Session 6

Relational Databases
(Database design for analysts)

NLAB: *Data at Scale*

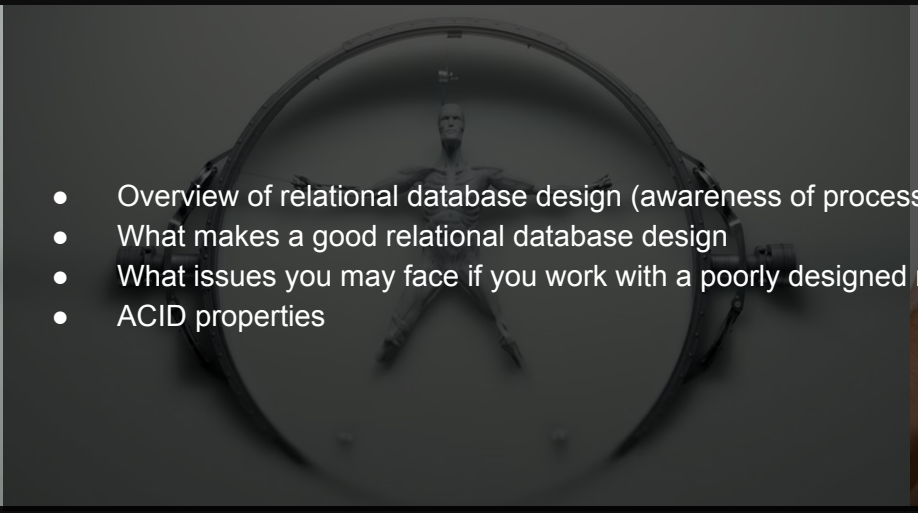
Dr Georgiana Nica-Avram

today.

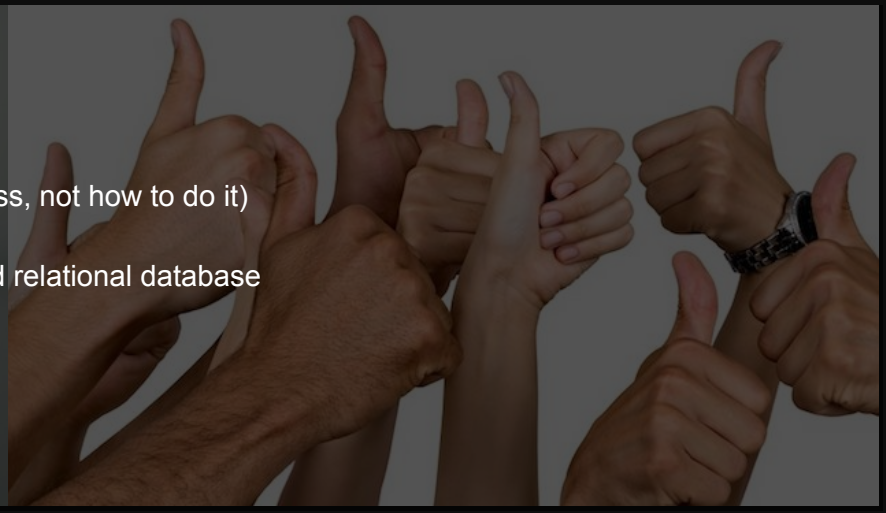
N/LAB: *Data at Scale*

Dr Georgiana Nica-Avram

today.



- Overview of relational database design (awareness of process, not how to do it)
- What makes a good relational database design
- What issues you may face if you work with a poorly designed relational database
- ACID properties



today.

N/LAB: *Data at Scale*

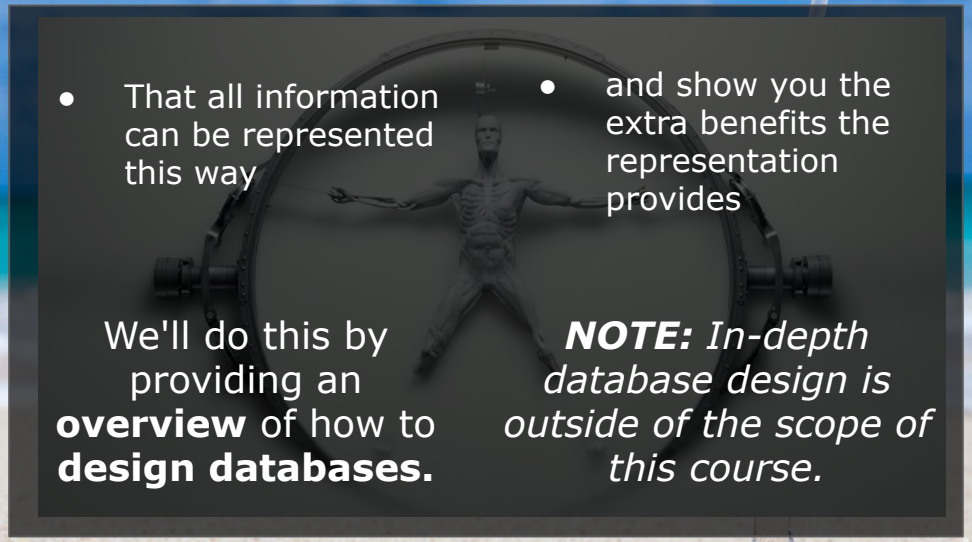
Dr Georgiana Nica-Avram

So… Hopefully I've now (or at least after next week) convinced you…

Facts provide good building blocks since

→ Once data is in this form it's "easy" to manipulate

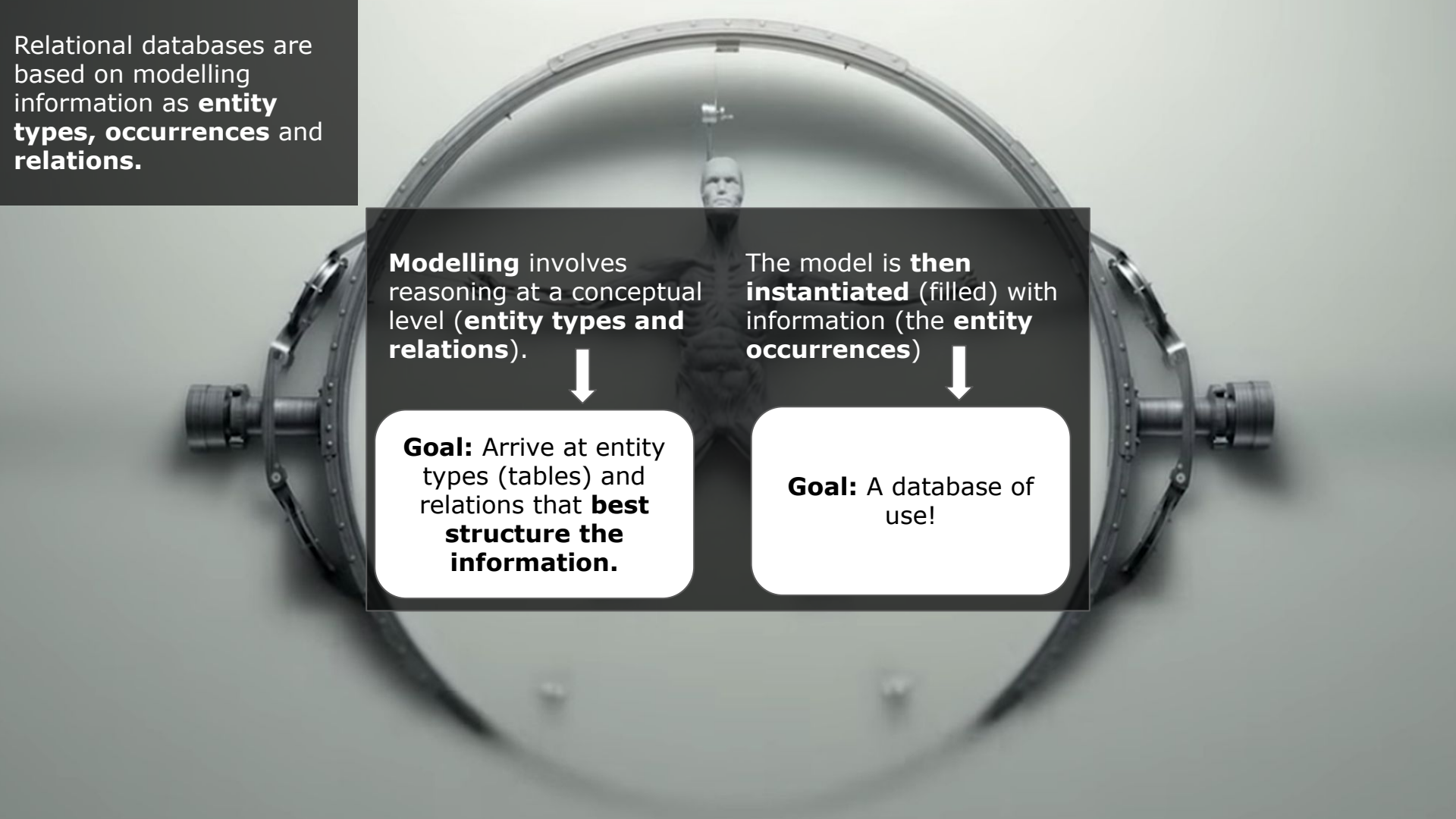
I now need to convince you…



- That all information can be represented this way
- and show you the extra benefits the representation provides

We'll do this by providing an **overview** of how to **design databases.**

NOTE: *In-depth database design is outside of the scope of this course.*



Relational databases are based on modelling information as **entity types, occurrences** and **relations.**

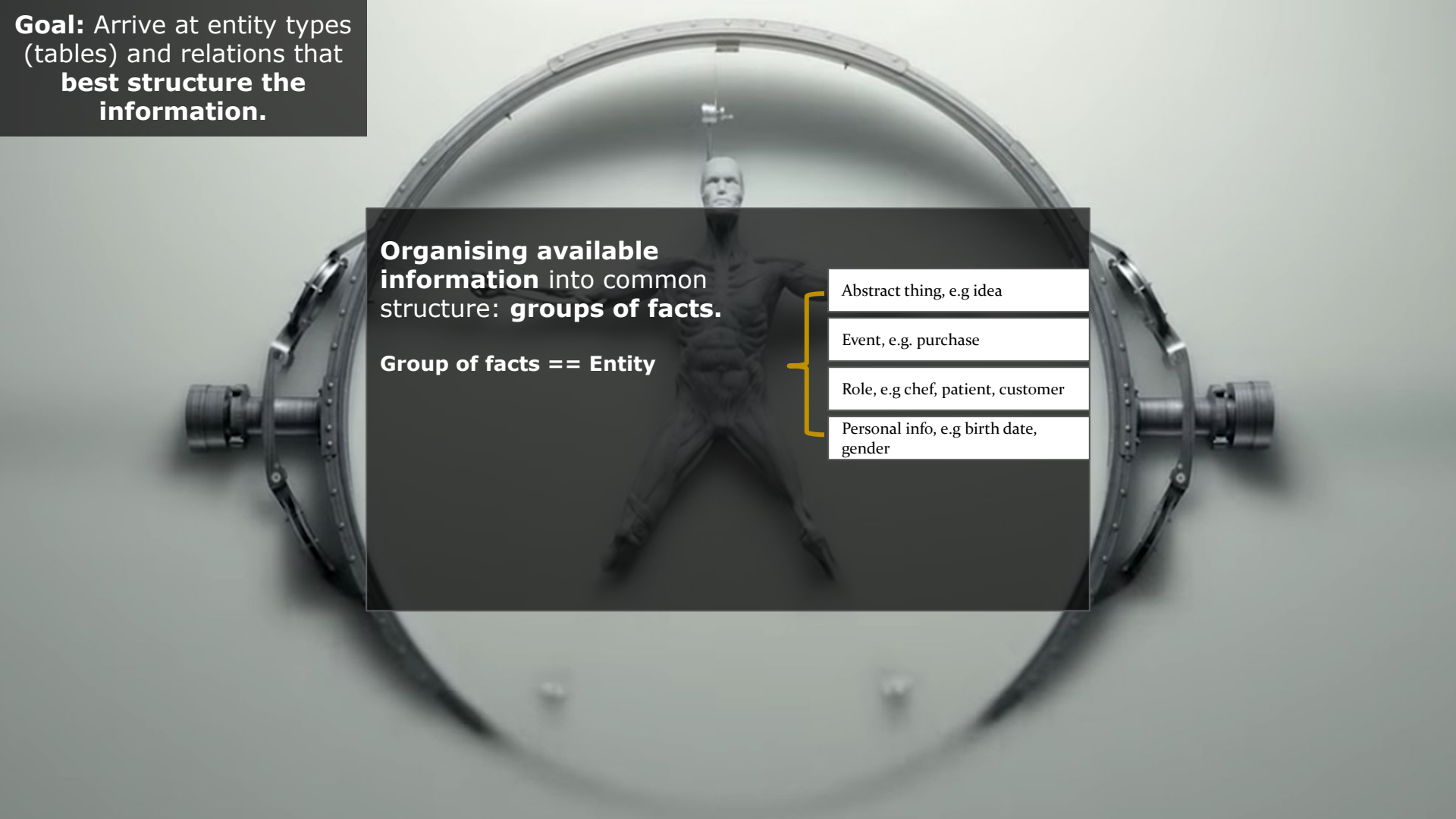
Modelling involves reasoning at a conceptual level (**entity types and relations**).

Goal: Arrive at entity types (tables) and relations that **best structure the information.**

The model is **then instantiated** (filled) with information (the **entity occurrences**)

Goal: A database of use!

Goal: Arrive at entity types (tables) and relations that **best structure the information.**



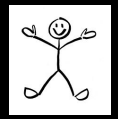
Organising available information into common structure: **groups of facts.**

Group of facts == Entity

- Abstract thing, e.g idea
- Event, e.g. purchase
- Role, e.g chef, patient, customer
- Personal info, e.g birth date, gender

An interlude...

We are NOT identifying and storing objects and properties!

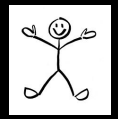


This is Bob.

Where should Bob be stored?

An interlude...

We are NOT identifying and storing objects and properties of objects!



This is Bob.

Where should Bob be stored?

Object Database:
In a "Person Store" as Bob (obviously!)!

An interlude...

We are NOT identifying and storing objects and properties of objects!

This is Bob.

Where should Bob be stored?

Object Database:
In a "Person Store" as Bob (obviously!)!

Relational Database:

But Bob is a Chef **and** a Patient.

The hospital doesn't care that Bob is a chef…

An interlude...

We are NOT identifying and storing objects and properties of objects!

This is Bob.

Where should Bob be stored?

Chef table

Patient table

Object Database:
In a "Person Store" as Bob (obviously!)!

Relational Database:

But Bob is a Chef **and** a Patient.

The hospital doesn't care that Bob is a chef…

Group information for a purpose:

Create "views" on Bob that can be linked back together if required.

Also helps prevent data duplication...

An interlude...

We are NOT identifying and storing objects and properties of objects!

This is Bob.

Where should Bob be stored?

Chef table

Patient table

Object Database:
In a "Person Store" as Bob (obviously!)!

Relational Database:

But Bob is a Chef **and** a Patient.

The hospital doesn't care that Bob is a chef...

Group information for a purpose:

Create "views" on Bob that can be linked back together if required.

Also helps prevent data duplication...

Dynamically create new "views" for different purposes via SQL.

More flexible modelling.

An interlude...

We are NOT identifying and storing objects and properties of objects!

This is Bob.

Where should Bob be stored?

Chef table

Patient table

Object Database:
In a "Person Store" as Bob (obviously!)!

Relational Database:

But Bob is a Chef **and** a Patient.

The hospital doesn't care that Bob is a chef...

Added benefit:

Less arguments on the philosophy of what an object is and how it should be stored (is a laptop an object? to Lenovo? to Intel?)

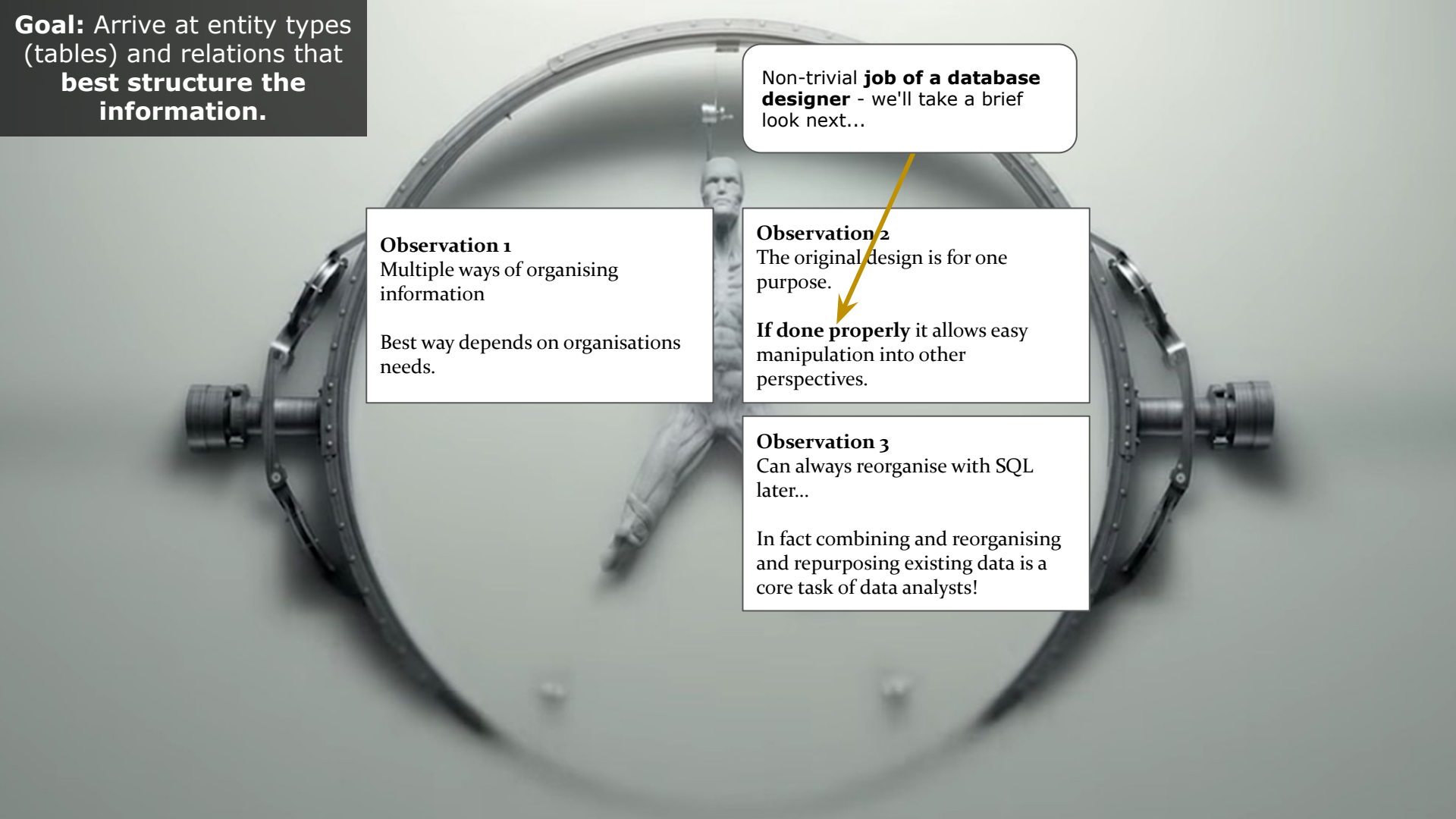
Group information for a purpose:

Create "views" on Bob that can be linked back together if required.

Also helps prevent data duplication...

Dynamically create new "views" for different purposes via SQL.

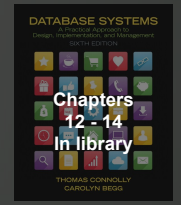
More flexible modelling.


Goal: Arrive at entity types (tables) and relations that **best structure the information.**
Non-trivial **job of a database designer** - we'll take a brief look next...
Observation 1
Multiple ways of organising information
Best way depends on organisations needs.
Observation 2
The original design is for one purpose.
If done properly it allows easy manipulation into other perspectives.
Observation 3
Can always reorganise with SQL later...
In fact combining and reorganising and repurposing existing data is a core task of data analysts!

Designing databases.

Or modelling information...

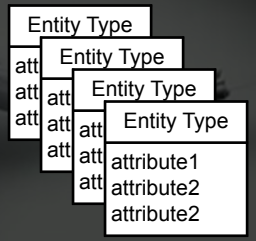
Draw a picture!



Repeat and refine, identifying attributes and cardinality of relationships

Step 1: Group information into entity types (identify entities)

Step 2: Identify relationships

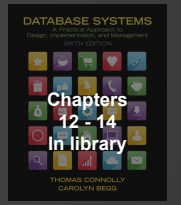


Step 3: Follow rules to make the entity types more atomic ("better lego pieces"). *Normalization.*

Step 4: Repeat until you (stakeholders) are happy or run out of time

Relationships

One-to-one



Relationships between individual entities are **associations** (links) between one or more **entities** (rows) from one entity type (table) to one or more entities (rows) of another entity type (table).

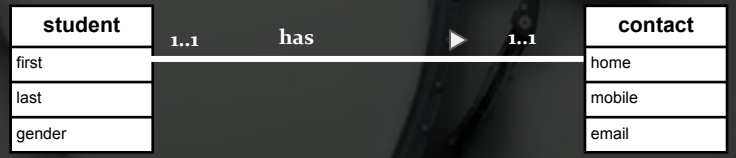
Relationships between **entity types** (tables) describe these relationships generally.

Entity type: student

first	last	gender
Mark	Jones	M
Victoria	Smith	F
Jane	Doe	F

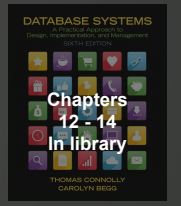
Entity type: contact

home	mobile	email
0115 84475	07586655810	m@x.com
0115 84786	07613399785	v@y.co.uk
0115 96375	07694411220	d@z.co.uk



Relationships

One-to-one



Relationships between individual entities are **associations** (links) between one or more **entities** (rows) from one entity type (table) to one or more entities (rows) of another entity type (table).

Relationships between **entity types** (tables) describe these relationships generally.

Entity type: student

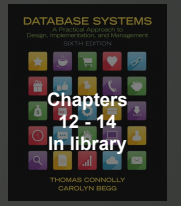
first	last	gender
Mark	Jones	M
Victoria	Smith	F
Jane	Doe	F

Entity type: contact

home	mobile	email
0115 84475	07586655810	m@x.com
0115 84786	07613399785	v@y.co.uk
0115 96375	07694411220	d@z.co.uk

Relationships

One-to-one



Relationships between individual entities are **associations** (links) between one or more **entities** (rows) from one entity type (table) to one or more entities (rows) of another entity type (table).

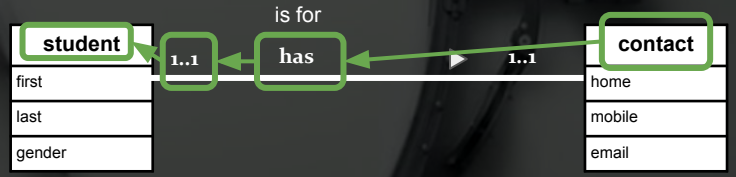
Relationships between **entity types** (tables) describe these relationships generally.

Entity type: student

first	last	gender
Mark	Jones	M
Victoria	Smith	F
Jane	Doe	F

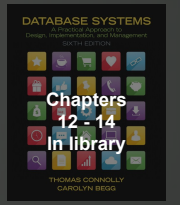
Entity type: contact

home	mobile	email
0115 84475	07586655810	m@x.com
0115 84786	07613399785	v@y.co.uk
0115 96375	07694411220	d@z.co.uk



Relationships

One-to-many



Relationships between individual entities are **associations** (links) between one or more **entities** (rows) from one entity type (table) to one or more entities (rows) of another entity type (table).

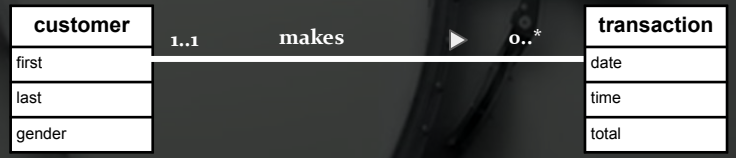
Relationships between **entity types** (tables) describe these relationships generally.

Entity type: customer

first	last	gender
Mark	Jones	M
Victoria	Smith	F
Jane	Doe	F

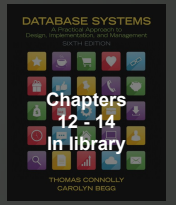
Entity type: transaction

date	time	total
2017-08-09	08:05:32	£12.00
2017-08-10	13:00:00	£6.50
2017-08-10	11:00:01	£3.00



Relationships

Many-to-many



Relationships between individual entities are **associations** (links) between one or more **entities** (rows) from one entity type (table) to one or more entities (rows) of another entity type (table).

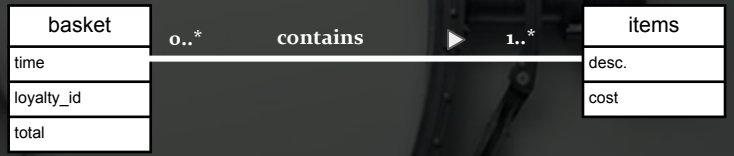
Relationships between **entity types** (tables) describe these relationships generally.

Entity type: basket

time	loyalty_id	total
11am	7896	£3.70
1pm	9934	£3.25
2pm	3900	£0.07

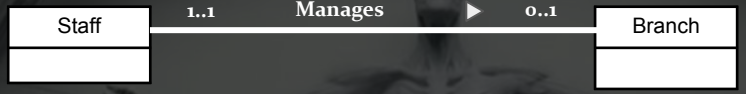
Entity type: items

desc.	cost
Doritos	£3.00
Chocolate bar	£0.70
Sandwich	£3.25

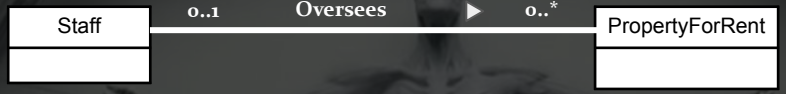


NOTE: An item can appear in multiple baskets as in the item table instances denote the generic item type. If item rows defined a specific physical item then the relationship would be one-to-many.

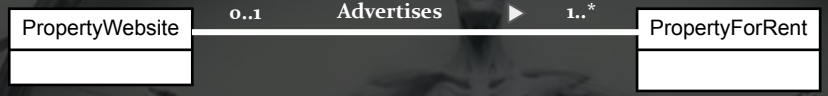
Some textbook examples…



Some textbook examples...



Some textbook examples...



Call Detail Record (CDR) Data (More event data...)

Calls

start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

duration
charge
serviceflow
roaming
result code
incoming
...

SMS

start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

charge
serviceflow
roaming
result code
incoming
sms length
...

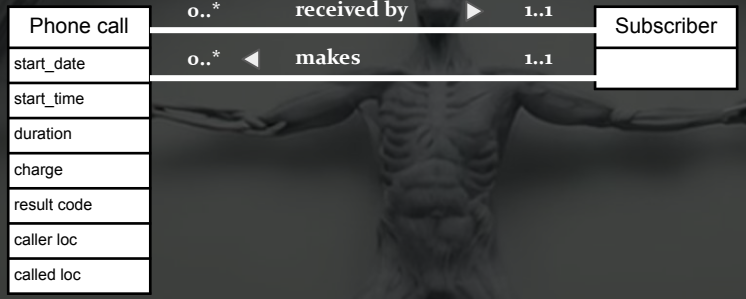
Data

start time
tower id
caller id
tariff type
prepaid balance
charge

product id
duration
data up
data down
...

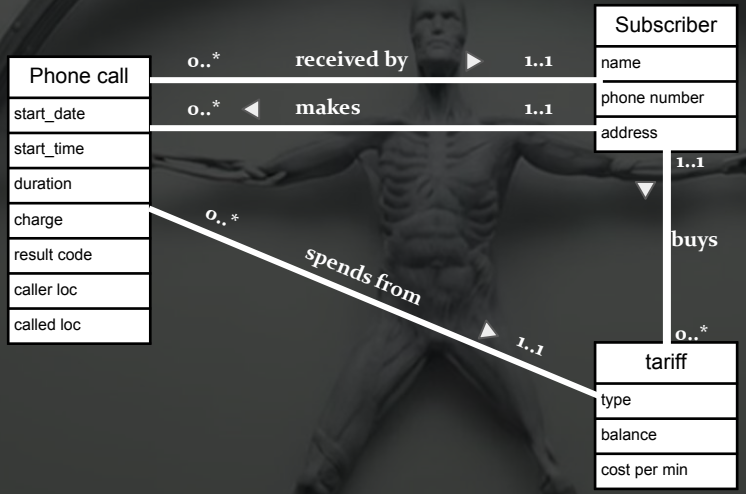
start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

duration
charge
roaming
result code
...



start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

duration
charge
roaming
result code
...

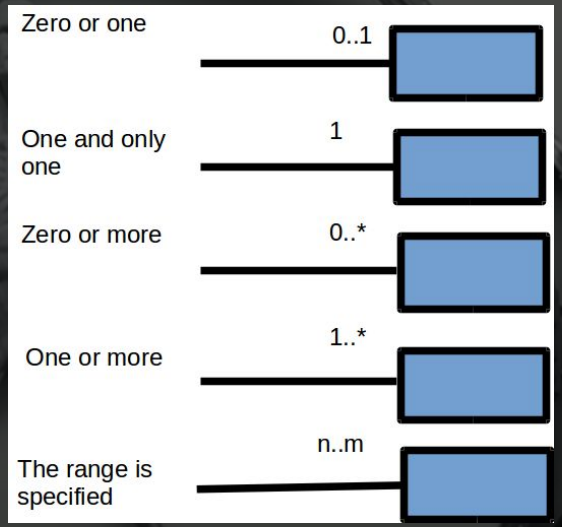


Nota bene

Among other functions, Unified Modeling Language (UML) is used to visualise, specify and document relationships between entity types (tables).

Relationships are expressed in terms of cardinality:

- one to one
- one to many
- many to many



Keys

In order **to utilise relationships** to manipulate data we need to be able to **uniquely identify entity occurrences**.

Keys:
Do not need to be IDs
(but very often are..)

Candidate key: The minimal set of attributes that uniquely identifies each occurrence of an entity type.

Primary key: The candidate key selected to uniquely identify each entity occurrence.

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Keys:
Do not need to be IDs
(but very often are..)

Candidate key: The minimal set of attributes that uniquely identifies each occurrence of an entity type.

Primary key: The candidate key selected to uniquely identify each entity occurrence.

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Keys should have:

- minimal attributes
- temporal invariance
- future certainty of uniqueness

Keys:
Do not need to be IDs
(but very often are..)

Candidate key: The minimal set of attributes that uniquely identifies each occurrence of an entity type.

Primary key: The candidate key selected to uniquely identify each entity occurrence.

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Keys should have:

- minimal attributes
- temporal invariance
- future certainty of uniqueness

Minimize data redundancy (more soon)

Keys:
Do not need to be IDs
(but very often are..)

Candidate key: The minimal set of attributes that uniquely identifies each occurrence of an entity type.

Primary key: The candidate key selected to uniquely identify each entity occurrence.

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Keys should have:

- minimal attributes
- temporal invariance
- future certainty of uniqueness

Required to prevent errors

Keys:
Do not need to be IDs
(but very often are..)

Candidate key: The minimal set of attributes that uniquely identifies each occurrence of an entity type.

Primary key: The candidate key selected to uniquely identify each entity occurrence.

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Entity type: basket

staff_id	date	time	shop	till	total
6	2017-01-01	11:00:00	123	4	£3.00
78	2017-01-02	12:00:00	849	3	£2.15
9	2017-01-02	13:00:01	837	2	£15.00

Keys should have:

- minimal attributes
- temporal invariance
- future certainty of uniqueness

Often this means artificially creating an ID for this purpose.

How are they used?

Implicitly define relationships!!

"keys should have minimal attributes"

→ minimise data redundancy

Foreign key: attribute (or set of) that uniquely identifies a row in another table

How are they used?

Implicitly define relationships!!

Candidate (primary) key

Subscriber

subscriber_id

What you need to know:

1) Entity types (tables) are linked together by relationships.

2) Relationships are implicitly defined through **candidate** and **foreign keys.**

3) These can exist **by design** or **opportunistically.**

4) How to read entity relationship diagrams

Candidate (primary) key

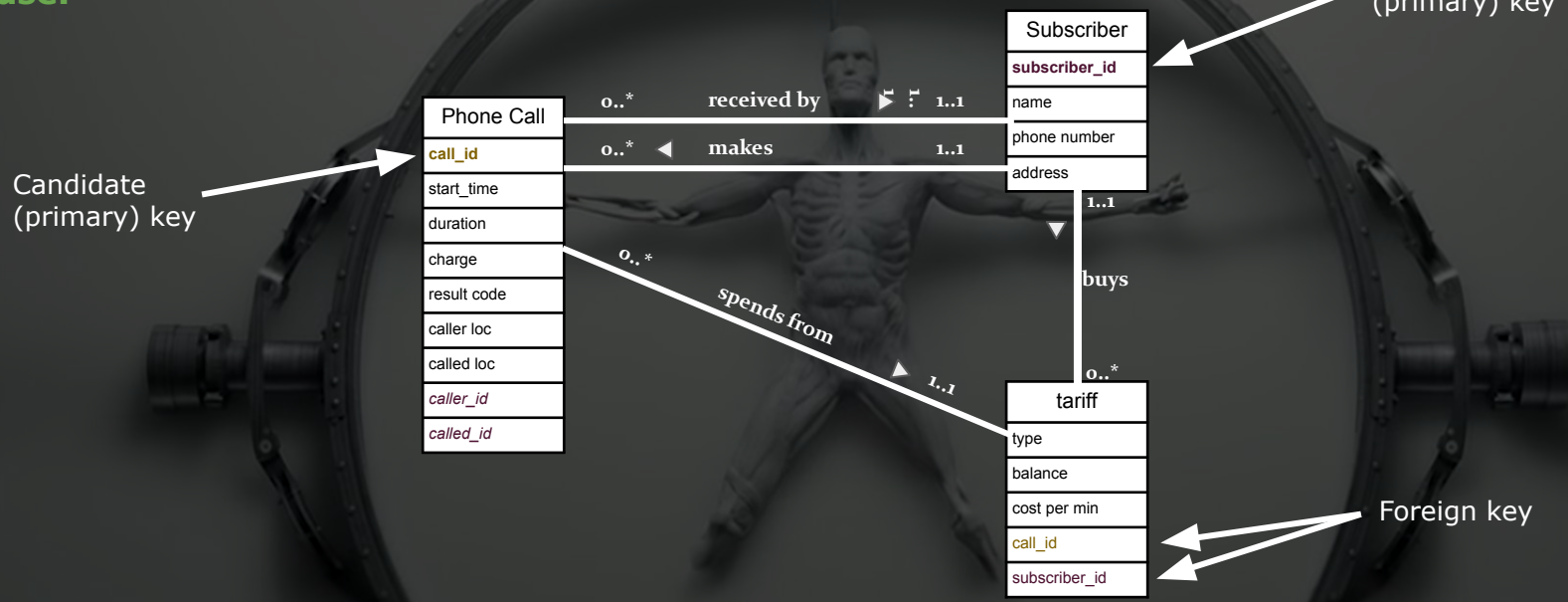
cost per min

call_id

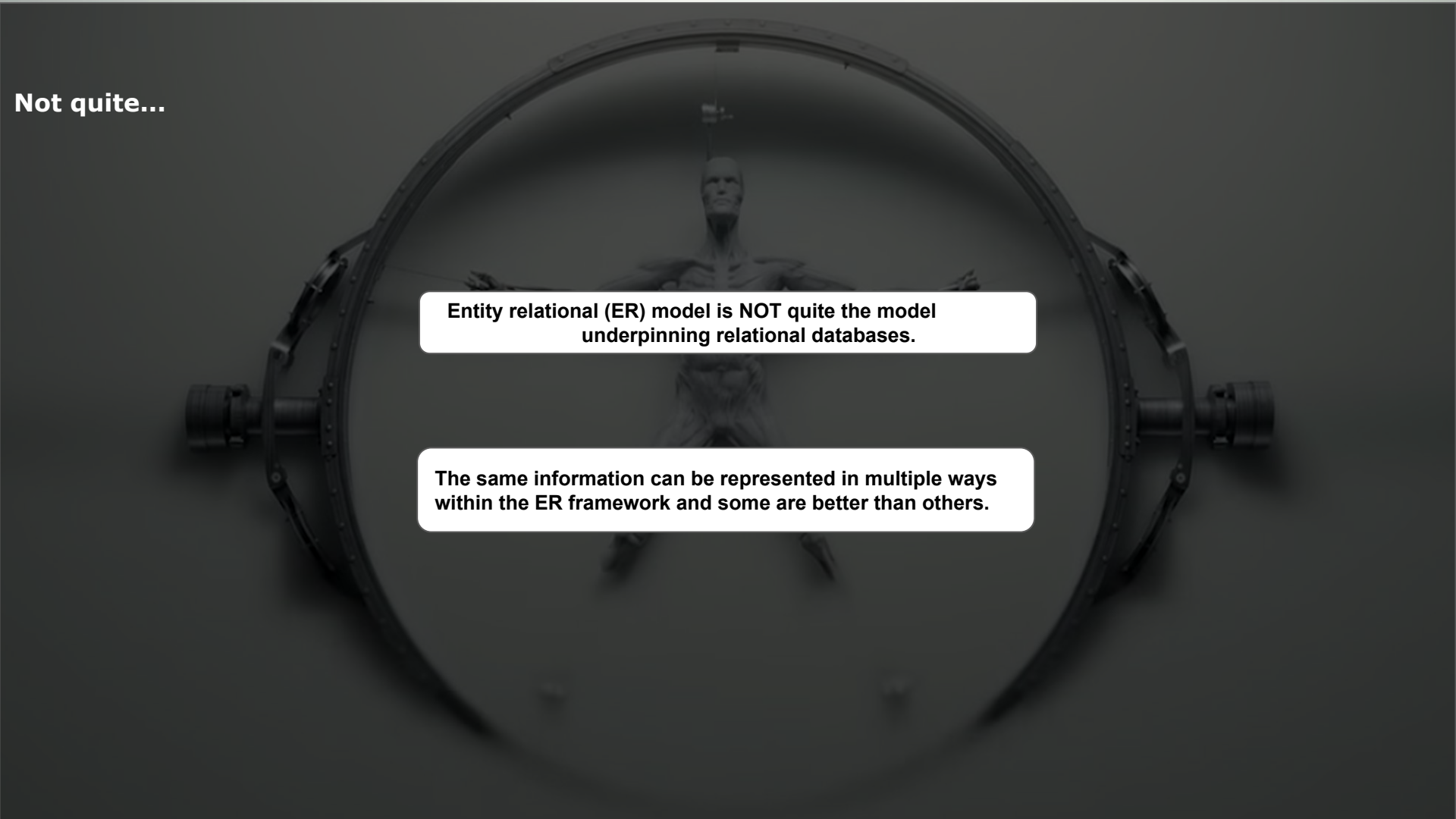
subscriber_id

Foreign key

Hurray! We now have a model of the world we can use!

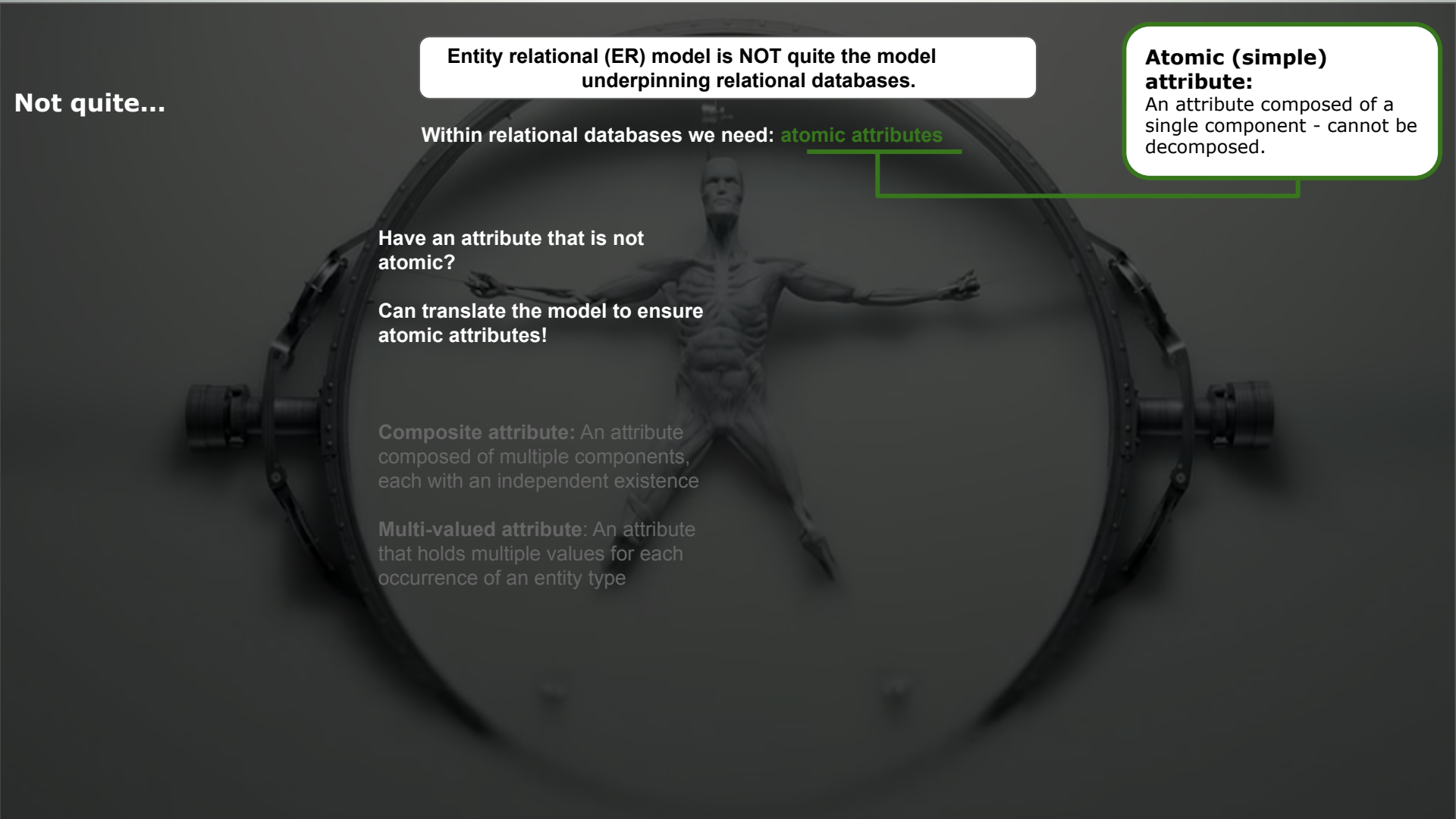


Not quite...



Entity relational (ER) model is NOT quite the model underpinning relational databases.

The same information can be represented in multiple ways within the ER framework and some are better than others.



Not quite...

Entity relational (ER) model is NOT quite the model underpinning relational databases.

Within relational databases we need: atomic attributes

Atomic (simple) attribute:
An attribute composed of a single component - cannot be decomposed.

Have an attribute that is not atomic?

Can translate the model to ensure atomic attributes!

Composite attribute: An attribute composed of multiple components, each with an independent existence

Multi-valued attribute: An attribute that holds multiple values for each occurrence of an entity type

Not quite...

Entity relational (ER) model is NOT quite the model Underpinning relational databases.

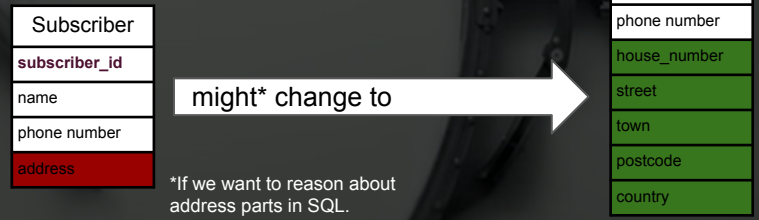
Within relational databases we need: atomic attributes

Have an attribute that is not atomic?

Can translate the model to ensure atomic attributes!

Composite attribute: An attribute composed of multiple components, each with an independent exists

Multi-valued attribute: An attribute that holds multiple values for each occurrence of an entity type



*If we want to reason about address parts in SQL.

Not quite...

Entity relational (ER) model underpinning is NOT quite the model relational databases.

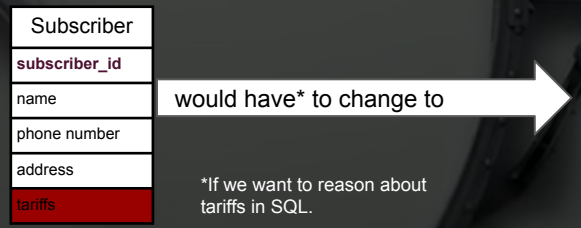
Within relational databases we need: atomic attributes

Have an attribute that is not atomic?

Can translate the model to ensure atomic attributes!

Composite attribute: An attribute composed of multiple components, each with an independent exists

Multi-valued attribute: An attribute that holds multiple values for each occurrence of an entity type



*If we want to reason about tariffs in SQL.

Not quite...

Entity relational (ER) model underpinning is NOT quite the model relational databases.

Within relational databases we need: atomic attributes

Have an attribute that is not atom

Ca at

Mult
that holds multiple values for each occurrence of an entity type

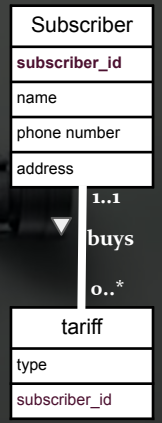
What you need to know:

1) Entity types (tables) only have atomic attributes

2) **This does not limit what can be modelled.**

to

*If we want to reason about tariffs in SQL.



Not quite...

The same information can be represented in multiple ways within the ER framework and some are better than others.

Good database design has minimal data redundancy

Redundant data

- Is data that _already_ exists elsewhere in the database
- Redundant data leads to various subtle, but important problems:
  - INSERT anomalies
  - UPDATE anomalies
  - DELETE anomalies

Good database design has minimal data redundancy

Entity Type: till_payments

Real world example of one transaction (basket) being paid for by **card + cash**

basket_id	till_id	store	total	amount paid	credit card num
B1	T12	Nottingham	£30.00	£20.00	C1
B1	T12	Nottingham	£30.00	£10.00	C2
B2	T11	Nottingham	£15.00	£5.00	C1
B2	T11	Nottingham	£15.00	£10.00	C2
B3	T33	York	£2.00	£2.00	C4
B4	T41	Bath	£60.00	£55.00	C5
B4	T41	Bath	£60.00	£5.00	C6
B5	T51	Bath	£44.00	£44.00	C7

- **INSERT anomalies**

  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the till_id and store even though the store can only have one value once the till_id is known.

- **UPDATE anomalies**

  To change store for T12, we have to change two rows

- **DELETE anomalies**

  If we remove B3, we remove any trace of the York store as well

Good database design has minimal data redundancy

Entity Type: till_payments

basket_id	till_id	store	total	amount paid	credit card num
B1	T12	Nottingham	£30.00	£20.00	C1
B1	T12	Nottingham	£30.00	£10.00	C2
B2	T11	Nottingham	£15.00	£5.00	C1
B2	T11	Nottingham	£15.00	£10.00	C2
B3	T33	York	£2.00	£2.00	C4
B4	T41	Bath	£60.00	£55.00	C5
B4	T41	Bath	£60.00	£5.00	C6
B5	T51	Bath	£44.00	£44.00	C7

- **INSERT anomalies**

  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the till_id and store even though they always appear together.

- **UPDATE anomalies**

  To change store for T12, we have to change two rows

- **DELETE anomalies**

  If we remove B3, we remove any trace of the York store as well

Good database design has minimal data redundancy

Entity Type: till_payments

basket_id	till_id	store	total	amount paid	credit card num
B1	T12	Nottingham	£30.00	£20.00	C1
B1	T12	Nottingham	£30.00	£10.00	C2
B2	T11	Nottingham	£15.00	£5.00	C1
B2	T11	Nottingham	£15.00	£10.00	C2
B3	T33	York	£2.00	£2.00	C4
B4	T41	Bath	£60.00	£55.00	C5
B4	T41	Bath	£60.00	£5.00	C6
B5	T51	Bath	£44.00	£44.00	C7

- **INSERT anomalies**

  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the till_id and store even though they always appear together.

- **UPDATE anomalies**

  To change store for T12, we have to change two rows

- **DELETE anomalies**

  If we remove B3, we remove any trace of the York store as well

Good database design has minimal data redundancy

Entity Type: till_payments

basket_id	till_id	store	total	amount paid	credit card num
B1	T12	Nottingham	£30.00	£20.00	C1
B1	T12	Nottingham	£30.00	£10.00	C2
B2	T11	Nottingham	£15.00	£5.00	C1
B2	T11	Nottingham	£15.00	£10.00	C2
B3	T33	York	£2.00	£2.00	C4
B4	T41	Bath	£60.00	£55.00	C5
B4	T41	Bath	£60.00	£5.00	C6
B5	T51	Bath	£44.00	£44.00	C7

Bad news: Identifying entities and relationships is not enough :(

Good news: Maths underpinning = rules to do this. Equivalent representation.

Bad news: It's complicated, based on analysing functional dependencies between attributes.

(i.e. if you know the till_id you know the store)

Good news: This is the role of database designers not business analytics.

Good news: We get to avoid these issues by design.

For fun let's convince ourselves this is true...

Entity Type: till_payments

basket_id	till_id	store	total	amount paid	credit card num
B1	T12	Nottingham	£30.00	£20.00	C1
B1	T12	Nottingham	£30.00	£10.00	C2
B2	T11	Nottingham	£15.00	£5.00	C1
B2	T11	Nottingham	£15.00	£10.00	C2
B3	T33	York	£2.00	£2.00	C4
B4	T41	Bath	£60.00	£55.00	C5
B4	T41	Bath	£60.00	£5.00	C6
B5	T51	Bath	£44.00	£44.00	C7

Entity Type: baskets

basket_id	till_id	total
B1	T12	£30.00
B2	T11	£15.00
B3	T33	£2.00
B4	T41	£60.00
B5	T51	£44.00

Entity Type: tills

till_id	store
T12	Nottingham
T11	Nottingham
T33	York
T41	Bath
T51	Bath

Entity Type: credit card transactions

basket_id	amount paid	credit card num
B1	£20.00	C1
B1	£10.00	C2
B2	£5.00	C1
B2	£10.00	C2
B3	£2.00	C4
B4	£55.00	C5
B4	£5.00	C6
B5	£44.00	C7

- **INSERT anomalies**

  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the till_id and store even though the store can only have one value once the till_id is known.

- **UPDATE anomalies**

  To change store for T12, we have to change two rows

- **DELETE anomalies**

  If we remove B3, we remove any trace of the York store as well

For fun let's convince ourselves this is true...

Entity Type: till_payments

basket_id	till_id	store	total	amount paid	credit card num
B1	T12	Nottingham	£30.00	£20.00	C1
B1	T12	Nottingham	£30.00	£10.00	C2
B2	T11	Nottingham	£15.00	£5.00	C1
B2	T11	Nottingham	£15.00	£10.00	C2
B3	T33	York	£2.00	£2.00	C4
B4	T41	Bath	£60.00	£55.00	C5
B4	T41	Bath	£60.00	£5.00	C6
B5	T51	Bath	£44.00	£44.00	C7

Entity Type: baskets

basket_id	till_id	total
B1	T12	£30.00
B2	T11	£15.00
B3	T33	£2.00
B4	T41	£60.00
B5	T51	£44.00

Entity Type: tills

till_id	store
T12	Nottingham
T11	Nottingham
T33	York
T41	Bath
T51	Bath

Entity Type: credit card transactions

basket_id	amount paid	credit card num
B1	£20.00	C1
B1	£10.00	C2
B2	£5.00	C1
B2	£10.00	C2
B3	£2.00	C4
B4	£55.00	C5
B4	£5.00	C6
B5	£44.00	C7

- **INSERT anomalies**

  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the till_id and store even though they always appear together.

- **UPDATE anomalies**

  To change store for T12, we have to change two rows

- **DELETE anomalies**

  If we remove B3, we remove any trace of the York store as well

For fun let's convince ourselves this is true...

Entity Type: till_payments

basket_id	till_id	store	total	amount paid	credit card num
B1	T12	Nottingham	£30.00	£20.00	C1
B1	T12	Nottingham	£30.00	£10.00	C2
B2	T11	Nottingham	£15.00	£5.00	C1
B2	T11	Nottingham	£15.00	£10.00	C2
B3	T33	York	£2.00	£2.00	C4
B4	T41	Bath	£60.00	£55.00	C5
B4	T41	Bath	£60.00	£5.00	C6
B5	T51	Bath	£44.00	£44.00	C7

Entity Type: baskets

basket_id	till_id	total
B1	T12	£30.00
B2	T11	£15.00
B3	T33	£2.00
B4	T41	£60.00
B5	T51	£44.00

Entity Type: tills

till_id	store
T12	Nottingham
T11	Nottingham
T33	York
T41	Bath
T51	Bath

Entity Type: credit card transactions

basket_id	amount paid	credit card num
B1	£20.00	C1
B1	£10.00	C2
B2	£5.00	C1
B2	£10.00	C2
B3	£2.00	C4
B4	£55.00	C5
B4	£5.00	C6
B5	£44.00	C7

- **INSERT anomalies**

  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the

  till_id and store even though they always appear together.

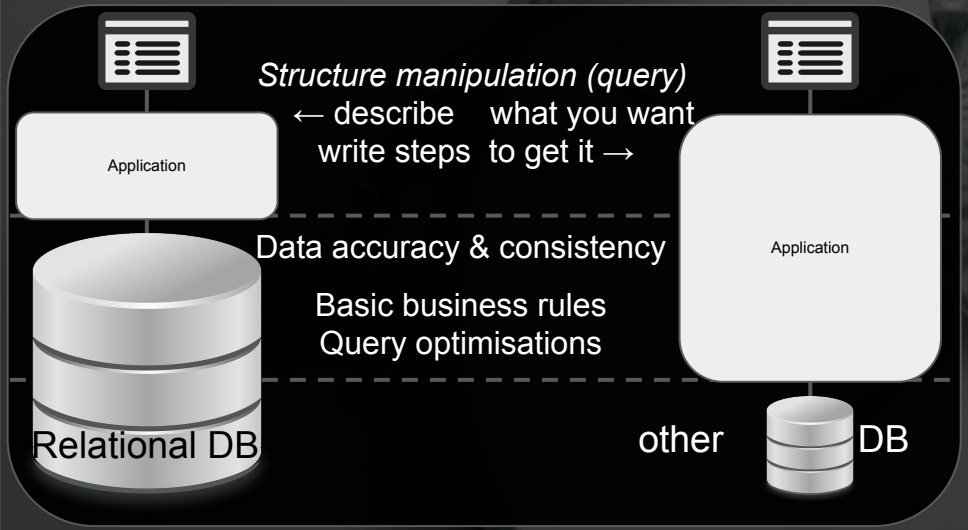
- **UPDATE anomalies**

  To change store for T12, we have to change two rows

- **DELETE anomalies**

  If we remove B3, we remove any trace of the York store as well

What if we don't have this by design?



In this form we must write code to enable statistics (and business logic) with NULLs….

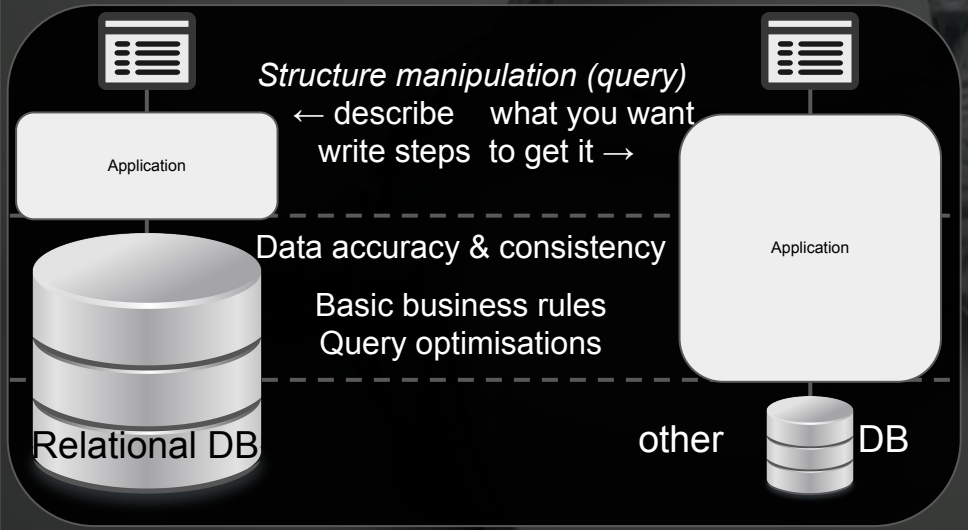
- **INSERT anomalies**
  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the till_id and store even though they always appear together.

- **UPDATE anomalies**
  To change store for T12, we have to change two rows

- **DELETE anomalies**
  If we remove B3, we remove any trace of the York store as well

What if we don't have this by design?



In this form, we must write code to enforce / fix consistency issues...

- **INSERT anomalies**
  Can't add a basket with no credit card number (without NULLs)

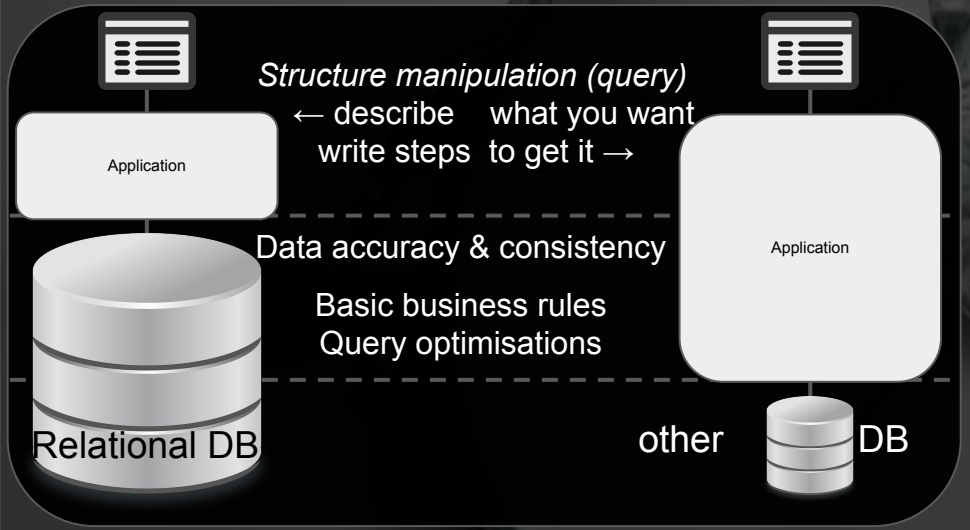
  For each basket we must correctly enter both the

  till_id and store even though they always appear together.

- **UPDATE anomalies**
  To change store for T12, we have to change two rows

- **DELETE anomalies**
  If we remove B3, we remove any trace of the York store as well

What if we don't have this by design?



In this form we potentially lose data… must account for this somehow...

- **INSERT anomalies**
  Can't add a basket with no credit card number (without NULLs)

  For each basket we must correctly enter both the till_id and store even though they always appear together.

- **UPDATE anomalies**
  To change store for T12, we have to change two rows

- **DELETE anomalies**
  If we remove B3, we remove any trace of the York store as well

Summary

What you need to know:

1) Entity types (tables) are linked together by relationships.
2) Relationships are implicitly defined through **candidate** and **foreign keys.**
3) These can exist **by design** or **inherently.**
4) How to read entity relationship diagrams.

What you need to know:

1) Entity types (tables) are linked by keys.
2) **This does not limit what can be modelled.**

What you need to know:

1) The relational model can model all information*
2) There is a fixed, mathematical approach to arrive at tables that prevents unnecessary data replication.
3) This enables data consistency **by design** without application code.
4) When you want to find information, you'll generally only find it in one table...

Want to know more about database design: Chapters 14 and 15 of Database Systems by Connolly & Begg

That's all for database design. However...

Relational databases still have an **extra set of properties** that make them **pretty awesome in the real world.**

That's all for database design. However...

Relational databases still have an **extra set of properties** that make them **pretty awesome in the real world.**

Part of this is due to the **simple structure** of tables and small number of SQL operations.

Part of this is due to **significant engineering and research effort** that has gone into relational databases over the years.

That's all for database design. However...

Specifically, relational databases have well defined methods to deal with **concurrency**.

That's all for database design. However...

Specifically, relational databases have well defined methods to deal with **concurrency**.

This enables **multiple people/machines** to update, read, add and delete data while **keeping the data consistent and correct**.

Prevents:

- Lost (overridden) updates
- Uncommitted update
- Inconsistent analysis
  (i.e. data point changes partway through an algorithm, different values for the same thing used)

That's all for database design. However...

Specifically, relational databases have well defined methods to deal with **concurrency**.

Magic of databases is that transactions are executed concurrently unless absolutely required to run sequentially - **high efficiency**.

That's all for database design. However...

More information, see Chapter 14: Database System Concepts, Silberschatz, Korth & Sudarshan

Relational databases define and implement **transactions**.

Transactions (statements) are defined as a "logical unit of work".

Transactions have a guaranteed set of properties: ACID...

That's all for database design. However...

More information, see Chapter 14: Database System Concepts, Silberschatz, Korth & Sudarshan

ACID: Atomicity, Consistency, Isolation, Durability.

That's all for database design. However...

More information, see Chapter 14: Database System Concepts, Silberschatz, Korth & Sudarshan

Atomicity: Everything in a transaction either happens or it doesn't.
(all pieces of information are committed or none)

That's all for database design. However...

More information, see Chapter 14: Database System Concepts, Silberschatz, Korth & Sudarshan

Atomicity: Everything in a transaction either happens or it doesn't.
(all pieces of information are committed or none)

Consistency: The database at the end of the transaction must be in a consistent state, or the transaction is rolled back (all [unique] constraints, keys, rules etc. holding).

That's all for database design. However...

More information, see Chapter 14: Database System Concepts, Silberschatz, Korth & Sudarshan

Atomicity: Everything in a transaction either happens or it doesn't.
(all pieces of information are committed or none)

Consistency: The database at the end of the transaction must be in a consistent state, or the transaction is rolled back (all [unique] constraints, keys, rules etc. holding).

Isolation: Execution of a transaction occurs in isolation, or fails.
(i.e. no interaction with any other concurrently running transactions)

That's all for database design. However...

More information, see Chapter 14: Database System Concepts, Silberschatz, Korth & Sudarshan

Atomicity: Everything in a transaction either happens or it doesn't.
(all pieces of information are committed or none)

Consistency: The database at the end of the transaction must be in a consistent state, or the transaction is rolled back (all [unique] constraints, keys, rules etc. holding).

Isolation: Execution of a transaction occurs in isolation, or fails.
(i.e. no interaction with any other concurrently running transactions)

Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

That's all for database design. However...

More information, see Chapter 14: Database System Concepts, Silberschatz, Korth & Sudarshan

Atomicity: Everything in a transaction either happens or it doesn't.
(all pieces of information are committed or none)

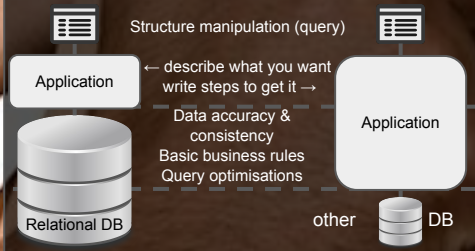
Consistency: The database at the end of the transaction must be in a consistent state, or the transaction is rolled back (all [unique] constraints, keys, rules etc. holding).

Isolation: Execution of a transaction occurs in isolation, or fails.
(i.e. no interaction with any other concurrently running transactions)

Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

That's all for database design. However...

So that's nice. What's the catch?



Turns out it is **hard to scale** these mechanisms up to distributed environments. **noSQL** (i.e. most object databases) **explicitly "traded this off"**.

Time for our morning exercises...

0..* 1..1 1..*

action

Vehicles	
Attrbute	**Data type**
Vehicle ID	

Parts	
Attrbute	**Data type**
Parts ID	
Vehicle ID	

Pieces	
Attrbute	**Data type**
Piece ID	
Type	
Parts ID	
Mechanics ID	

Mechanics	
Attrbute	**Data type**
Job ID	
Factory address	