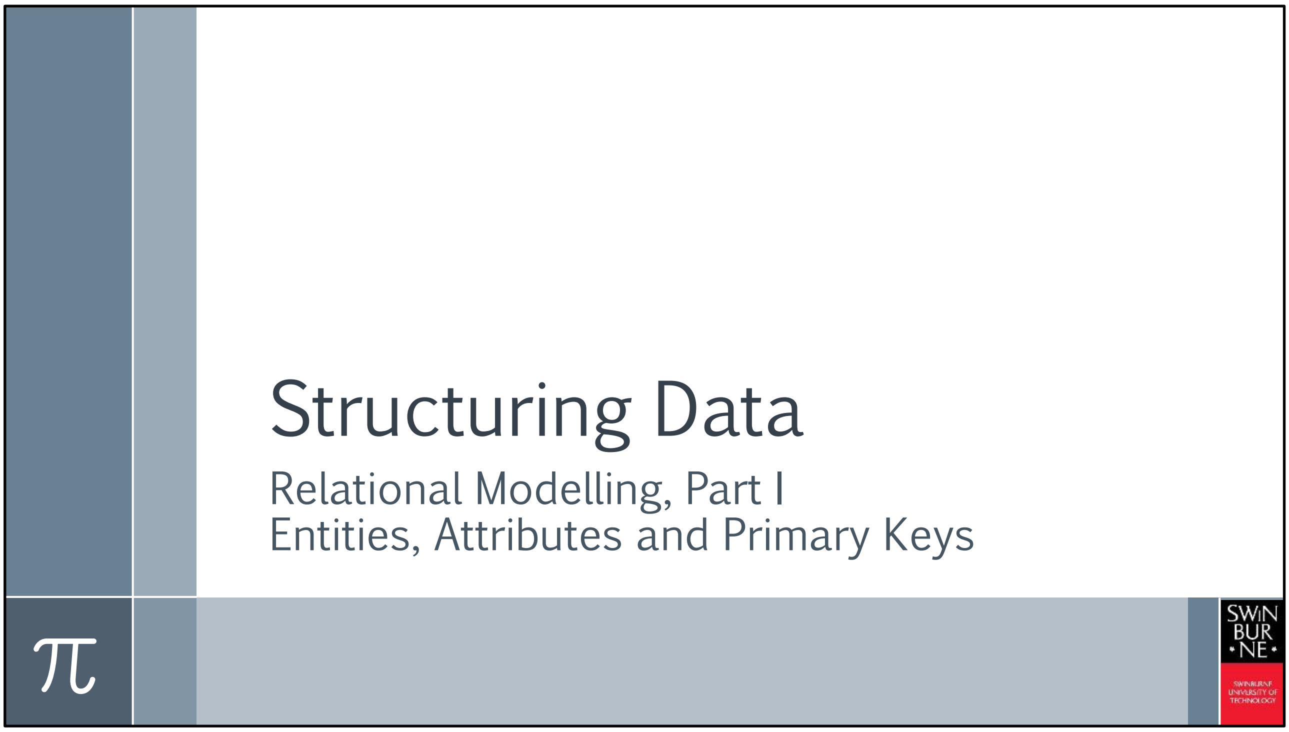


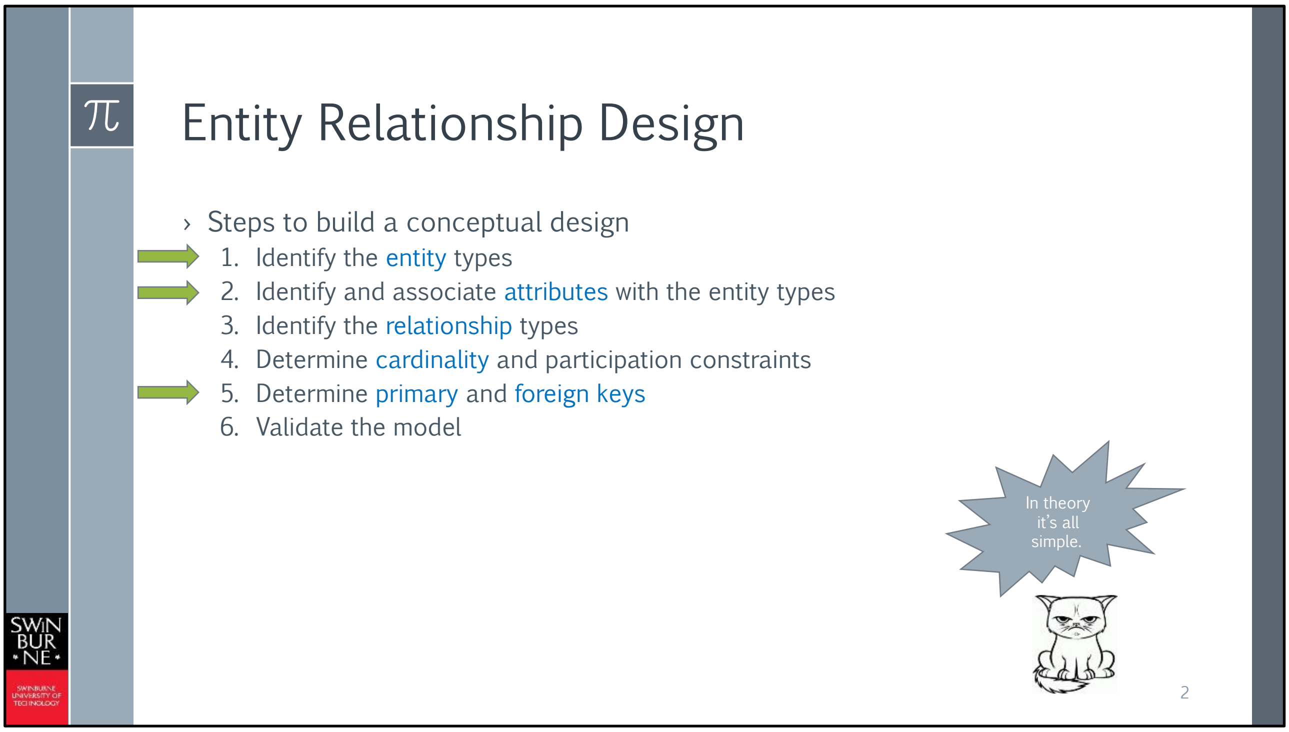
Structuring Data

Relational Modelling, Part I
Entities, Attributes and Primary Keys

π

SWIN BUR NE
SWINBURNE UNIVERSITY OF TECHNOLOGY

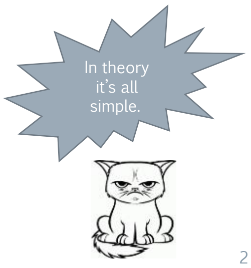
This is the first part of three in our discussion of relational modelling.



Entity Relationship Design

› Steps to build a conceptual design

1. Identify the entity types
2. Identify and associate attributes with the entity types
3. Identify the relationship types
4. Determine cardinality and participation constraints
5. Determine primary and foreign keys
6. Validate the model



2

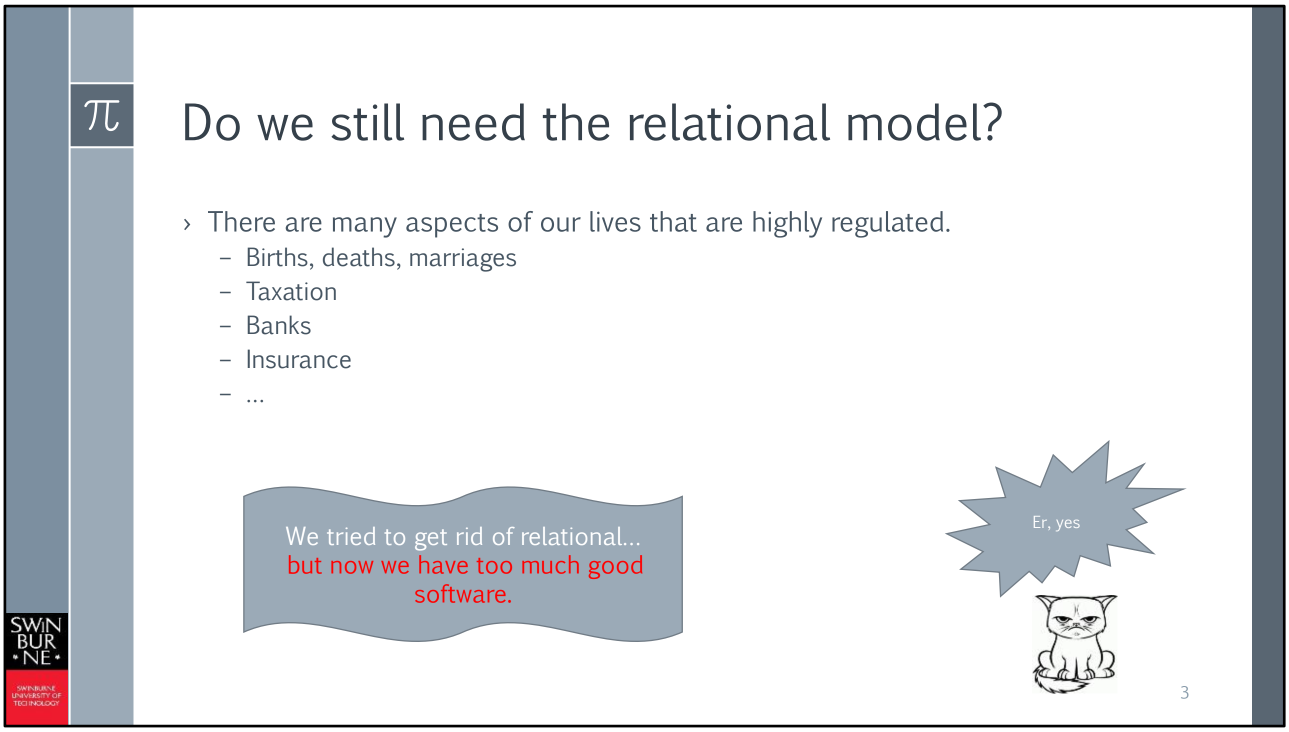
First we identify the entities. Many people think of a collection of entities as a table where a single entity is a table row. That's not strictly correct, but rather helpful in practice. That's why I'm illustrating this using tables. Attributes are often thought of as table columns. The column defines the name and type of the data. Each entry has a value in each column. Naming attributes so that their purpose is easily understood is good modelling practice. Defining attribute types with thought helps ensure the wrong data cannot be entered by accident.
One entity does not make a database. Complex information can be stored by linking the tables, i.e. creating relationships between tables.
It is important to know what the cardinalities of these relationships are: How many rows of table 1 link to one row of table 2, for example.

Primary keys are also attributes, but they are really important. They are the identifiers of the entities, and every other attribute in the entity should be defined by it. A foreign key is an enforced relationship between entities.

The step of validating the model is best achieved by working through the normal forms of relational design. These are explained in the Normalisation module.

It's possible to follow this step by step model, but you are likely to revisit step 1 after you have been through step 5, it’s mostly an iterative process.

In this module, we discuss steps 1 and 2 – we leave the rest for later.



Do we still need the relational model?

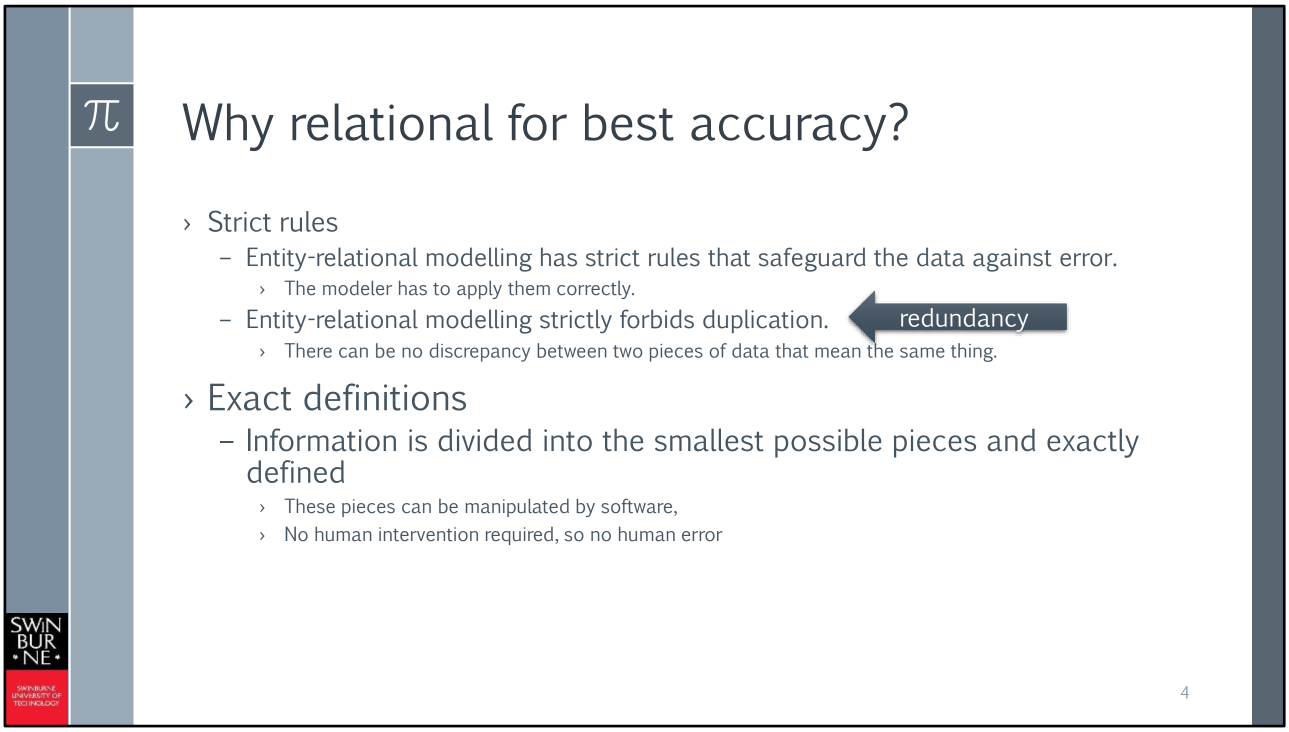
› There are many aspects of our lives that are highly regulated.
  - Births, deaths, marriages
  - Taxation
  - Banks
  - Insurance
  - …

We tried to get rid of relational…
but now we have too much good software.

Er, yes

3

There are lots of aspects of our lives where accuracy matters. You don't want your parents registered wrongly on your birth certificate. You don't want to pay a hefty tax bill because the taxation office has their records in a muddle. You don't want your bank to lose your balance, or your insurance to lose your details.
The relational model went out of fashion for a while. Maybe 1995 – 2005, a lot of development was spent on XML and object-oriented databases, which were a better fit with web languages and programming paradigms. None of this stuck. And by now, relational database management systems, and there are many, have become very reliable with lots of convenient functionality.



Why relational for best accuracy?

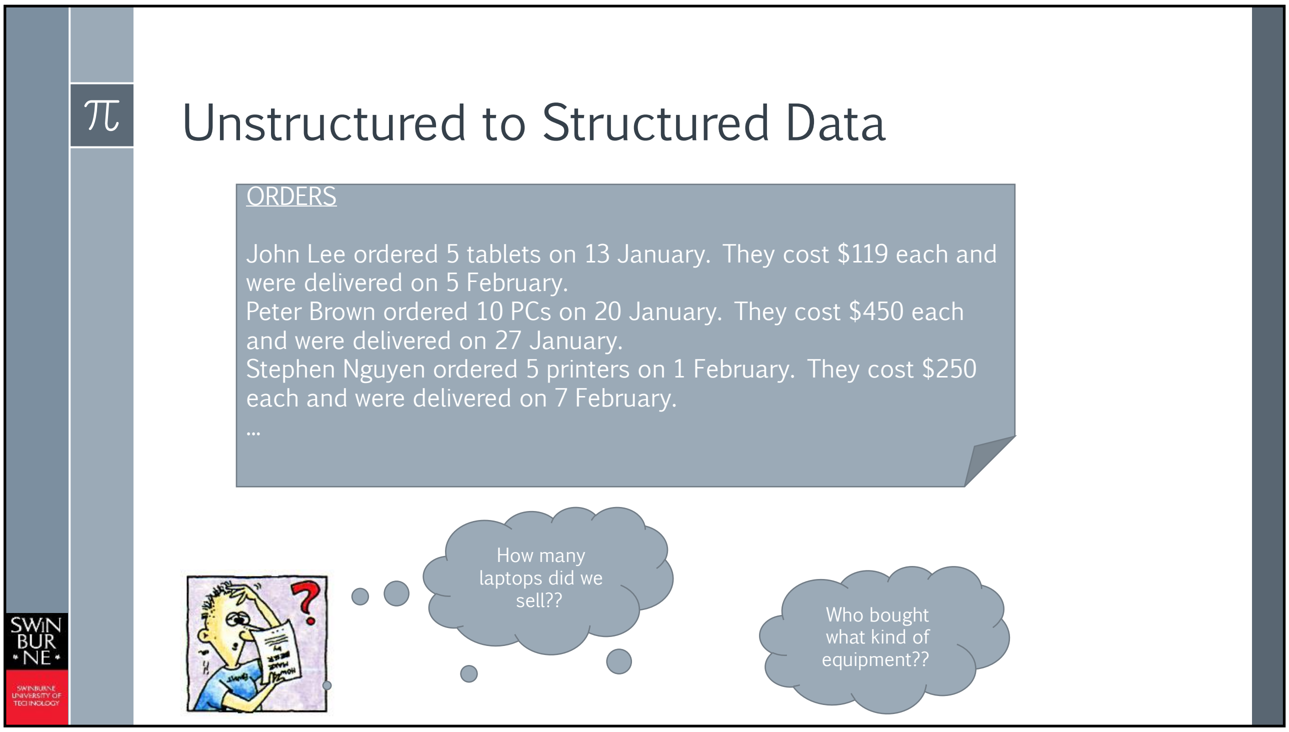
- Strict rules
  - Entity-relational modelling has strict rules that safeguard the data against error.
    - The modeler has to apply them correctly.
  - Entity-relational modelling strictly forbids duplication. ← redundancy
    - There can be no discrepancy between two pieces of data that mean the same thing.
- Exact definitions
  - Information is divided into the smallest possible pieces and exactly defined
    - These pieces can be manipulated by software,
    - No human intervention required, so no human error

4

Relational modelling has many rules and only works if done expertly. That's why we talk about it here in great detail.
One of the biggest risks of inaccuracy is duplication. If we record the same person's address twice in the same database, we might forget to update one of the entries when we want to change the addresses. Relational design talks of duplication as 'redundancy' and regards databases with redundant data as poor design.

Relational DBMSs have many tools to automate calculations and extractions of facts from databases. This reduces the need for manual work and therefore, the risk of human error.

Attributes and Entities



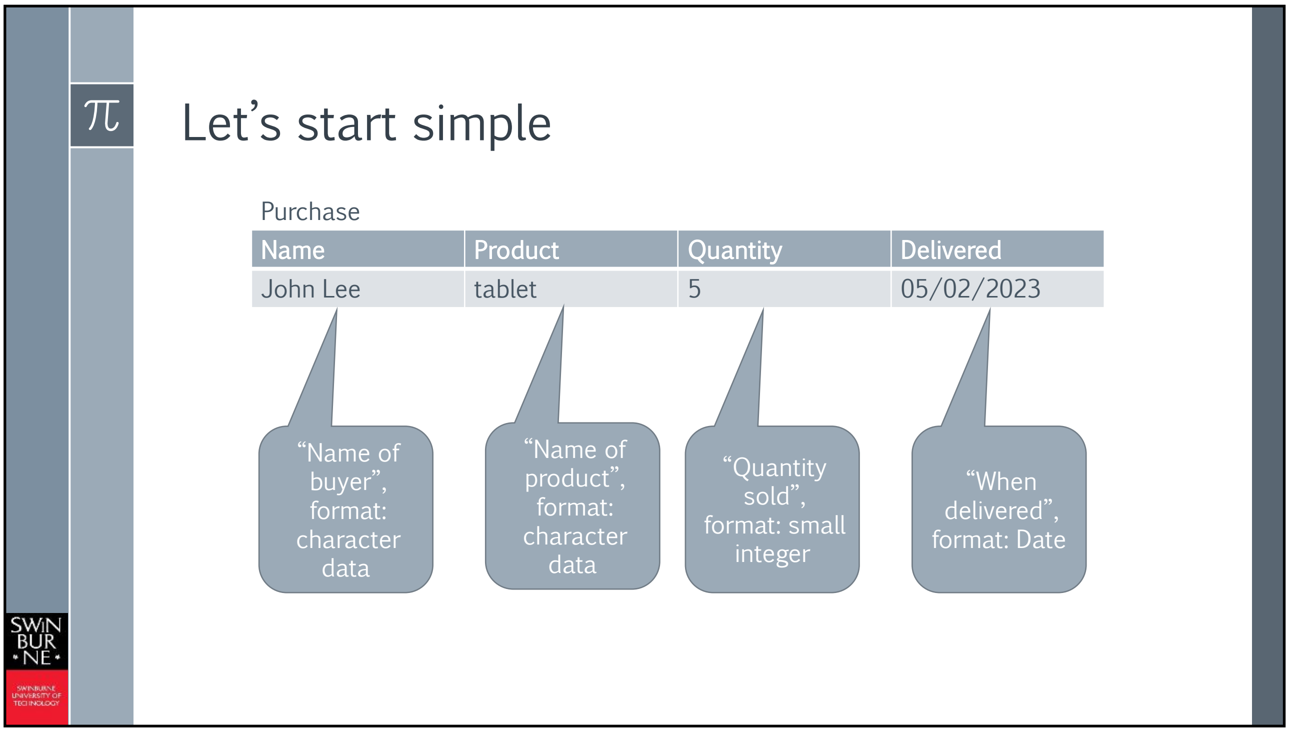
Unstructured to Structured Data

ORDERS

John Lee ordered 5 tablets on 13 January. They cost $119 each and were delivered on 5 February.
Peter Brown ordered 10 PCs on 20 January. They cost $450 each and were delivered on 27 January.
Stephen Nguyen ordered 5 printers on 1 February. They cost $250 each and were delivered on 7 February.
…

How many laptops did we sell??
Who bought what kind of equipment??

Many IT projects start small with a business needing to automate some of their processes. They come and ask you to make a nifty app to make this easier. You realise the first thing you'll need is a database. You decide which database product you'll use (let's skip this bit), and then you'll have to model.
So how do you start?
You start by asking lots of questions – questions about questions. What questions does the business want answered with your database? How much did we sell? How much did we earn? How much was our profit? Who were our best customers? What products did we sell the most of?
If your model is good, at the end the database will be able to answer the questions year after year.



Let's start simple

Purchase

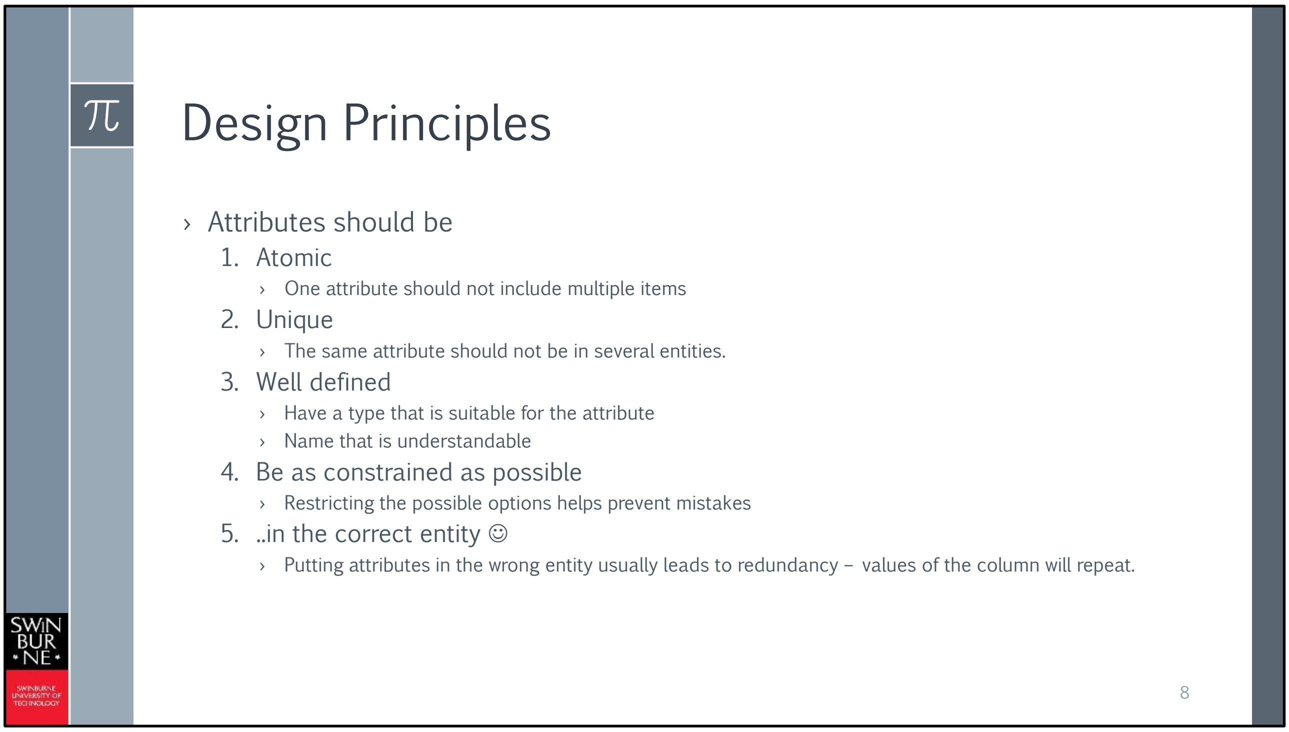
Name	Product	Quantity	Delivered
John Lee	tablet	5	05/02/2023

"Name of buyer", format: character data
"Name of product", format: character data
"Quantity sold", format: small integer
"When delivered", format: Date

At first, we decide what our entities are. As we usually do, we imagine them to be table rows. Our example is about people buying IT equipment, so one of our entities might just be about a person buying a quantity of some product. So we could name this collection of entities, or table, our purchase table. Note that the convention is to name tables in the singular (Purchase, not Purchases).

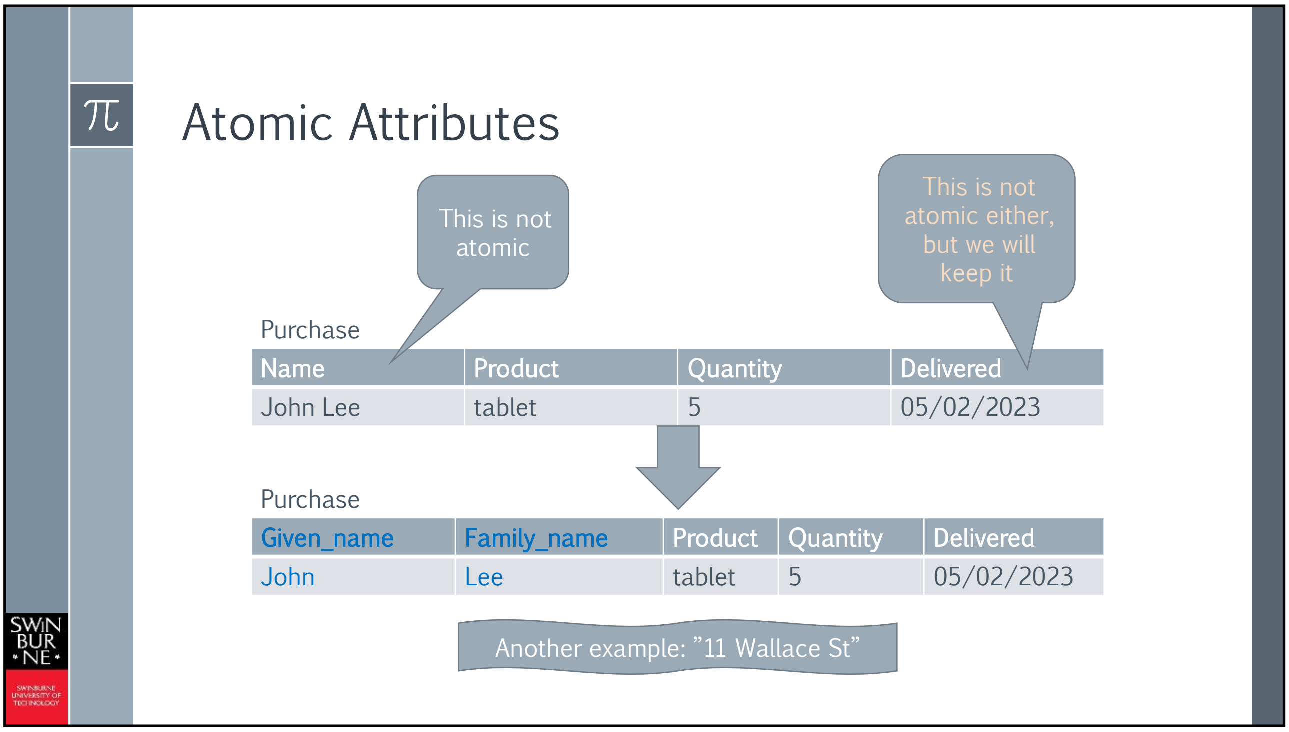
We can reasonably assume that the two questions ”How many laptops did we sell” and ”Who bought what equipment” can be answered with this data, once we have added all entities of people buying equipment over the period.

Naturally, if another question is, how much revenue did this raise, we have to ensure we include prices.


Design Principles

- Attributes should be
  1. Atomic
     - One attribute should not include multiple items
  2. Unique
     - The same attribute should not be in several entities.
  3. Well defined
     - Have a type that is suitable for the attribute
     - Name that is understandable
  4. Be as constrained as possible
     - Restricting the possible options helps prevent mistakes
  5. ..in the correct entity ☺
     - Putting attributes in the wrong entity usually leads to redundancy – values of the column will repeat.

Good attribute design abides by these principles
Atomic means you cannot divide it further. If your attribute includes city and street address, we can argue that we can subdivide them into these.
Unique means, the same attribute should not be found in different entities. Every piece of data should appear only once.
Defining an attribute well means, for example, not using an Integer type for a phone number, or not using ambiguous names for attributes. Different attributes should not be named the same. It is helpful to adopt a naming convention.
Reducing the options of values that are accepted into a field is a good way of maintaining database integrity.
The question which attributes belong in which entity is tightly linked to the concept of a primary key.



Atomic Attributes

This is not atomic
This is not atomic either, but we will keep it

Purchase

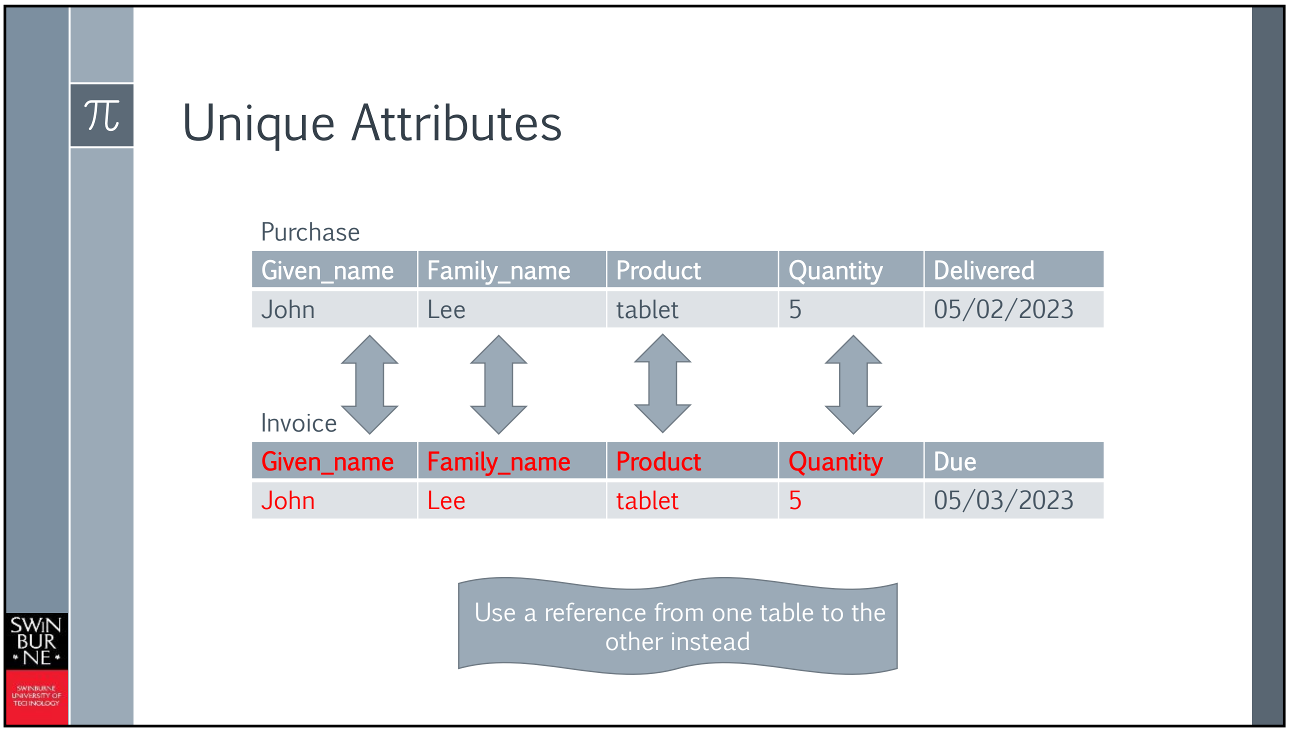
Name	Product	Quantity	Delivered
John Lee	tablet	5	05/02/2023

Purchase

Given_name	Family_name	Product	Quantity	Delivered
John	Lee	tablet	5	05/02/2023

Another example: ”11 Wallace St”

Combining first and last names in the same attribute is not convenient. You can think of many examples where you need to know which is which, e.g. to search for a person whose first name is John and not last name.
You may consider subdividing dates into days, months and years. This is a valid thought, because on occasion you may want to pick only the year or year and month. However, date manipulation is a well-established feature in all database management systems, where the extraction of years or days of week are all provided as functions. DBMSs usually also let you subtract dates to calculate a duration, and they provide the date format according to your language settings, on top of taking care the date is valid. If you split up the date into its parts, all of these features have to be provided by you, so this is not practical.
Another example where we deviate from the atomic requirement is street addresses: 11 Wallace St could be devided into house number and street, but this is usually not done, because normally people do not need the house number separately from the street – they are usually used together. However, if you have a situation where dividing them is warranted, do it – the design just has to make sense and work in all use cases.



Unique Attributes

Purchase

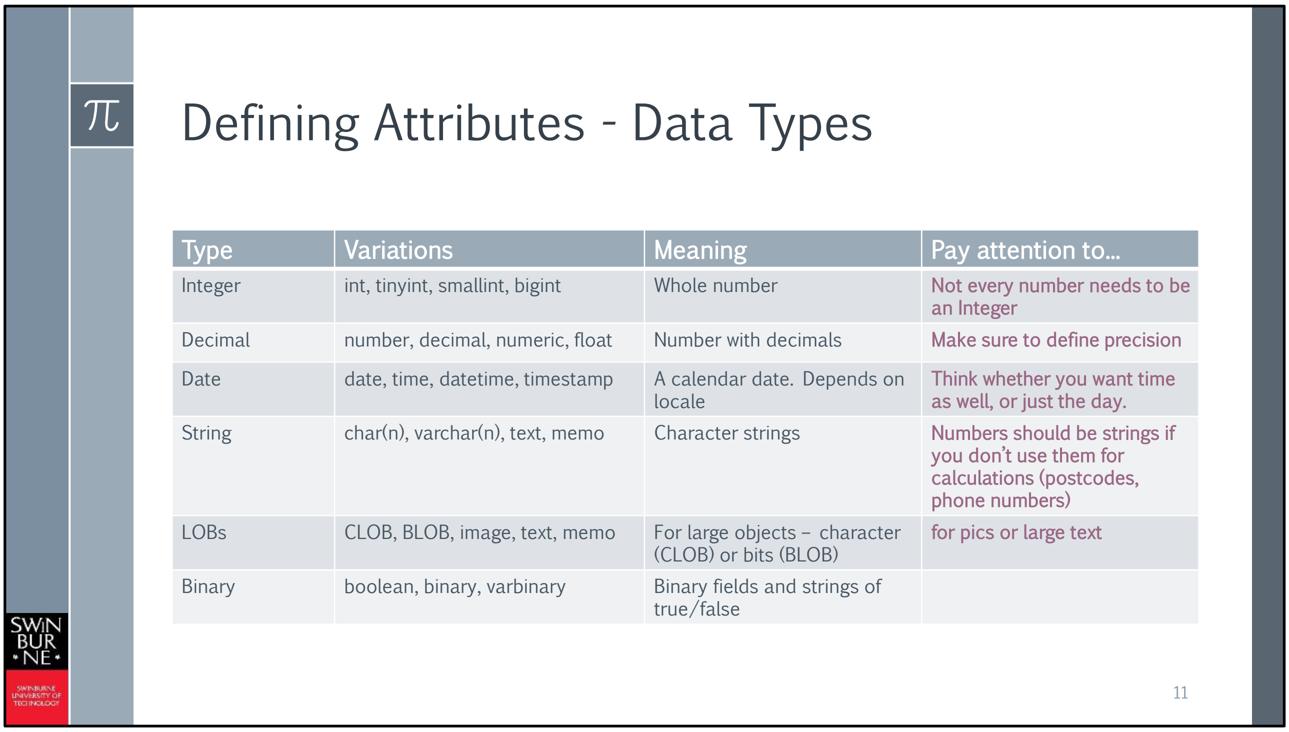
Given_name	Family_name	Product	Quantity	Delivered
John	Lee	tablet	5	05/02/2023

Invoice

Given_name	Family_name	Product	Quantity	Due
John	Lee	tablet	5	05/03/2023

Use a reference from one table to the other instead

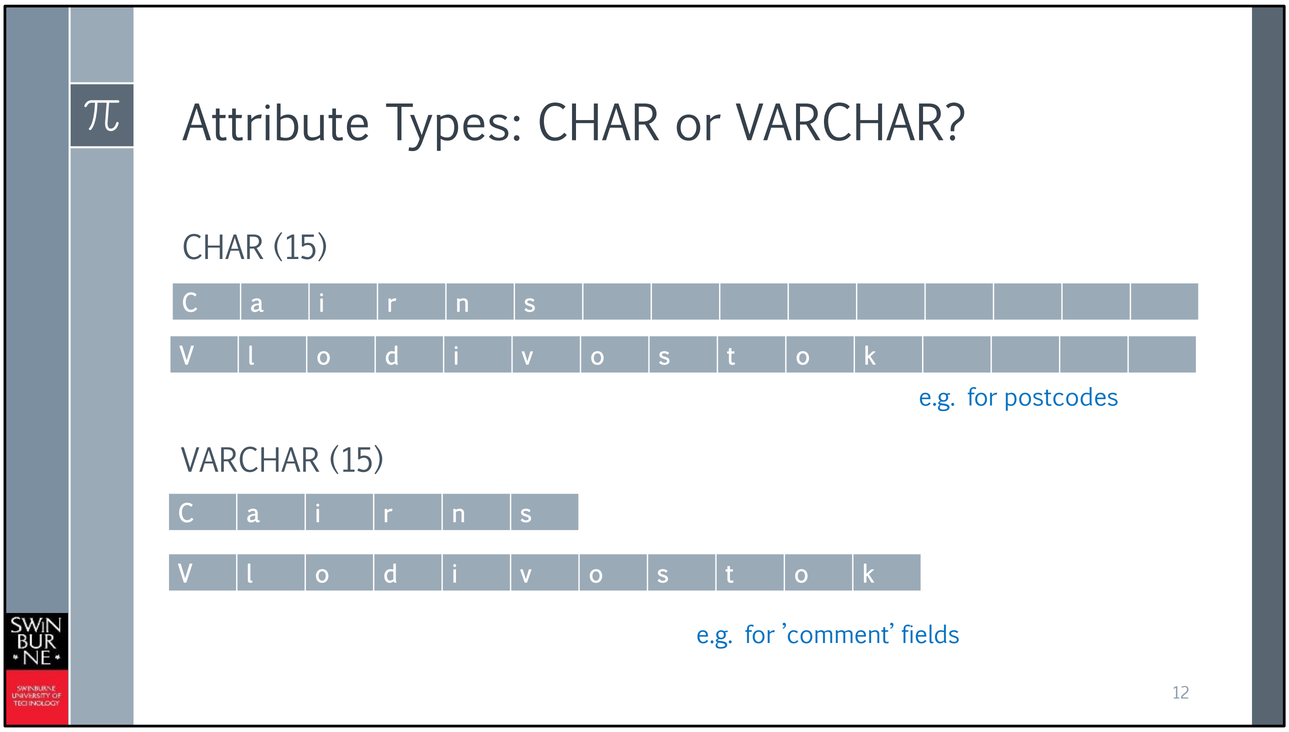
Repeating the same data in different places is always a sign of relational design gone wrong. The way to fix this problem is by creating a slightly different invoice table that references Purchase, but relationships between tables are for another module, so we won't go there now.


Defining Attributes - Data Types

Type	Variations	Meaning	Pay attention to…
Integer	int, tinyint, smallint, bigint	Whole number	Not every number needs to be an Integer
Decimal	number, decimal, numeric, float	Number with decimals	Make sure to define precision
Date	date, time, datetime, timestamp	A calendar date. Depends on locale	Think whether you want time as well, or just the day.
String	char(n), varchar(n), text, memo	Character strings	Numbers should be strings if you don't use them for calculations (postcodes, phone numbers)
LOBs	CLOB, BLOB, image, text, memo	For large objects – character (CLOB) or bits (BLOB)	for pics or large text
Binary	boolean, binary, varbinary	Binary fields and strings of true/false	

11

Every DBMS has its own version of the same datatypes, but there can be small differences. It may be worth checking when you work with a new product.
In principle, you always want to spend the minimal possible space for each attribute, but it should not turn out to be too little space later on, so we have to think about it a bit.
Decimals are needed for a price field, unless the DBMS comes with an inbuilt type for currency. If it doesn't, you have to define the precision, otherwise your price calculations have unexplained results.
Attributes like phone numbers should not be Integers. Integers don't allow for leading zeros, and definitely not for plus signs, which are used in international area codes.
Similarly, some countries have postcodes with leading zeros.
If a field is not meant for calculations, it should be a string type.



Attribute Types: CHAR or VARCHAR?

CHAR (15)

C	a	i	r	n	s									
V	l	o	d	i	v	o	s	t	o	k				

e.g. for postcodes

VARCHAR (15)

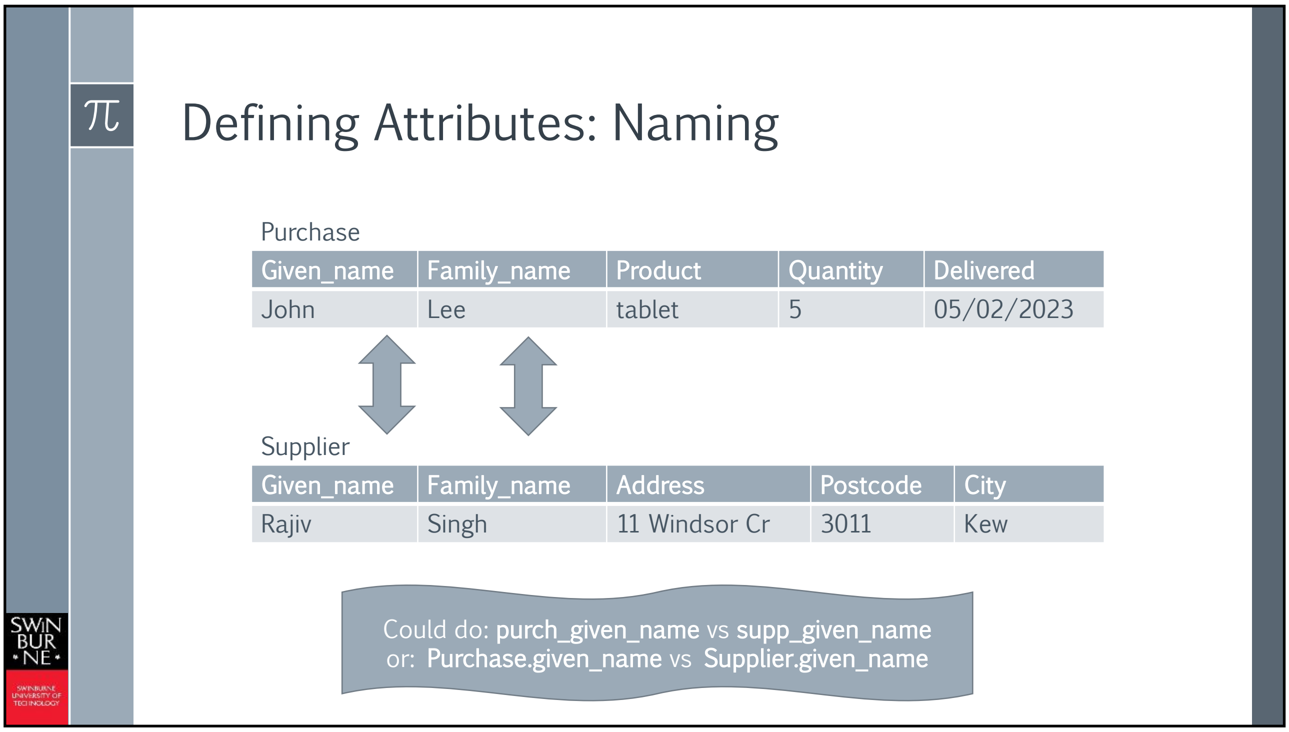
C	a	i	r	n	s

V	l	o	d	i	v	o	s	t	o	k

e.g. for 'comment' fields

12

The Char type maintains the same space at all times, even if the entry is a lot shorter. Many people think that's a waste, and use VARCHAR all the time. That's not a good choice.
If you later overwrite a varchar with a longer string, chances are it does not fit where it belongs in the entry, and will be amended after the entity. That slows down search operations.



Defining Attributes: Naming

Purchase

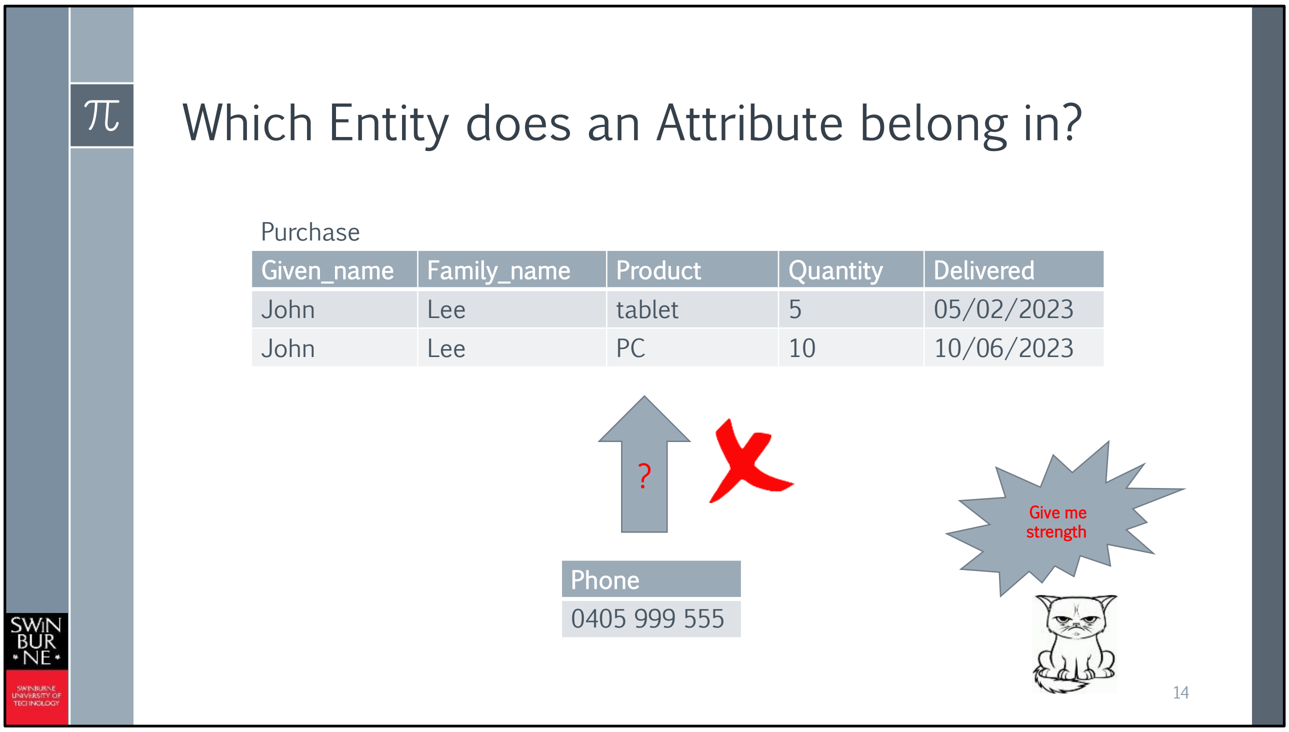
Given_name	Family_name	Product	Quantity	Delivered
John	Lee	tablet	5	05/02/2023

Supplier

Given_name	Family_name	Address	Postcode	City
Rajiv	Singh	11 Windsor Cr	3011	Kew

Could do: **purch_given_name** vs **supp_given_name**
or: **Purchase.given_name** vs **Supplier.given_name**

Although we are not repeating information here, as buyers are different from suppliers, we have a naming problem. These columns could be mixed up in a query. The best way to avoid this is to adopt a naming convention that includes the table name.
Another option is to adopt a convention of always prefixing the attribute name with the table (or entity) name: Purchase.given_name. The advantage is that most DBMSs understand this notation, you can use it in queries.
Traditionally, in databases, names use underscores to combine two words. More recently, camel case is used as an alternative. Choose either one and stick to it.



Which Entity does an Attribute belong in?

Purchase

Given_name	Family_name	Product	Quantity	Delivered
John	Lee	tablet	5	05/02/2023
John	Lee	PC	10	10/06/2023

?

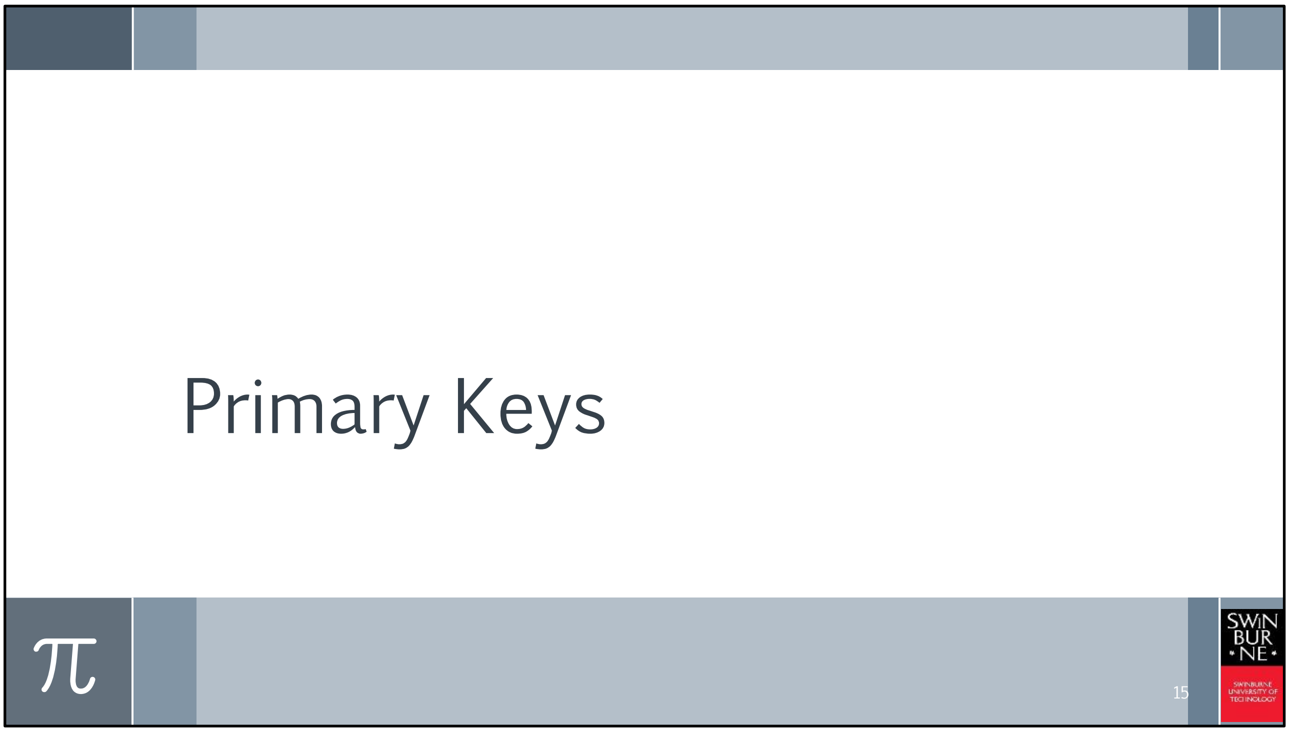
Phone
0405 999 555

Give me strength

14

Let's take another look at our trusty Purchase table. It is very imperfect yet, but all attributes that are in it, do logically belong there. A purchase logically included the name of the customer, the product bought, how many were bought and when they were delivered. So far so good. I have added a second purchase for the same customer to show customers can buy repeatedly.
But what happens if we need to store the phone number of the customer? Can we put a phone number into the purchase table?
It turns out that this would be absolutely awful design.
Why is this? Will the purchase cause the phone number to change? No, it will not. If the phone number changes, this is nothing to do with the purchase. Only organised crime use one burner phone for each job.
If we put the phone number into the Purchase entity, we are very likely recording the same phone number again and again. This is redundancy, and against relational design principles, as redundancies can lead to inconsistencies. We might make a mistake in recording the phone number, then we have two phone numbers even when the number has not changed.
You might be terribly confused now and say, but the name of the person keeps repeating anyway and that is fine? This is a valid question. The answer is, we need the name to identify the person. We should not need the phone number to identify the customer.
We will revisit this situation when we have learned a bit more about

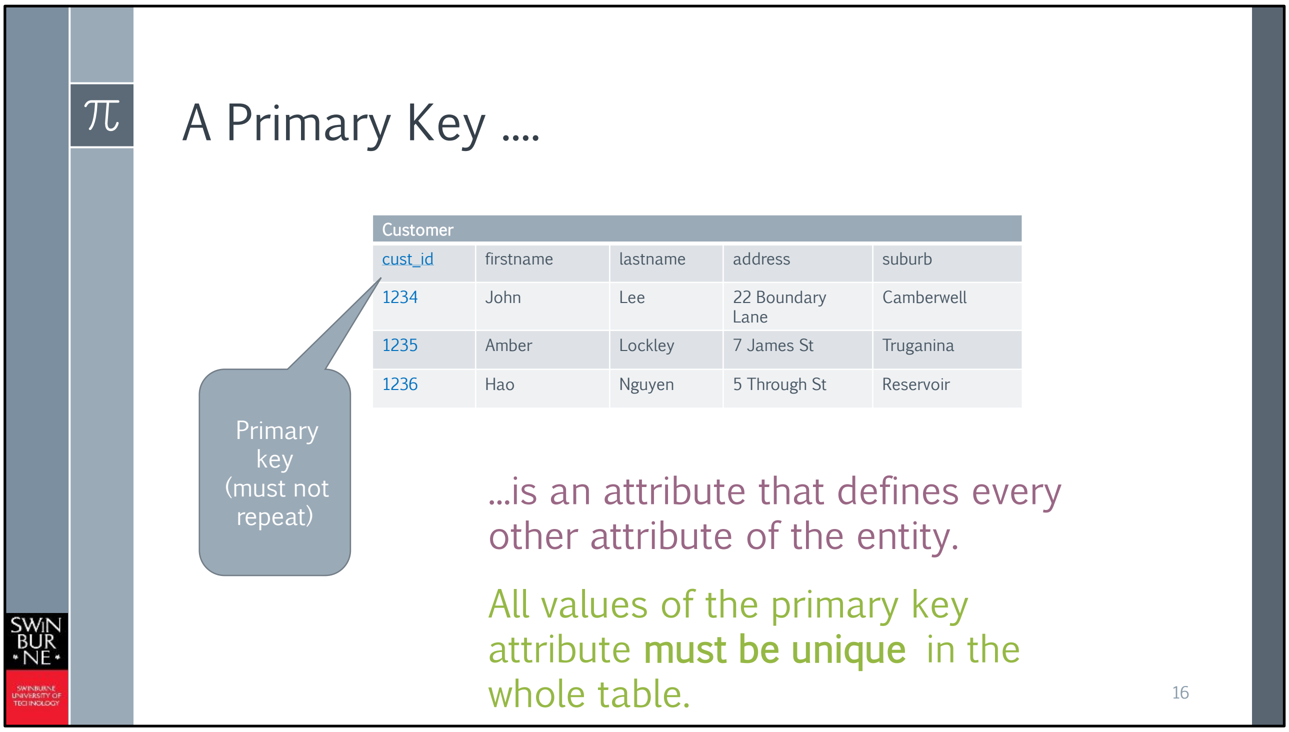
relational modelling.



Primary Keys

15

The question where an attribute belongs is linked to the question of primary keys, because primary keys define the entity. If you have a good understanding how to choose a primary key for an entity, you are most likely also able to assess whether an attribute belongs in an entity.



A Primary Key ....

Customer				
cust_id	firstname	lastname	address	suburb
1234	John	Lee	22 Boundary Lane	Camberwell
1235	Amber	Lockley	7 James St	Truganina
1236	Hao	Nguyen	5 Through St	Reservoir

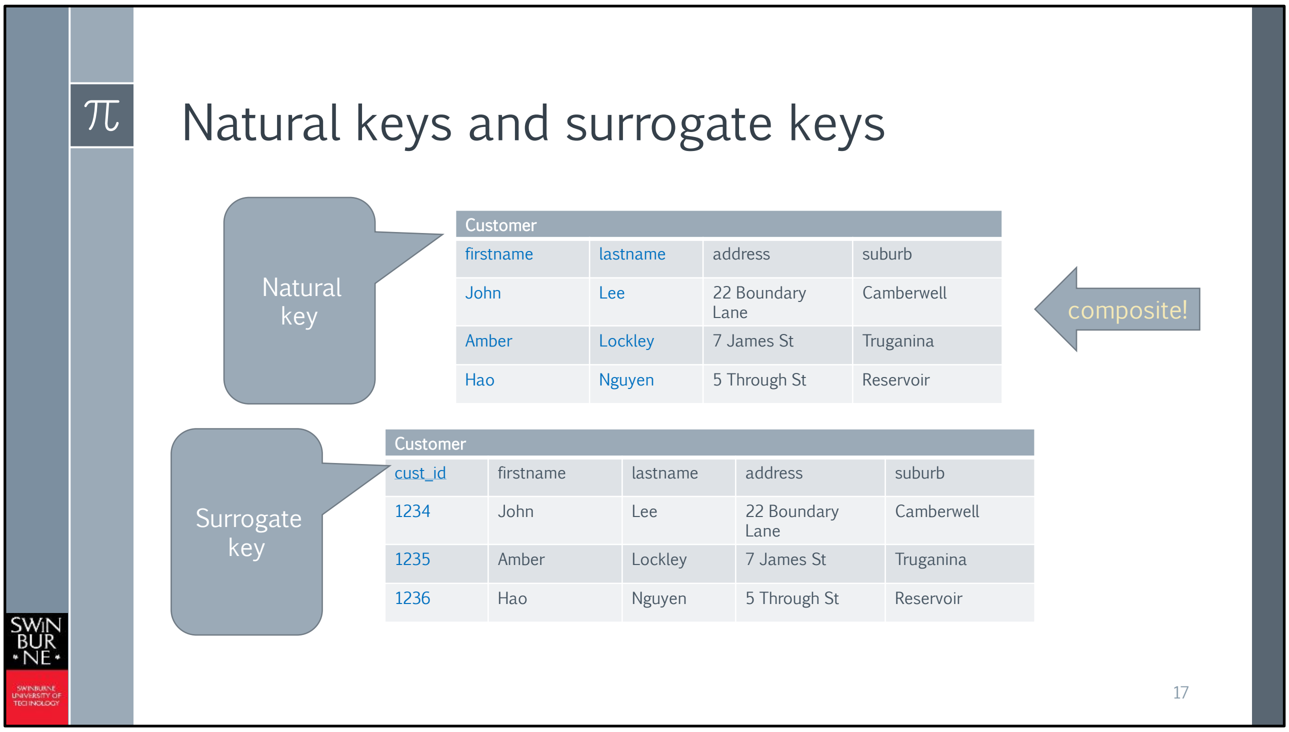
Primary key (must not repeat)

...is an attribute that defines every other attribute of the entity.

All values of the primary key attribute **must be unique** in the whole table.

16

A primary key is an attribute or a column of a table. If the entity has been designed according to relational principles, every other attribute value of an entity depends on the value of the primary key.
What does this mean?
Our example here is a customer table, where we have added a customer id as a primary key. Because the PK is the defining variable, it cannot repeat in the table.
So how does the PK define all other attributes?
If we know that the ID is 1234, we know that the customer is John Lee, and we know that John Lee's address is Boundary Lane in Camberwell. So customer 1234 cannot have other values, unless John Lee moves house.



Natural keys and surrogate keys

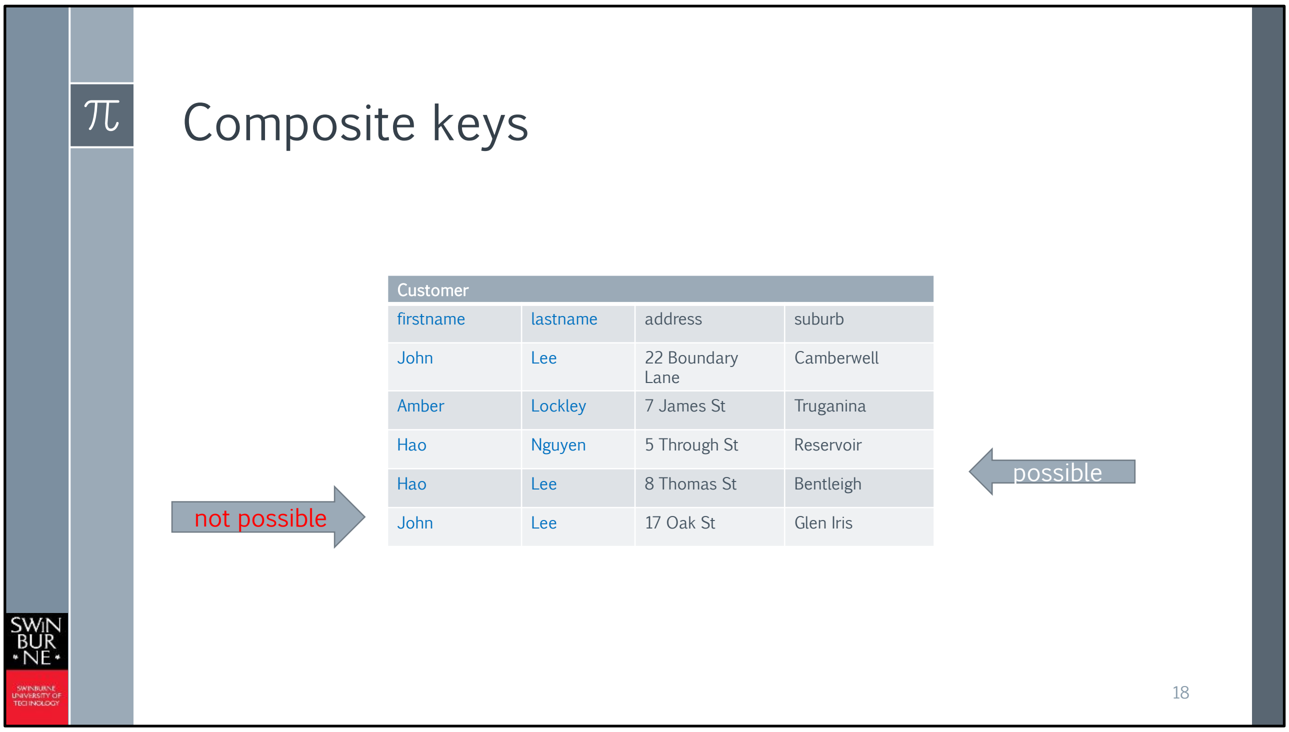
Natural key
composite!
Surrogate key

Customer			
firstname	lastname	address	suburb
John	Lee	22 Boundary Lane	Camberwell
Amber	Lockley	7 James St	Truganina
Hao	Nguyen	5 Through St	Reservoir

Customer				
cust_id	firstname	lastname	address	suburb
1234	John	Lee	22 Boundary Lane	Camberwell
1235	Amber	Lockley	7 James St	Truganina
1236	Hao	Nguyen	5 Through St	Reservoir

17

A primary key can be a natural key or a surrogate key.
A natural key is a key that is composed of attributes that naturally are part of the entity. Choosing both first name and last name, in combination, is an option for a key to a customer table. Obviously, there is the danger that two customers have exactly the same name, which will cause problems.
As you know, primary keys must not repeat, so if we have two John Lees, we will only be able to add one John Lee to the customer table. So we will have to refuse selling to a customer who has the same name as another customer.
This is not a good way to be, therefore most relational designers resort to adding a surrogate key, which is an artificial column whose values have no meaning of their own – they are just IDs to define every row.
Natural keys can be composite keys. In this case, if you choose the person's name as a key, you need first and last, so you have to make a composite key.
Some entities have natural single-attribute keys, such as ISBN numbers for books.

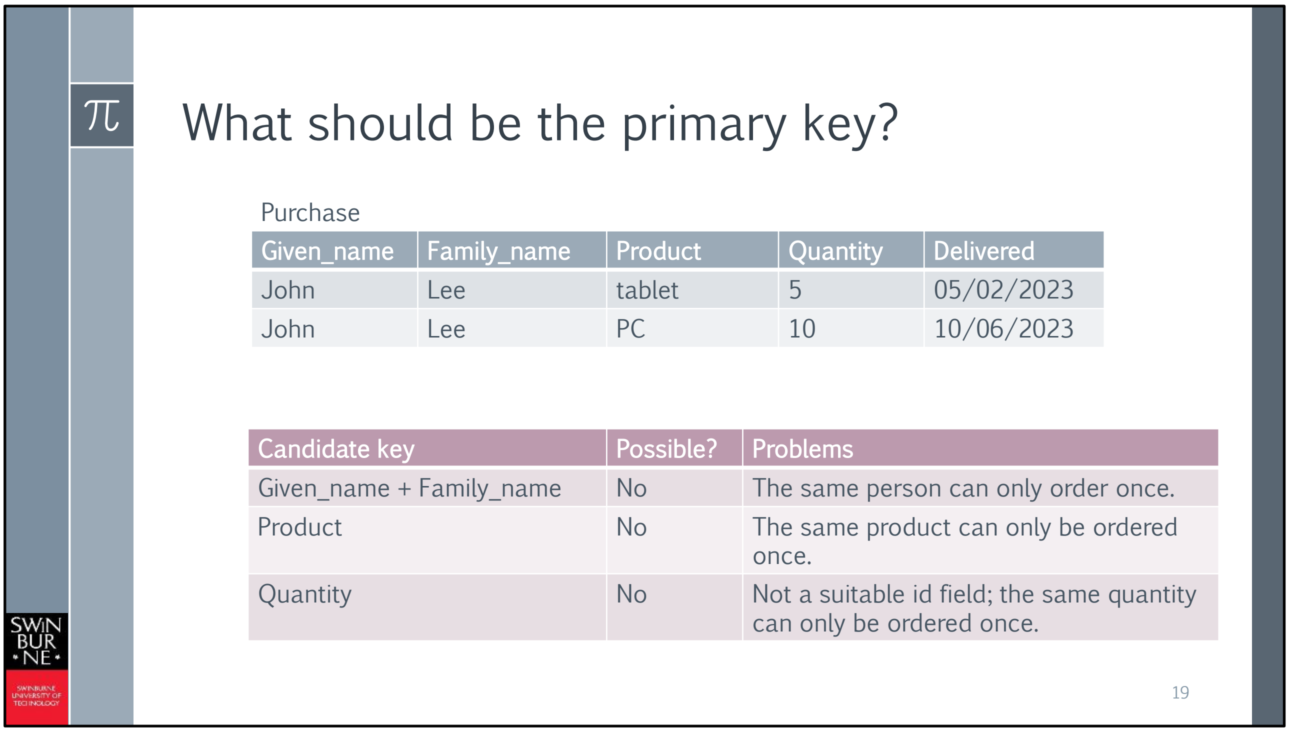


Composite keys

Customer			
firstname	lastname	address	suburb
John	Lee	22 Boundary Lane	Camberwell
Amber	Lockley	7 James St	Truganina
Hao	Nguyen	5 Through St	Reservoir
Hao	Lee	8 Thomas St	Bentleigh
John	Lee	17 Oak St	Glen Iris

not possible
possible

Note that when a key consists of several attributes, the combination of values has to be unique, not just one of the values.
So if we have another person with the last name of Lee, we're fine. But we can't have another John Lee if we decide that the first + last name combination is our primary key.



What should be the primary key?

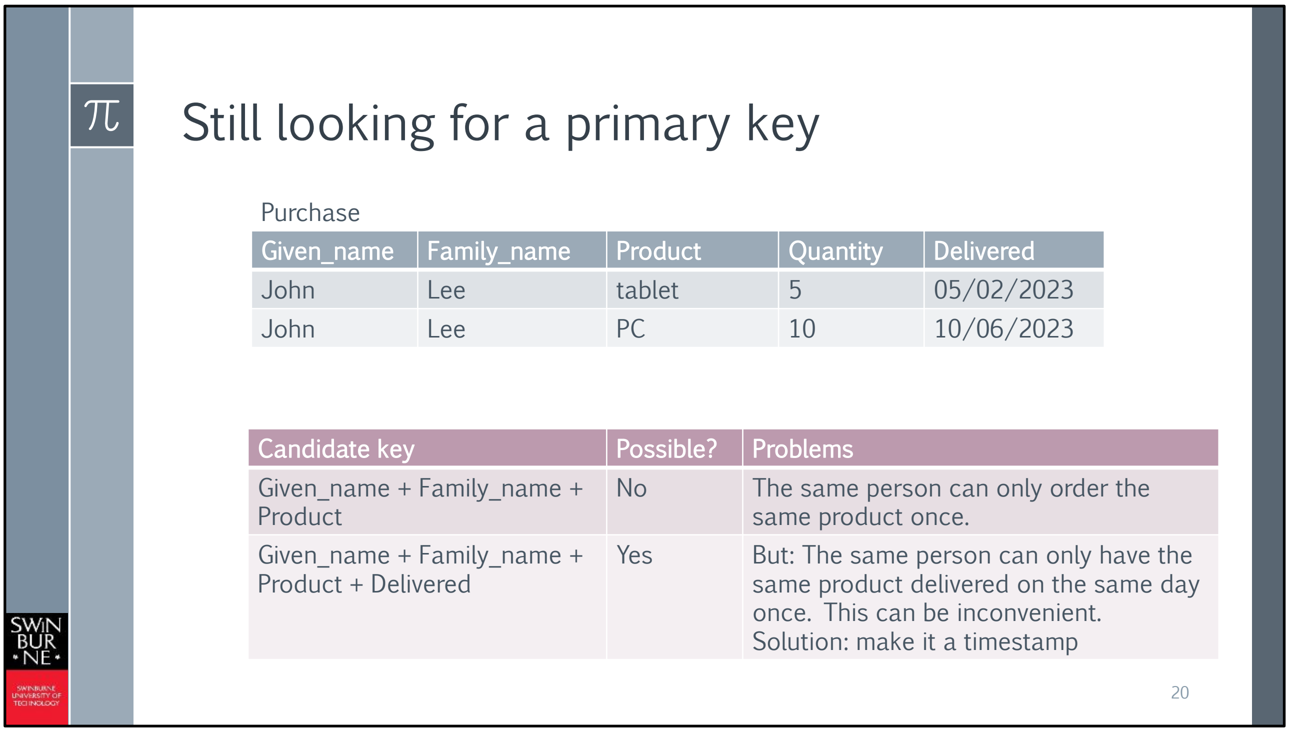
Purchase

Given_name	Family_name	Product	Quantity	Delivered
John	Lee	tablet	5	05/02/2023
John	Lee	PC	10	10/06/2023

Candidate key	Possible?	Problems
Given_name + Family_name	No	The same person can only order once.
Product	No	The same product can only be ordered once.
Quantity	No	Not a suitable id field; the same quantity can only be ordered once.

19

We are now very familiar with the Purchase table, but we haven't defined a primary key for it yet. Unless this table remains the only entity in your database, we do have to define one. It's not an easy entity to pick a primary key for.
For the moment, we won't add a surrogate key, because this does not help us learn how to make good primary keys and how to understand the essence – or the identity – of an entity.
Among the existing attributes, which would be suitable as an id for the entity?
A primary key has to be unique, which means if you choose an attribute for the key, it's values cannot repeat.
Considering the Quantity field, this is clearly not suited as an identifier, because it merely states how many of something someone has bought.



Still looking for a primary key

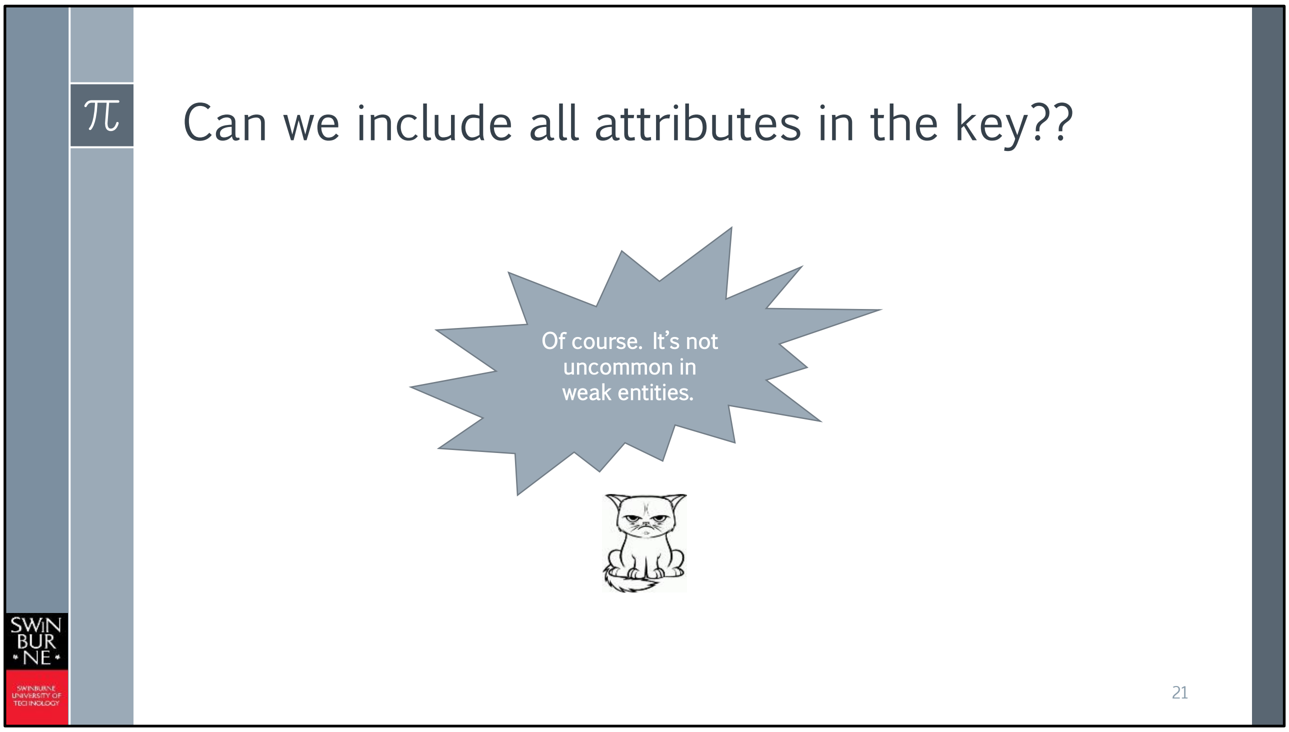
Purchase

Given_name	Family_name	Product	Quantity	Delivered
John	Lee	tablet	5	05/02/2023
John	Lee	PC	10	10/06/2023

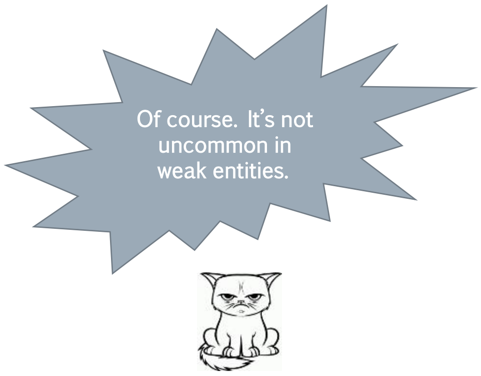
Candidate key	Possible?	Problems
Given_name + Family_name + Product	No	The same person can only order the same product once.
Given_name + Family_name + Product + Delivered	Yes	But: The same person can only have the same product delivered on the same day once. This can be inconvenient. Solution: make it a timestamp

20

If we have to make do with the attributes we have, and are not allowed to add new attributes, we have to use almost all the attributes in the table to make a functional key.
If we use given_name, family_name and product, the same person will never be able to buy the same product again.
If we add the delivered date to this key, we can make it work – but now we have to take care not to deliver the same product to the same customer on the same day. This might be a problem, because deliveries should be combined to save freight. But if we record the time in addition to the date, this design might actually work.

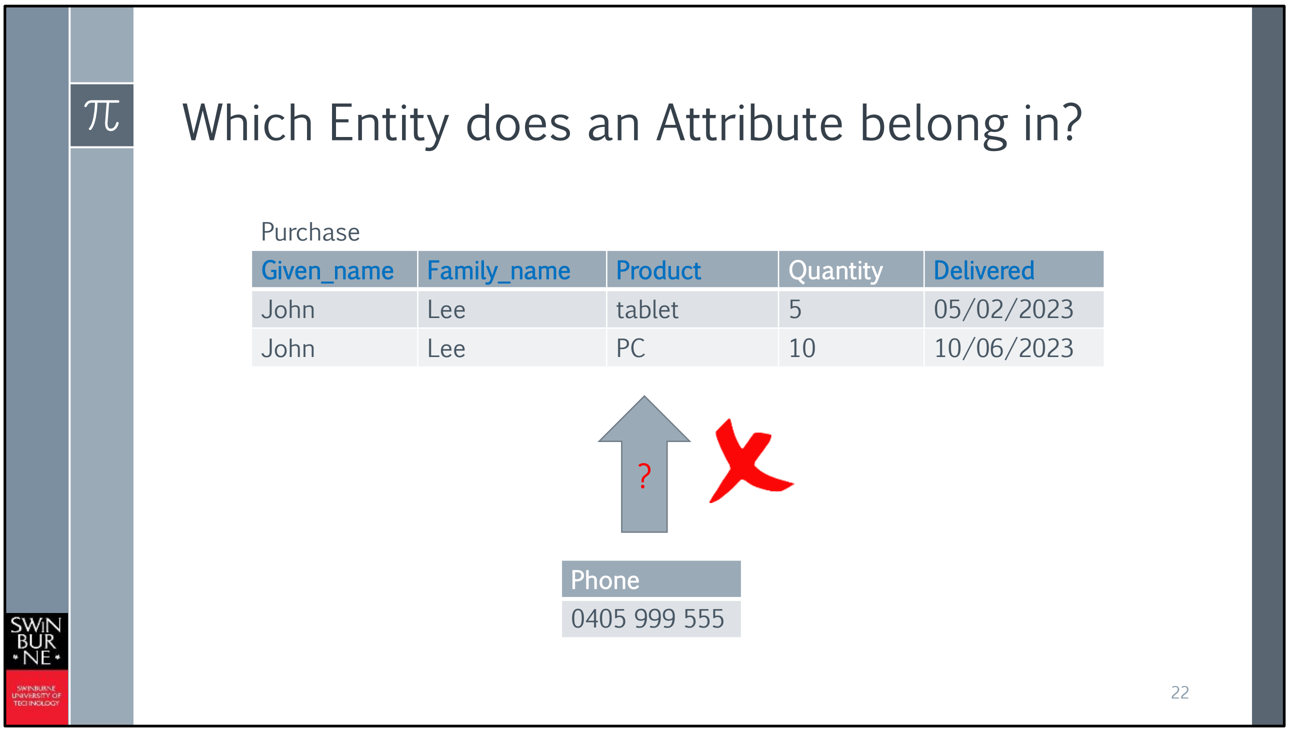


Can we include all attributes in the key??



21

We can make a composite key that consists of all the attributes in a relation. In the next module, we will have a look at weak and strong entities, and we'll see why using all attributes as part of our composite primary key is not that bad an idea.



Which Entity does an Attribute belong in?

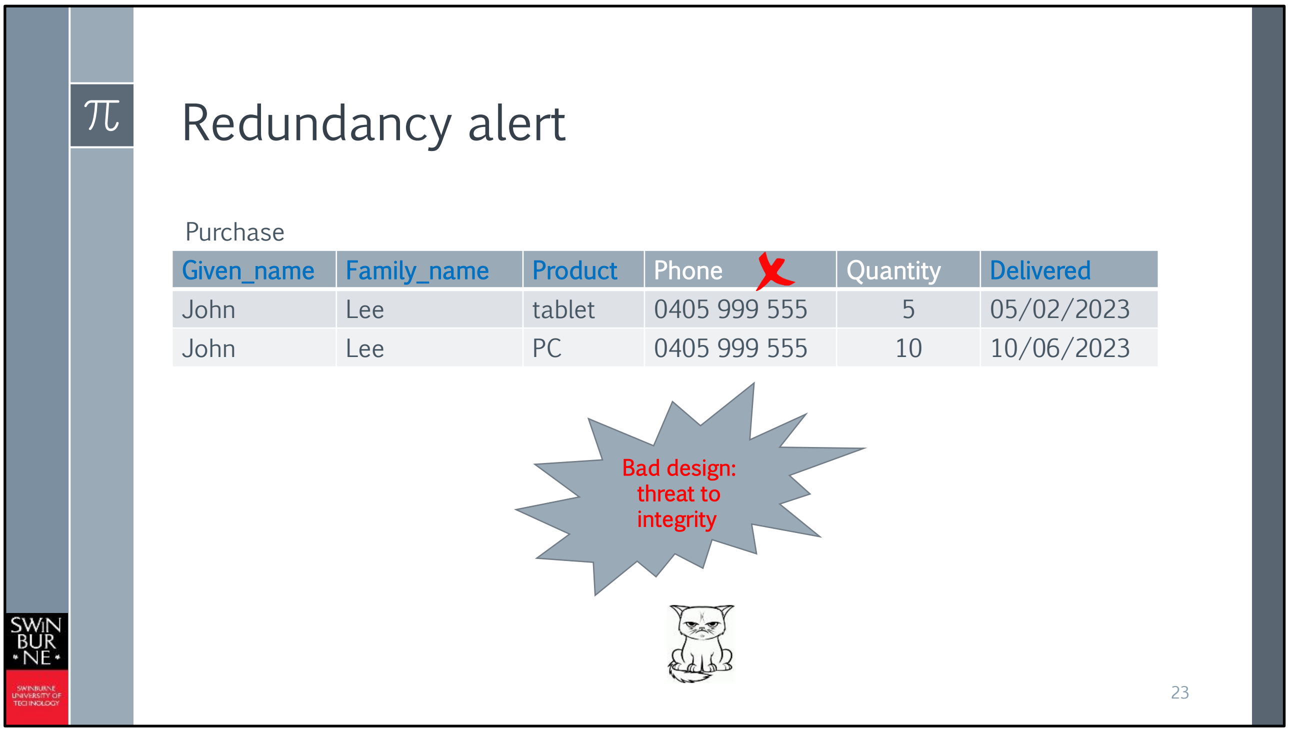
Purchase

Given_name	Family_name	Product	Quantity	Delivered
John	Lee	tablet	5	05/02/2023
John	Lee	PC	10	10/06/2023

Phone
0405 999 555

22

Back to our original question. We have now determined that the only possible key for this table is a combination of given_name, family_name, product and delivered.
This should help us decide if the phone number is fine to join this table.
To find out, we have to decide whether all parts of our key are needed to define the value of the phone number, and whether these parts of the key are sufficient to determine the phone number.
We have to ask ourselves, does the key depend on the given_name? We can say yes it does, if the given_name is used in combination with the family_name, as it is in this key. The given_name by itself is not sufficient. In actual fact, the full name is not really sufficient, because names often repeat, but we will ignore this problem for now. We will pretend that full names are unique.
Now we have to ask ourselves, does the product attribute in combination with the customer's name define the phone number? And here we have to say, absolutely not.
Does a phone number depend on what product the phone's owner is buying? Clearly not. The same applies to the 'delivered' date – it does not help us decide what someone's phone number should be.



Redundancy alert

Purchase

Given_name	Family_name	Product	Phone ✗	Quantity	Delivered
John	Lee	tablet	0405 999 555	5	05/02/2023
John	Lee	PC	0405 999 555	10	10/06/2023

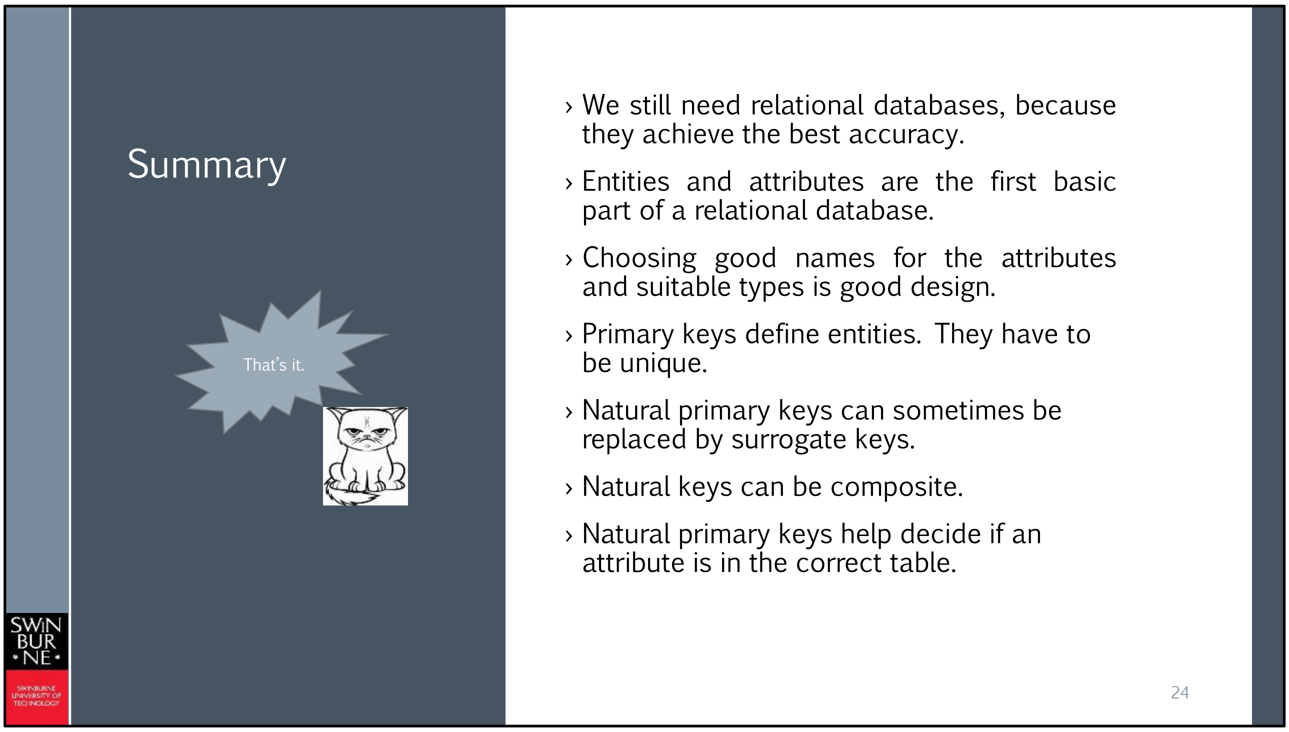
Bad design: threat to integrity

23

We have already had a little discussion about the evils of redundancy in relational databases. We said redundancy means the duplication of values, and duplication is dangerous, because if we need to update the value, we might not remember to update the value in all copies.
Therefore: Multiple copies of the same value are an indication of bad relational design. Bad design is a threat to database integrity.

The fact that the phone number repeats is another indicator that the phone sits in the wrong entity. As we have seen, only parts of the primary key determine this field (the full name of the owner), we have the same phone number again and again in this table, whenever the customer buys something.

This is why it is important to know the natural key, even if it is a complicated one. It helps us identify problems with our design.



Summary

- We still need relational databases, because they achieve the best accuracy.
- Entities and attributes are the first basic part of a relational database.
- Choosing good names for the attributes and suitable types is good design.
- Primary keys define entities. They have to be unique.
- Natural primary keys can sometimes be replaced by surrogate keys.
- Natural keys can be composite.
- Natural primary keys help decide if an attribute is in the correct table.

24

Here are the most important points discussed in this module. You may want to stop the recording to have a read through them. When you are ready, start the quiz about this module.