly and display all data from them
  - `select last, city, age from employees where age > 30;` find all employees which are older than 30 years of age and display last name, city and age
  - `select last, city, age from employees where (age > 30) and (last like '%s');` find all employees which are older than 30 years of age *and* have a last name that ends in 's' and display last name, city and age
  - . . . let's do some examples live. . .

Live examples

- Please download the employees.sqlite from Moodle
- If you have a Mac or a Linux laptop, you may open a terminal and type `sqlite3 employees.sqlite` in the folder where you downloaded the db
- if you have Windows, you may find a pre–compiled binary at `https://www.sqlite.org/download.html`

Walk-through exercise

- Let’s imagine reverse–engineering. . . FaceBook at least partially ;-)

Walk-through exercise

- Let's imagine reverse–engineering. . . FaceBook at least partially :-)

Walk-through exercise

- Let's imagine reverse–engineering. . . FaceBook at least partially :-)

Walk-through exercise

- Let’s imagine reverse–engineering. . . FaceBook at least partially :-)

DIY Exercise (30 minutes)

1. Pick your own data set and/or application
2. Design it in terms of a relational database:
    1. Establish tables
    2. Establish relationships among tables, making sure that no data is replicated
3. Discuss your design

DIY Exercise (30 minutes)

1. Pick your own data set and/or application
2. Design it in terms of a relational database:
    1. Establish tables
    2. Establish relationships among tables, making sure that no data is replicated
3. Discuss your design

DIY Exercise (30 minutes)

1. Pick your own data set and/or application
2. Design it in terms of a relational database:
    1. Establish tables
    2. Establish relationships among tables, making sure that no data is replicated
3. Discuss your design

DIY Exercise (30 minutes)

1. Pick your own data set and/or application
2. Design it in terms of a relational database:
    1. Establish tables
    2. Establish relationships among tables, making sure that no data is replicated
3. Discuss your design

DIY Exercise (30 minutes)

1. Pick your own data set and/or application
2. Design it in terms of a relational database:
    1. Establish tables
    2. Establish relationships among tables, making sure that no data is replicated
3. Discuss your design

DIY Exercise (30 minutes)

1. Pick your own data set and/or application
2. Design it in terms of a relational database:
    1. Establish tables
    2. Establish relationships among tables, making sure that no data is replicated
3. Discuss your design

What have we covered

1. What are databases and why do they exist at all
2. How they are conceived
3. What are data relationships
4. An introduction to the SQL language
5. How to design proper database structures (with some exercising)

What have we covered

1. What are databases and why do they exist at all
2. How they are conceived
3. What are data relationships
4. An introduction to the SQL language
5. How to design proper database structures (with some exercising)

What have we covered

1. What are databases and why do they exist at all
2. How they are conceived
3. What are data relationships
4. An introduction to the SQL language
5. How to design proper database structures (with some exercising)

What have we covered

1. What are databases and why do they exist at all
2. How they are conceived
3. What are data relationships
4. An introduction to the SQL language
5. How to design proper database structures (with some exercising)

What have we covered

1. What are databases and why do they exist at all
2. How they are conceived
3. What are data relationships
4. An introduction to the SQL language
5. How to design proper database structures (with some exercising)

What have we covered

1. What are databases and why do they exist at all
2. How they are conceived
3. What are data relationships
4. An introduction to the SQL language
5. How to design proper database structures (with some exercising)

THANK YOU!