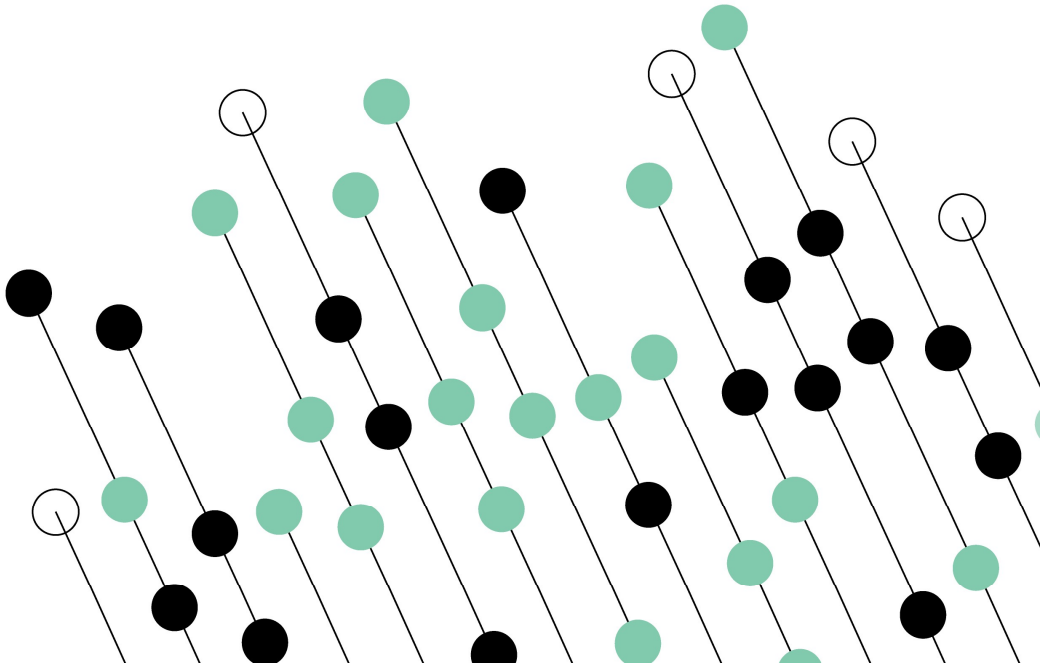
Normalisation



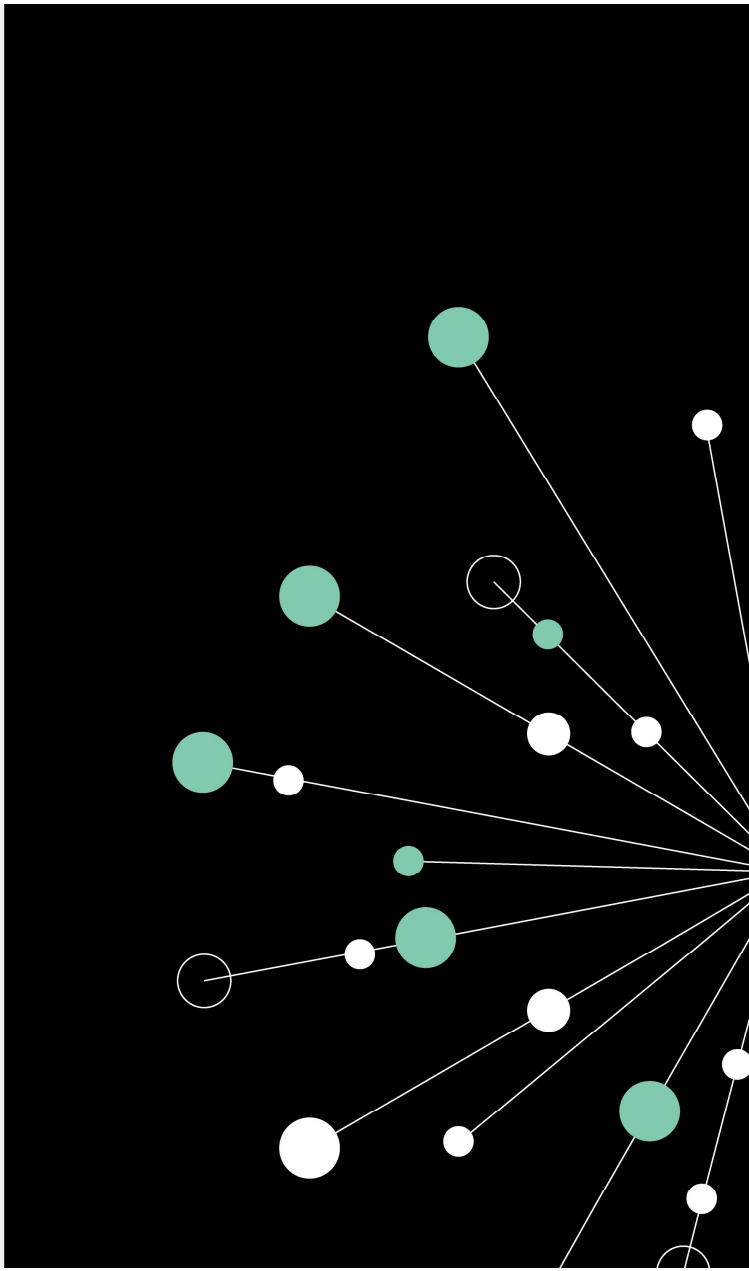
Data Normalisation

Objectives

- Gain an understanding of how to normalise data

Contents

- Overview
- First Normal Form
- Second Normal Form
- Third Normal Form



QA

Data Normalisation

Normalisation method:

- Approach to database design from ‘top-down’ or ‘bottom up’
- Useful for transforming existing files, checking existing databases
- Rigorous and formal approach
- Leads to increasing the number of tables, hence slowing down the querying of data
- Denormalisation provides a “trade-off” between eliminating redundancies and achieving acceptable query speed.

Data Normalisation

Benefits:

- Strategy for key and table selection
- Problems show up early in the design
- Eliminates insertion anomalies
- Eliminates deletion anomalies
- Reduction in database modification time

When can spreadsheets go wrong?

How could this Excel table become complicated to manage?

PatientID	PatientName	PatientAge	DoctorID	DoctorName	DoctorOffice
1	John Smith	43	101	Dr Holmes	11
2	Peter Brown	36	101	Dr Holmes	11
3	Emily Davidson	54	101	Dr Holmes	11
4	Samantha Newton	26	102	Dr Watson	12
5	George Elson	43	102	Dr Watson	12
6	Marie Goodwin	58	103	Dr Moriarty	13

The three anomalies:

1. Update anomaly
2. Insertion anomaly
3. Deletion anomaly

Update, insertion, delete anomalies

An **update anomaly** means that when updating a value, other values which should also be updated, are not changed.

Example: If Dr Holmes retires and is replaced by Dr Poirot, the doctor's name has to be changed for all his patients.

An **insertion anomaly** means it's not possible to introduce new entities into the database without adding other unrelated entities.

Example: If a new doctor starts working in the clinic but has not been assigned patients yet, it's not possible to add the doctor to the table without adding a non-existent patient.

A **deletion anomaly** means that the deletion of an entity may cause data about other entities to be lost.

Example: If the patient Marie Goodwin goes away and is removed from the patient list, Dr Moriarty is removed too because she is his only patient.

Un-normalised table

In an un-normalised table

- a primary key may not have been identified for the table.
- it may contain multi-valued columns.
- it may contain repeating columns.

Example:

product	Loyalty_price	Non_loyalty_price	colour
t-shirt	8.99	9.99	white
t-shirt	9.99	10.99	black, red
sweater	18.99	19.99	red, blue
sweater	19.99	20.99	black

- There is no primary key - none of the columns uniquely identify every row.
- There is a multi-valued column (colour) - there is no practical limit on the number of entries in the colour column a single row might have.
- There are repeating columns of different types of prices – loyalty and non-loyalty price.

When to apply normalisation

Normalisation can be applied both when designing the tables for a new project, or to check or transform existing files in databases.

Normalisation in the top-down approach

When the database tables have been designed from an ER model (‘top-down’) it is very important that these tables are normalised to eliminate redundancy and use space optimally.

This happens at the level of the Logical model.

Normalisation in the bottom-up approach

Normalisation can also be applied from the bottom up. This is when the design of the database is done by collating all the business reports and placing this information in database tables, which are then normalised.

Converting to first normal form (1nf)

The requirements for a database to be in first normal form are:

- Every table has a primary key.
- No column contains multiple values.
- No repeating groups of data. A table should not contain repeating columns.

Consider the un-normalised table:

product	Loyalty_price	Non_loyalty_price	colour
t-shirt	8.99	9.99	white
t-shirt	9.99	10.99	black, red
sweater	18.99	19.99	red, blue
sweater	19.99	20.99	black

There aren't any suitable primary keys, so it's necessary to introduce a surrogate primary key.

Converting to first normal form (1nf)

Surrogate primary key – column ID

ID	Product	Loyalty_price	Non_loyalty_price	colour
1	t-shirt	8.99	9.99	white
2	t-shirt	9.99	10.99	black, red
3	Sweater	18.99	19.99	red, blue
4	Sweater	19.99	20.99	black

When there are multi-valued columns, it is necessary to create a new table that will hold the related multi-valued columns.

Whenever you extract a multi-valued column from a table, you must always include a foreign key pointing back to the source row.

The two columns of the new table form a composite primary key.

Converting to first normal form (1nf)

In the example, it is necessary to introduce a new table for the values of column colour.

ID	Product	Loyalty_price	Non_loyalty_price
1	t-shirt	8.99	9.99
2	t-shirt	9.99	10.99
3	Sweater	18.99	19.99
4	Sweater	19.99	20.99

ID	Colour
1	White
2	Black
2	Red
3	Red
3	Blue
4	Black

A table should not contain repeating columns.

If there are any repeating groups of data (repeating columns), it is necessary to create a new table that will hold the related repeating groups.

In the example, Loyalty_price and Non_loyalty_price columns contain repeating groups of data (prices).

Converting to first normal form (1nf)

In the example, Loyalty_price and Non_loyalty_price columns contain repeating groups of data (prices). It is necessary to separate them into a new table.

ID	Product
1	t-shirt
2	t-shirt
3	Sweater
4	Sweater

ID	Colour
1	White
2	Black
2	Red
3	Red
3	Blue
4	Black

ID	type_price	value_price
1	L	8.99
1	NL	9.99
2	L	9.99
2	NL	10.99
3	L	18.99
3	NL	19.99
4	L	19.99
4	NL	20.99

Columns ID and type_price form a composite primary key

Second normal form (2NF)

A database is in second normal form if:

- **It satisfies the conditions of a first normal form (you cannot skip a normal form).**

AND

- **it has no partial dependency on a composite key.** This means that for a table that has a composite primary key, each column in the table that is not part of the primary key must depend upon the entire composite key.

The part of the primary key and the partially dependent column(s) are moved to another table.

Converting to second normal form (2NF)

Example: IDs and names of teachers and the courses they teach. A teacher can teach multiple courses.

TeacherID	Course	TeacherName
1	Databases	Edward Keane
1	Data Warehousing	Edward Keane
2	Visualisation	Stephen Raw
2	Power BI	Stephen Raw
3	Statistics	Wayne Jones
3	Machine Learning	Wayne Jones
3	Predictive Modelling	Wayne Jones

The primary key is composite, including columns TeacherID and Course.

The remaining column, TeacherName, depends only on TeacherID but not on Course. It is partially dependent on the composite primary key.

Converting to second normal form (2NF)

To resolve the partial dependency, TeacherName is put in a separate table together with TeacherID.

TeacherID	Course
1	Databases
1	Data Warehousing
2	Visualisation
2	Power BI
3	Statistics
3	Machine Learning
3	Predictive Modelling

TeacherID	TeacherName
1	Edward Keane
2	Stephen Raw
3	Wayne Jones

Third normal form (3NF)

A database is in third normal form if:

- **it satisfies the conditions of a second normal form.**

AND

- **no non-primary key column (attribute) depends on another non-primary key column (attribute).**

When converting to third normal form, the dependent non-key columns are removed to form a new table.

A column is promoted to be the primary key of the new table. This becomes a foreign key in the original table.

Converting to third normal form (3NF)

Example: In the table below, the primary key is CustomerID. CustomerCity depends (is uniquely determined) on CustomerPostcode.

CustomerID	CustomerName	CustomerCity	CustomerPostcode
1234	John Smith	Manchester	M1 3HL
5678	Paul Newman	Liverpool	L1 5XY
2468	Michael North	Manchester	M4 5LM
9876	Lydia Peters	Stockport	SK1 2BR

CustomerPostcode and CustomerCity need to be separated in a new table.

CustomerID	CustomerName	CustomerPostcode
1234	John Smith	M1 3HL
5678	Paul Newman	L1 5XY
2468	Michael North	M4 5LM
9876	Lydia Peters	SK1 2BR

CustomerPostcode	CustomerCity
M1 3HL	Manchester
L1 5XY	Liverpool
M4 5LM	Manchester
SK1 2BR	Stockport

Beyond 3NF

There are further normal forms:

- **Boyce – Codd**
- **Fourth Normal Form (4NF)**
- **Fifth Normal Form (5NF)**

However, they have primarily academic significance. They are almost never used in practice.

De-normalisation

The higher the normal form, the more the tables.

Each piece of data is reached through relationships to each table, joins. Normalisation reduces redundancy and maintains data integrity. On the other hand, the increased number of tables means that the queries are slower.

De-normalisation means knowingly disregarding some of the normalisation rules in order to achieve easier access to the data and faster queries.

De-normalisation: pros and cons

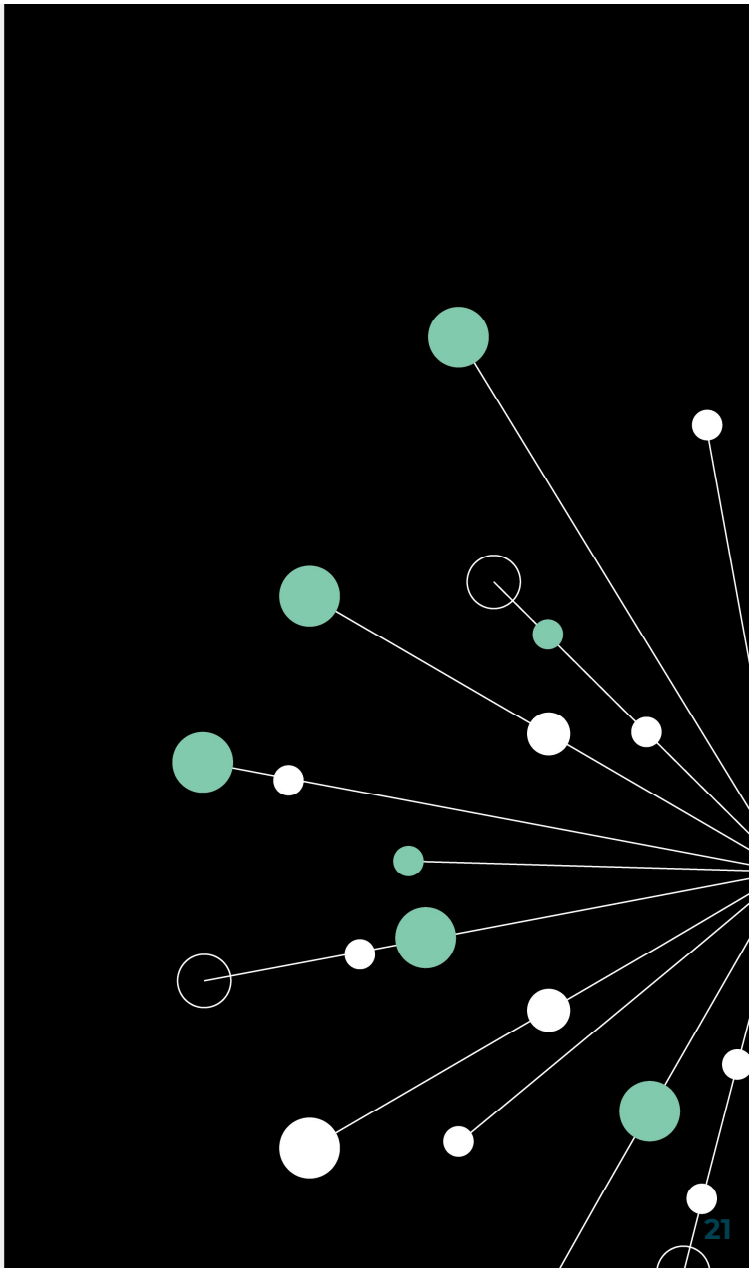
PROS:

- Retrieving data is faster since we do fewer joins.
- Queries to retrieve can be simpler (and therefore less likely to have bugs), since we need to look at fewer tables.

CONS:

- Updates and inserts are more expensive.
- Denormalisation can make *update* and *insert* code harder to write.
- Data may be inconsistent.
- Data redundancy necessitates more storage.

Example: From a spreadsheet to 3NF



Customers and orders: Excel style

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Ord. No.	Date	Cust. No.	Cust. Name.	Cust. Address	Cust. Town	Cust. County	Cust. PostCode	Prod. ID	Prod. Desc.	Prod. Qty.	Prod. Price	Prod. Amount	Ord. Total
2	100001	31/07/2011	8573	Games-R-Us	45 Shoppingt	Shoppingtc	Bucks.	SH8 2AB	849	Alpine Ski Instr	2	£29.99	£59.98	£836.80
3									773	SIM Accountan	28	£23.46	£656.88	£836.80
4									382	Recorder Hero	6	£19.99	£119.94	£836.80
5	100002	31/07/2011	5644	Gamerz	304 The Mall	Mallville	Herts.	MV9 1AQ	849	Alpine Ski Instr	200	£29.99	£5,998.00	£6,772.50
6									562	Deck Chair Att	50	£15.49	£774.50	£6,772.50
7														

A possible solution?

Representing repeated items is not a good idea.

What if a customer wants to order more than 3 items?

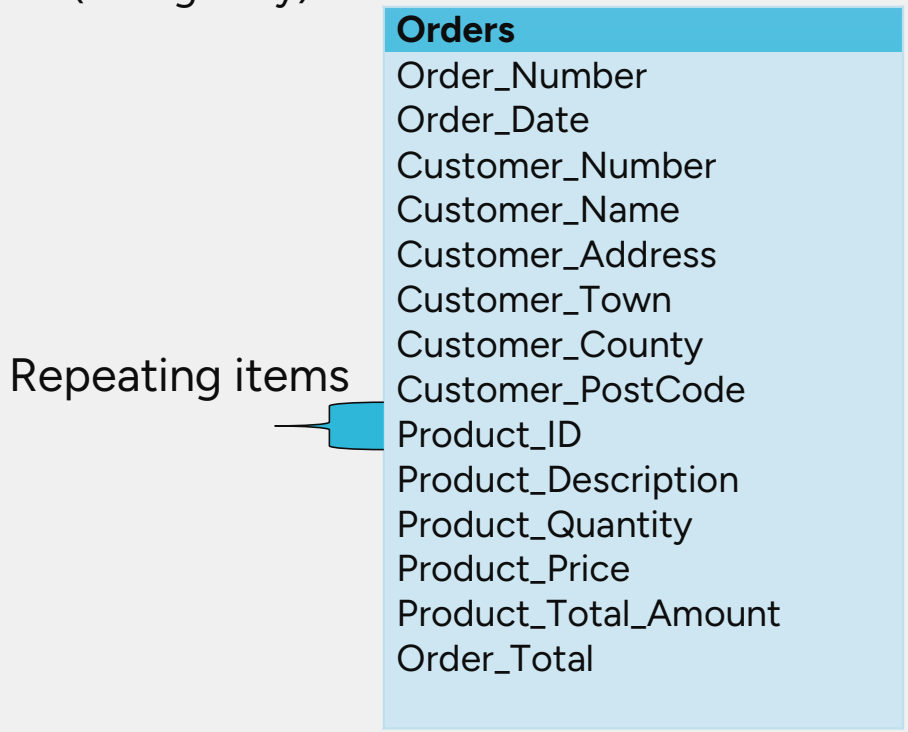
What if a customer wants only 1 item?

Orders	
Order_Number Order_Date Customer_Number Customer_Name Customer_Address Customer_Town Customer_County Customer_PostCode Product_ID_1 Product_Desc_1 Product_Quantity_1 Product_Price_1 Product_Tot_Amt_1	Product_ID_2 Product_Desc_2 Product_Quantity_2 Product_Price_2 Product_Tot_Amt_2 Product_ID_3 Product_Desc_3 Product_Quantity_3 Product_Price_3 Product_Tot_Amt_3 Order_Total

Converting to first normal form (1NF)

We have to:

- remove the repeating group of attributes to a new table.
- add the original key to it (foreign key).

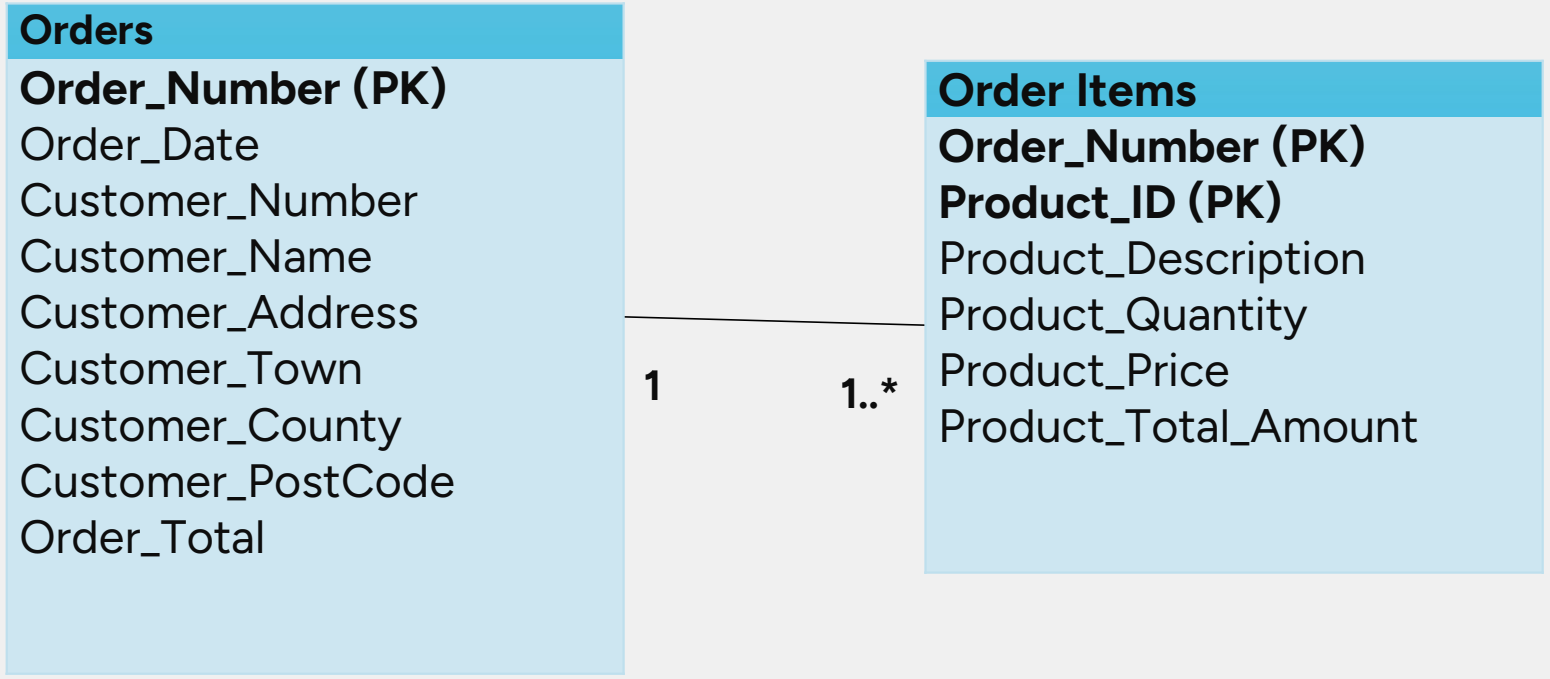


Converting to first normal form (1NF)

Orders
Order_Number (PK)
Order_Date
Customer_Number
Customer_Name
Customer_Address
Customer_Town
Customer_County
Customer_PostCode
Order_Total

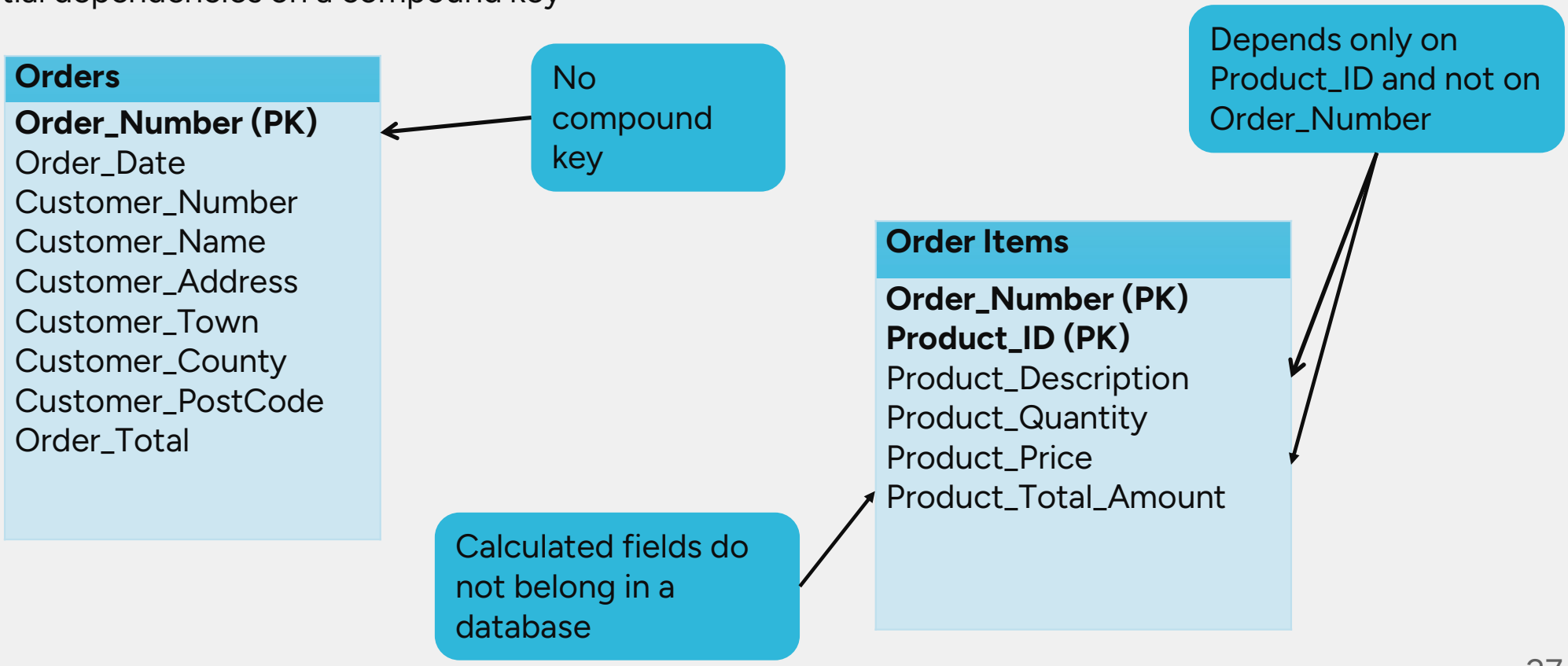
Order Items
Order_Number (PK)
Product_ID (PK)
Product_Description
Product_Quantity
Product_Price
Product_Total_Amount

The Entity Relationship Diagram



Converting to second normal form (2Nf)

No partial dependencies on a compound key



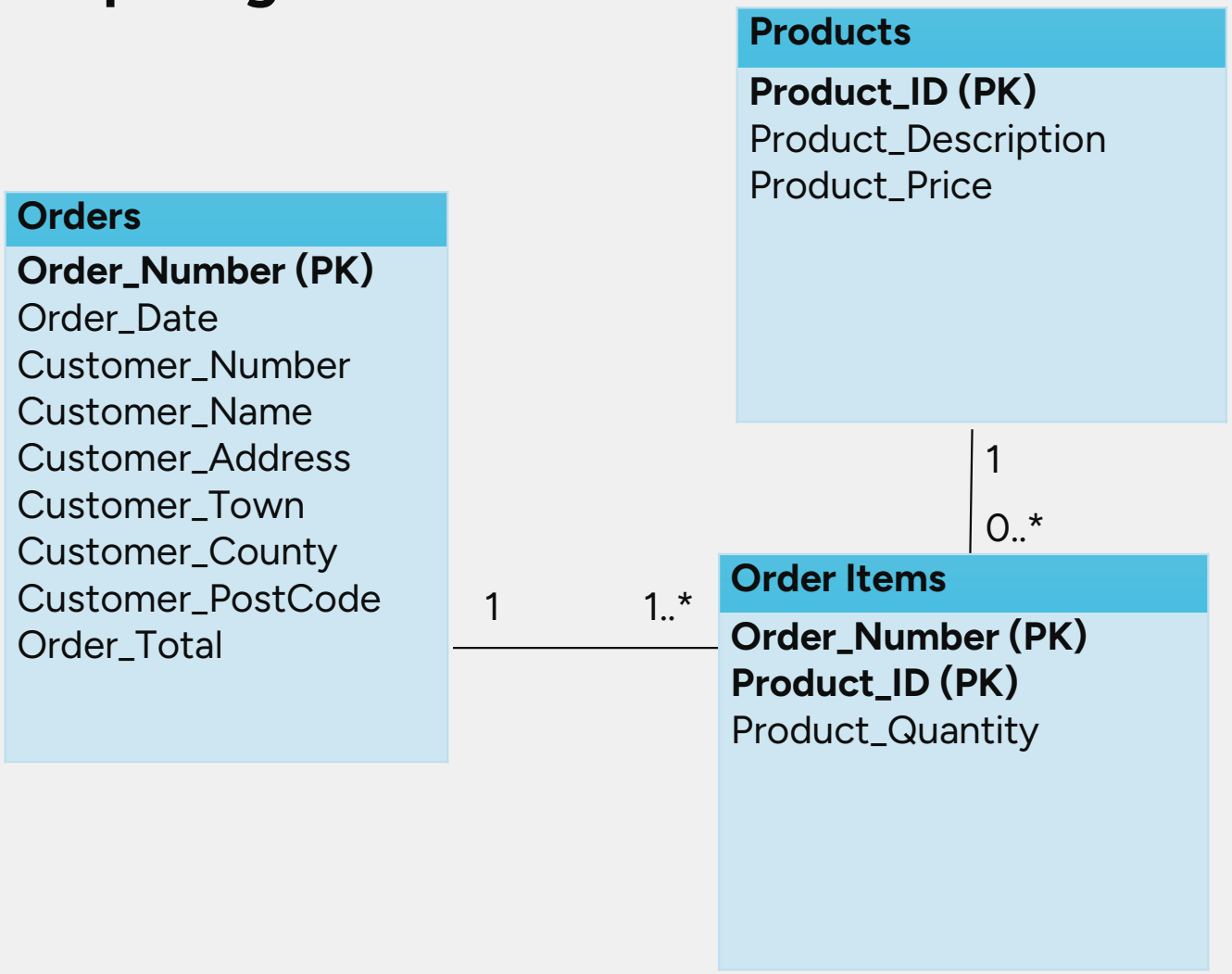
Second normal form (2nf)

Orders
Order_Number (PK)
Order_Date
Customer_Number
Customer_Name
Customer_Address
Customer_Town
Customer_County
Customer_PostCode
Order_Total

Order Items
Order_Number (PK)
Product_ID (PK)
Product_Quantity

Products
Product_ID (PK)
Product_Description
Product_Price

Entity Relationship Diagram



A problem

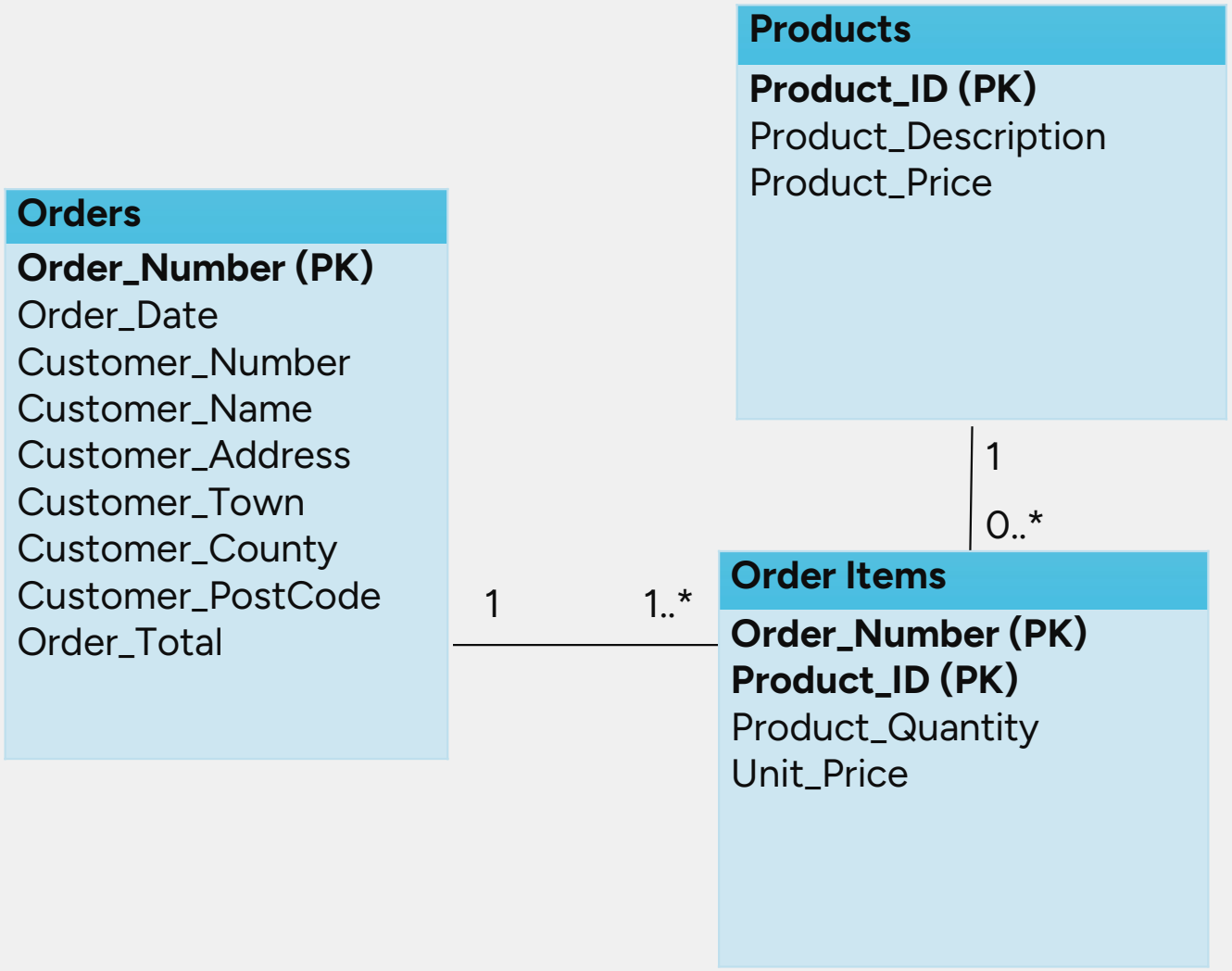
- What happens if the price of a product changes after an order has been placed?
- What should the customer pay?
- The old price or the new?

Customer purchases game for £50.

Product Price increases to £55

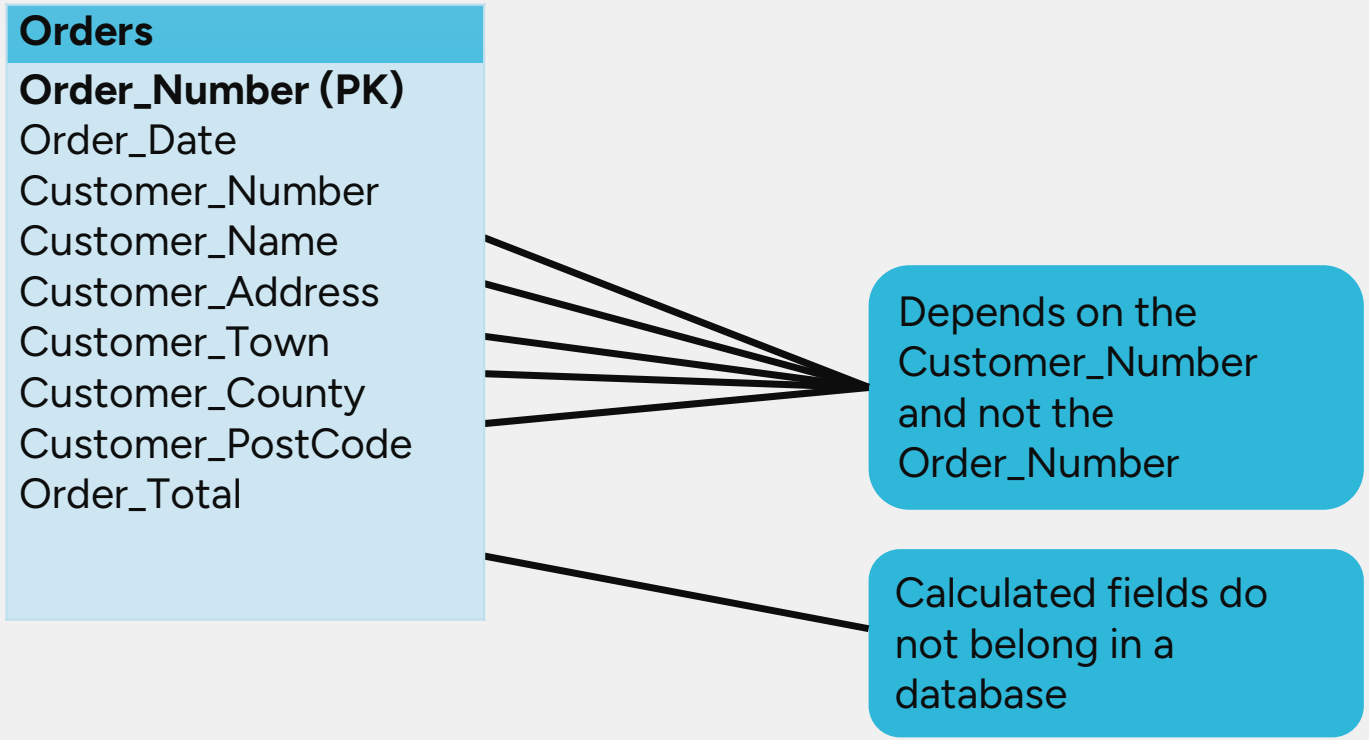
Customer returns game and demands a refund.

A solution


Products
Product_ID (PK)
Product_Description
Product_Price
1
0..*
Orders
Order_Number (PK)
Order_Date
Customer_Number
Customer_Name
Customer_Address
Customer_Town
Customer_County
Customer_PostCode
Order_Total
1
1..*
Order Items
Order_Number (PK)
Product_ID (PK)
Product_Quantity
Unit_Price

Converting to third normal form (3nf)

No non-primary key column should depend on another non-primary key column.



Entity Relationship Diagram