

Databases

(6G4Z0016)

Data Modelling 2: ERDs, Further Details

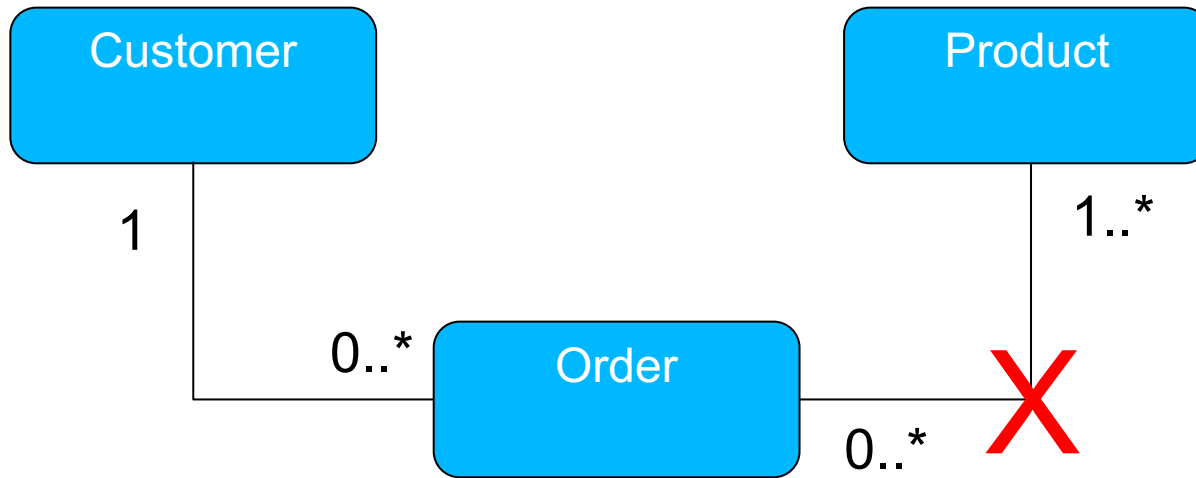
Stephen Gordon

s.gordon@mmu.ac.uk

Entity Relationship Diagrams

- Many to many relationships
 - Why are they a problem?
- Weak entities
 - What exactly is a weak entity?
- Binary and Unary Relationships
- Navigating through tables

Recap: Many to Many Relationships

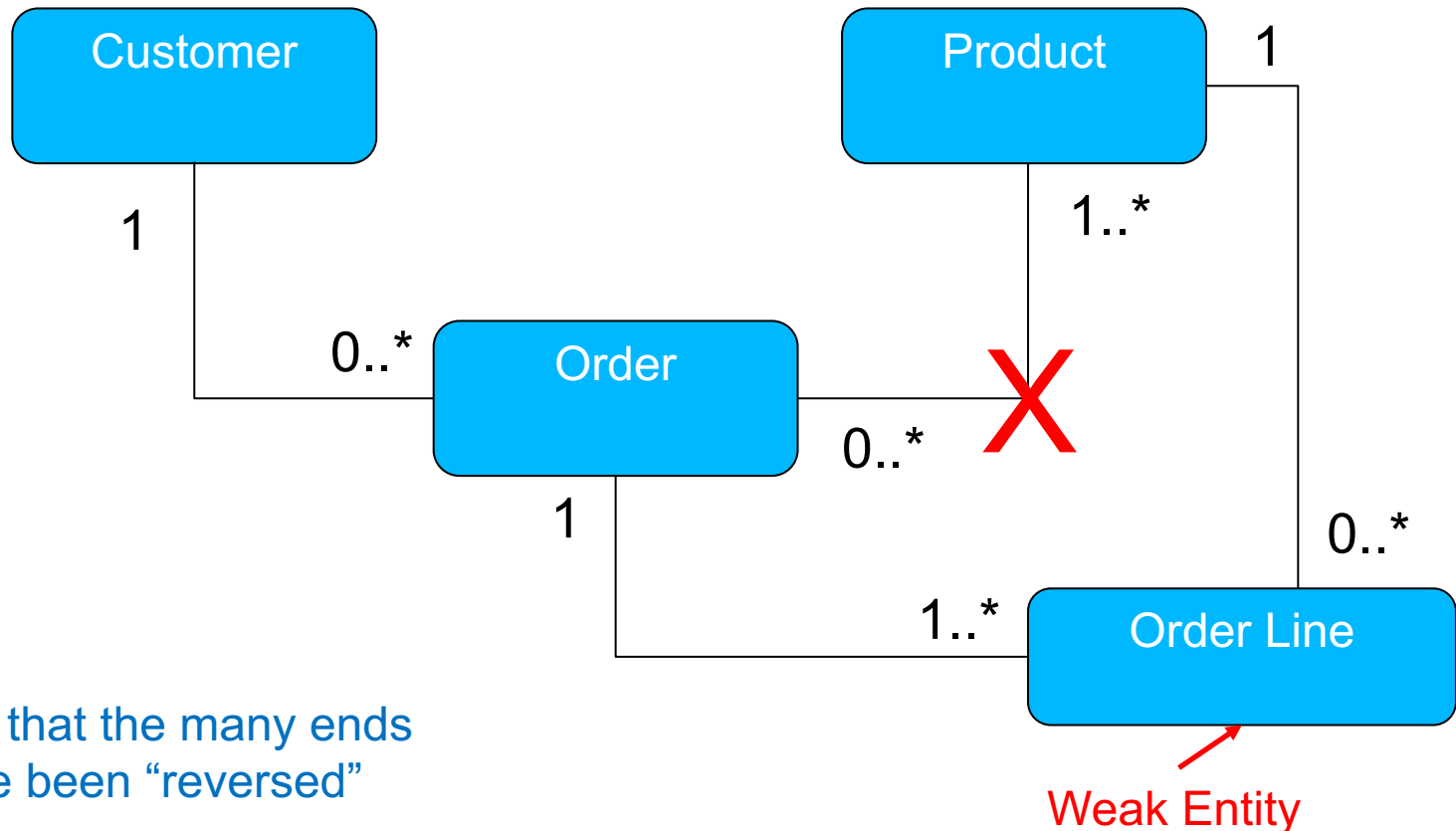


We can't model many to many relationships
in a relational database!

Recap: Weak Entities

- **Many-to-many relationships are not allowed** in relational databases
- So, if they occur, they must be ‘resolved’
- This is done by adding an **extra entity** (a “**weak entity**”) that fits between the two “strong” entities that have the many-to-many relationship
- The relationship can then be broken up into two one-to-many relationships

The Relationship Between Order and Product



Many to Many Issue

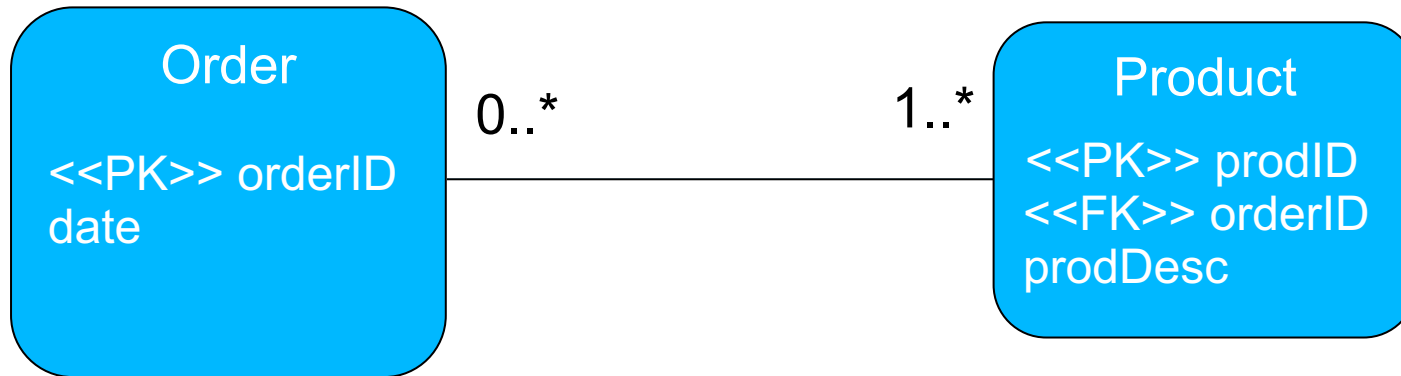


Order ID	Date	Product ID
123	3/2/23	03
124	5/2/23	05
123	3/2/23	06
123	3/2/23	05

If you have a many to many and put the FK on one side of relationship, you will not be able to store all the data you need to.

PK is OrderID so the highlighted records in the order table cannot be stored without violating the PK unique constraint

Many to Many Issue

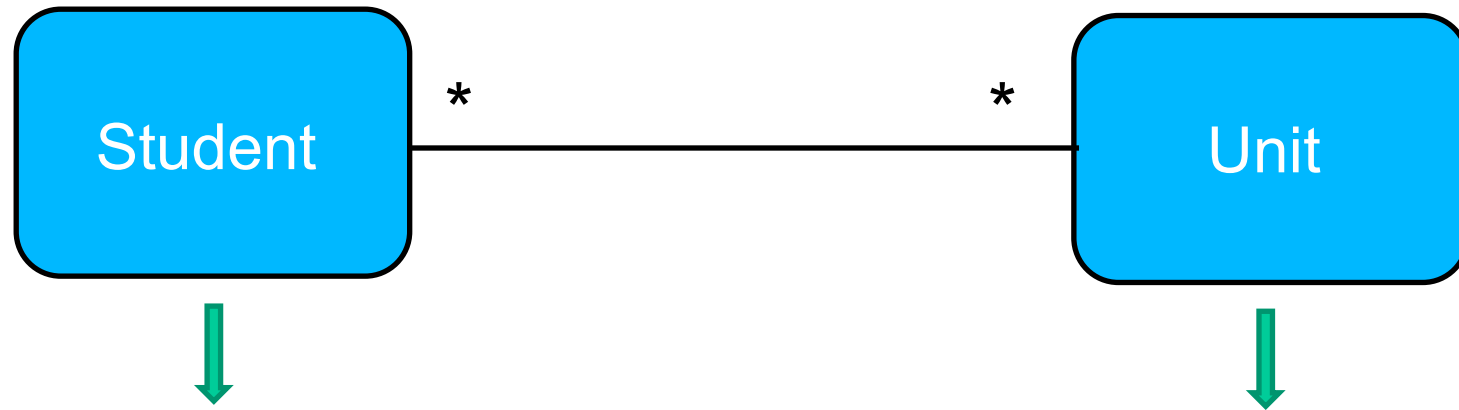


PK is ProductID so the highlighted record in the Product table cannot be stored without violating the PK unique constraint

Let's try putting FK in Product table

Product ID	Prod Desc	Order ID
03	Widget	123
05	Sproket	124
06	Screw	123
05	Sproket	123

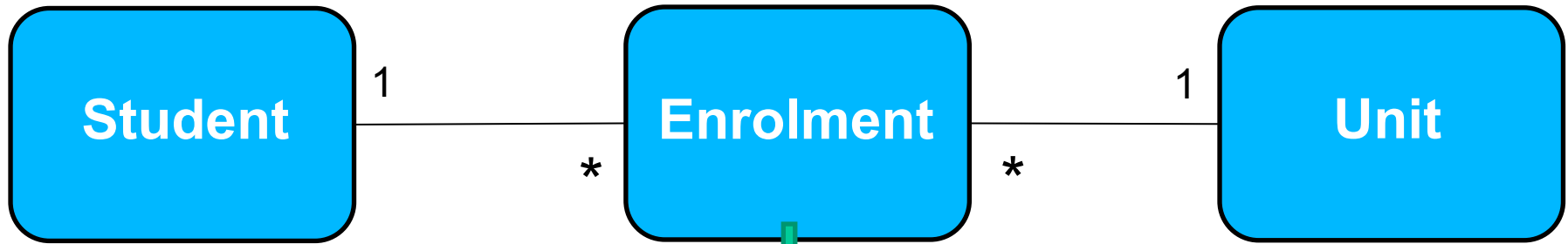
Many-to-Many Relationships



Name	DOB	ID
Fred Smith	2/2/54	12
Joe Ryan	6/1/22	13
Sue Jones	3/2/77	14
Win Yung	2/1/80	15

Name	Code
Modern Pottery	AD1
Programming in VB	CM2
Spanish II	LL1
Systems Analysis	CM3

Many-to-Many Relationships



Name	DOB	ID
Fred Smith	2/2/54	12
Joe Ryan	6/1/22	13
Sue Jones	3/2/77	14
Win Yung	2/1/80	15



ID	Code
12	AD1
13	AD1
14	AD1
12	CM2
13	CM2
14	CM2
12	CM3
14	CM3
14	LL1
15	LL1

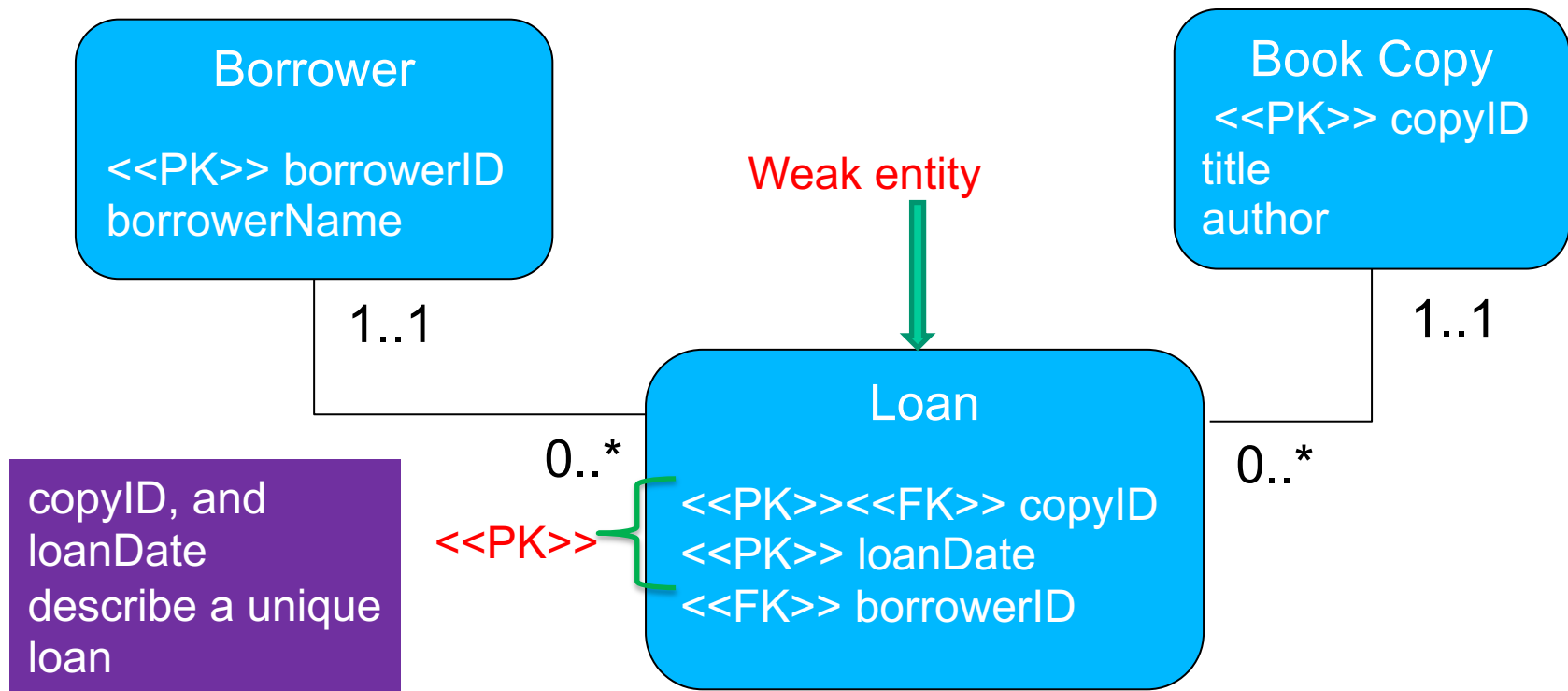


Name	Code
Modern Pottery	AD1
Programming in VB	CM2
Spanish II	LL1
Systems Analysis	CM3

↑ ↑ Composite primary key

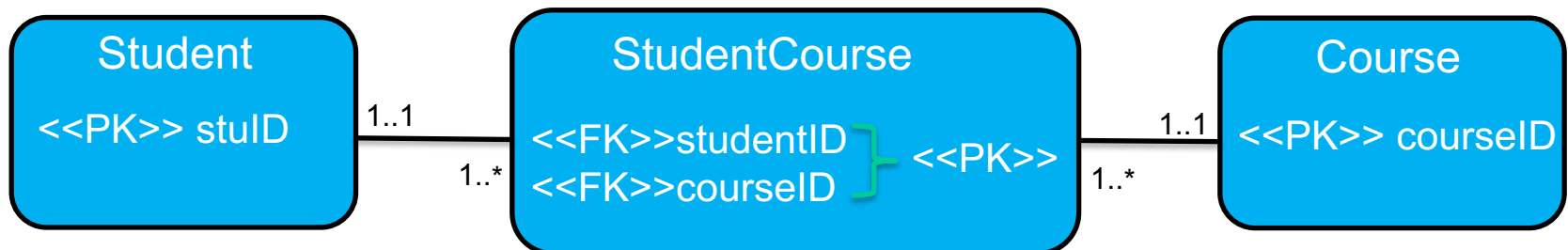
Weak Entity Definition

- A weak entity is an entity that cannot exist without being related to at least one other entity (often two, but sometimes just one)
- In the below example, loan MUST be linked to at least one book copy, and at least one borrower. Loan is therefore a weak entity.

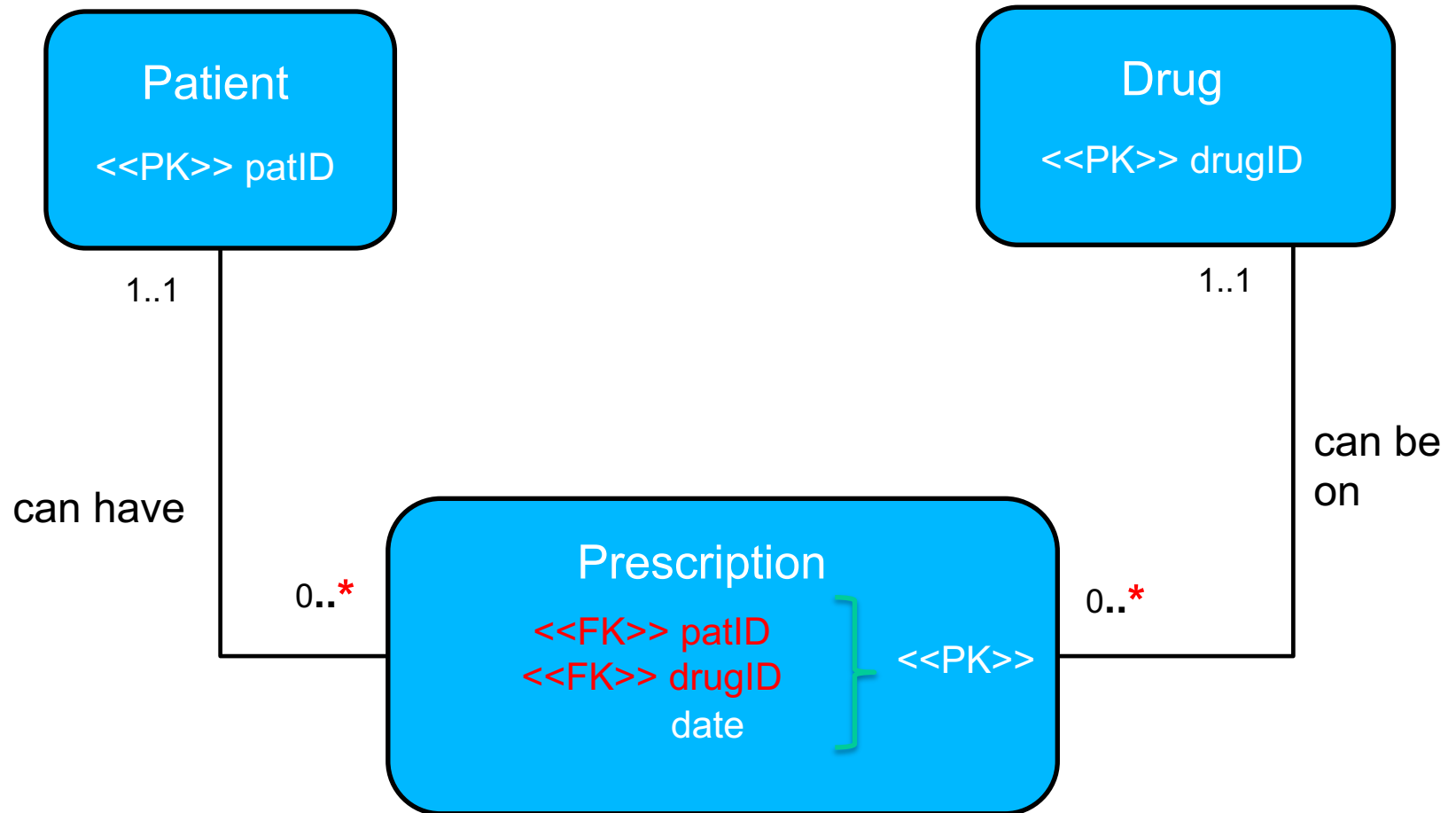


Weak entity

- A **weak entity** depends upon its owner entity(s) and its “natural” primary key is made up of at least one attribute from another table
- If a weak entity has 2 “owner” entities, it will have at least **two foreign keys**
- Often the primary key of a weak entity is formed by taking the primary key of the two “owner” entities upon which its existence depends



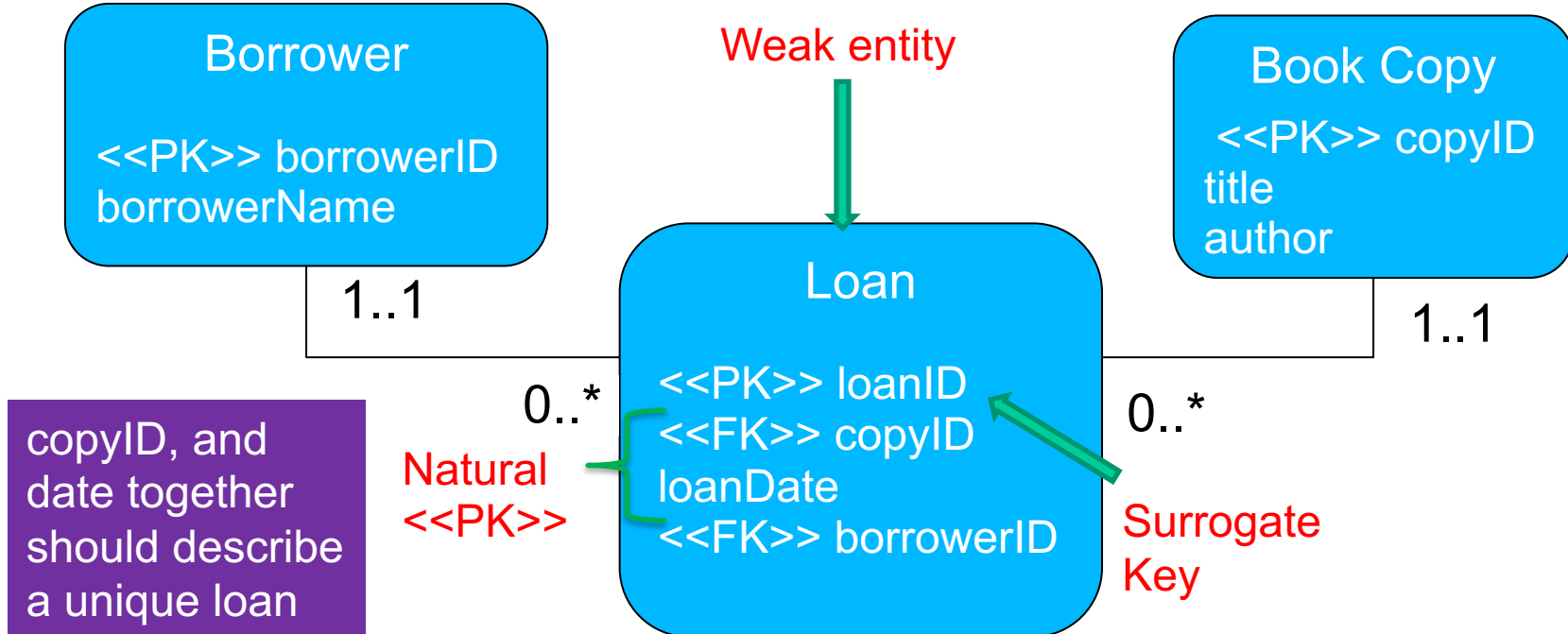
Weak Entity Example



We also need a date to uniquely ID one prescription

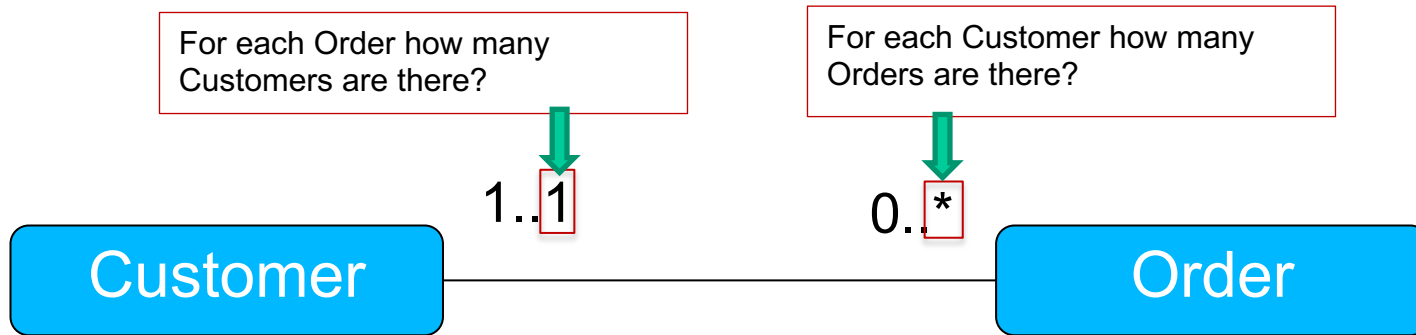
Surrogate and Natural Keys

- A surrogate key is a PK that we use to replace a “natural” PK. It has no business meaning and is generated purely as a unique identifier
- There are several pros and cons of using surrogate PKs and whether it is right use one depends on context. One disadvantage is that you might accidentally allow through duplicates, unless a UNIQUE constraint is set for the natural PK.
- Here, for example, if we use a loanID as a surrogate PK we should also add a UNIQUE constraint for copyID and loanDate together (e.g., UNIQUE (copyID, loanDate)).



Binary Relationships

- Most relationships are between two entities

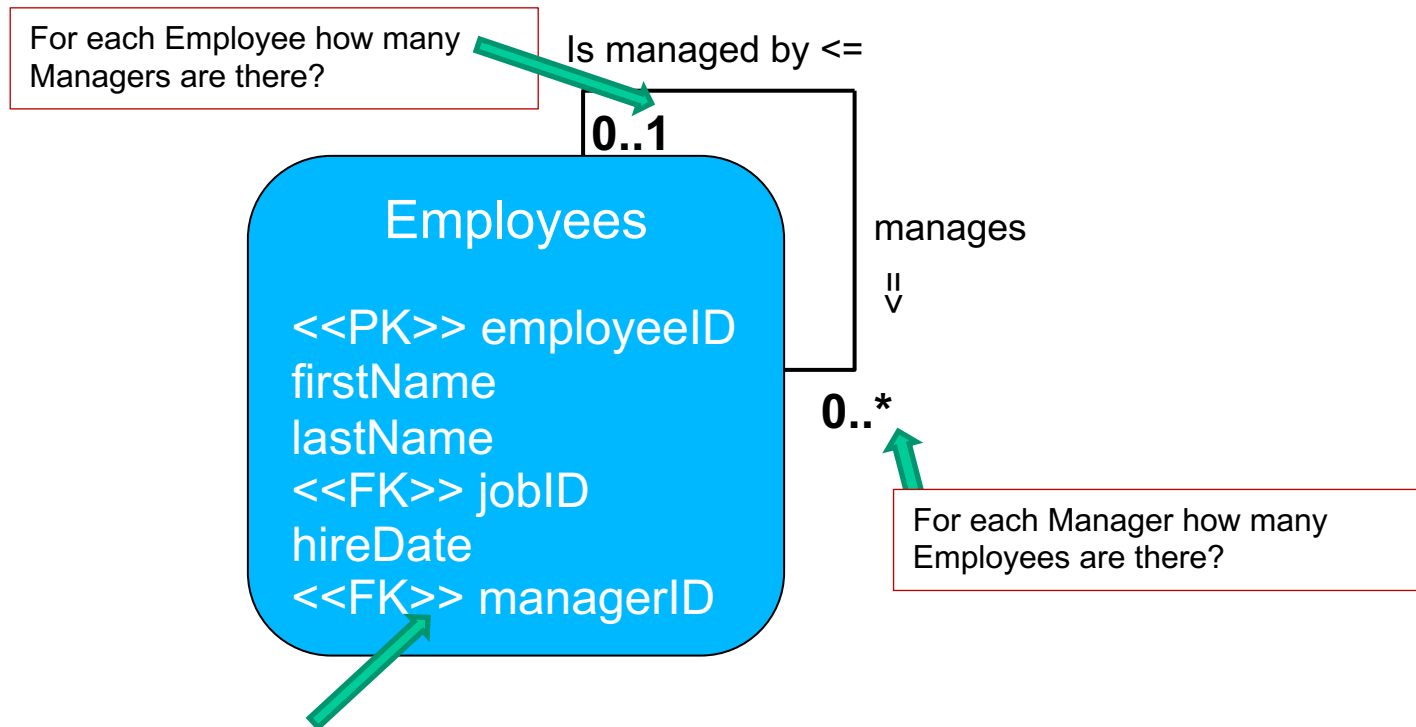


Unary Relationship

- An unary relationship (AKA a '**pig's ear**') is a relationship of an entity with itself
- It is a recursive (self-call) relationship
- For example:
 - In a personnel system with a "employees" entity, we want to record each employee's manager
 - In most cases, the manager is *a/so* an employee with a manager
 - We can use an unary relationship here...

ERD with Unary Relationship

- Here, the FK of manager_ID points to the PK of employee_ID



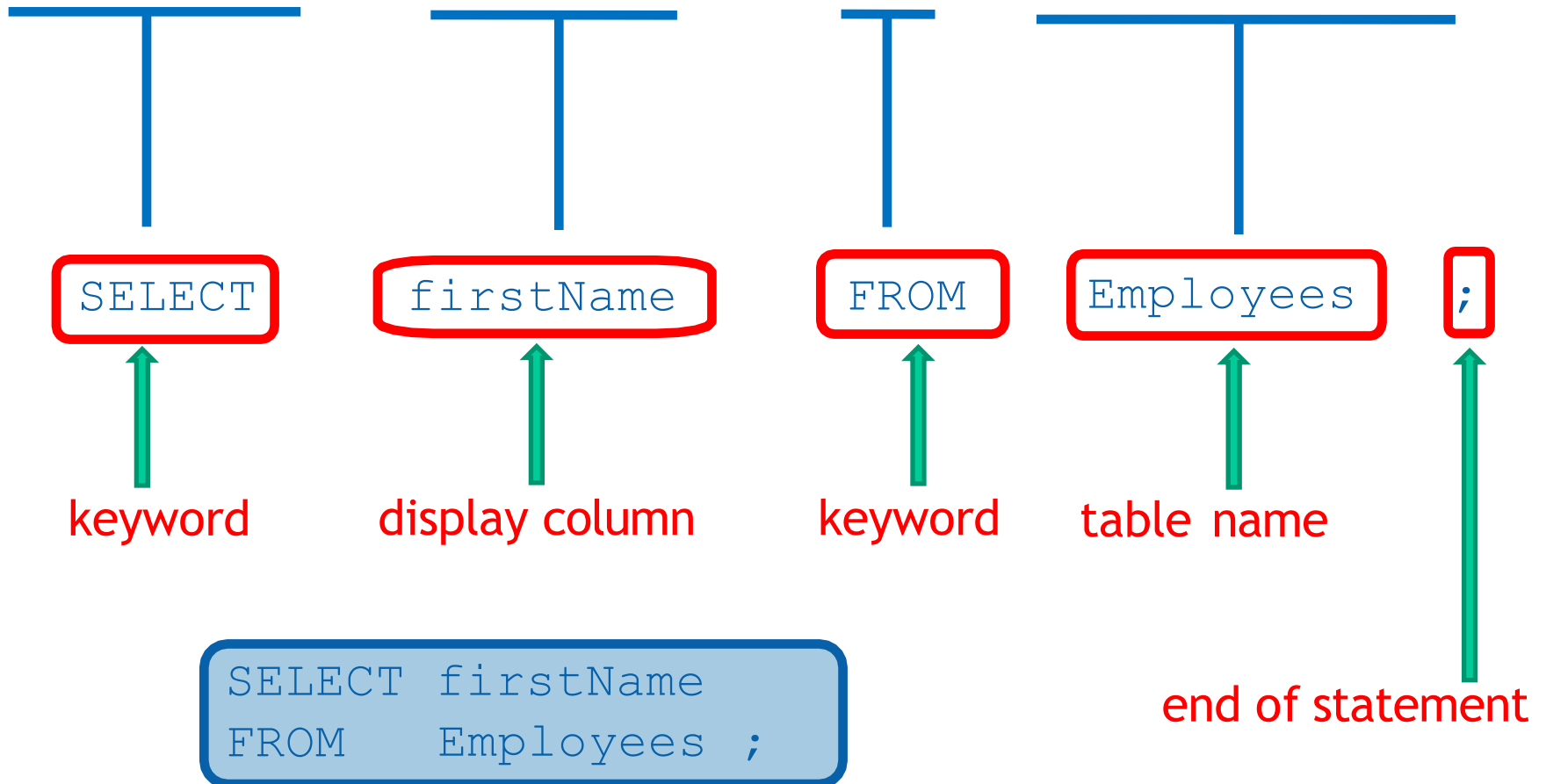
managerID is an alias of employeeID

ERD to Database

- Once we have an ERD, we have designed our data structure
- We then can code it in SQL:
 - We produce our (empty) database tables using CREATE statements
 - We add data using INSERT statements
 - We can then use SELECT statements, or queries, to pull data out of databases
 - This will be covered later in the unit

SELECT Statement Structure

Retrieve data from first name column in the Employees table



Navigating Relationships Using Joins

- If we need data from more than one table, we can use a JOIN statement in SQL
- The SQL for JOINS will be covered, in detail, later in the unit
- We are going to look at JOINS in SQL briefly now, but here...
- Rather than coding, the focus is on understanding how databases are navigated using queries
- Once you understand this, we will be able to assess your understanding of complex data structures by asking questions like:
 - “If you know a copyID, what tables do you need to read to find out the title of the book?”

JOIN clause

- The JOIN clause is used to combine rows from two tables, based on a common place (columns or attributes)
- Why “common place” – could have:
 - SAME column name OR
 - SAME column values (e.g., primary keys = foreign keys)
- In other words, columns might have different names (aliases) in different tables but hold data that overlaps, so that values can still be matched

INNER JOIN

We would like to know order_id, customer name and order date for all orders. But that information comes from two tables.

```
SELECT OrderID, CustomerName, OrderDate  
FROM
```



columns from both tables

INNER JOIN

ORDERS

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CUSTOMERS

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

INNER JOIN

ORDERS

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CUSTOMERS

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

INNER JOIN

ORDERS

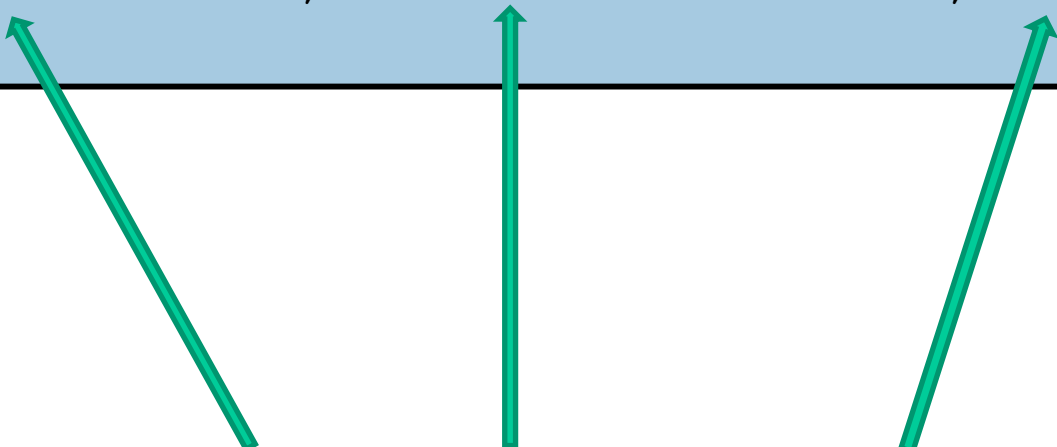
OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CUSTOMERS

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM
```

The diagram consists of three teal arrows pointing upwards from the text below to the table names in the SQL query above. The first arrow points from 'the tables' to 'Orders'. The second arrow points from 'the columns' to 'Customers'. The third arrow points from 'are coming from' to 'Orders'.

the tables the columns are coming from

INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders
```

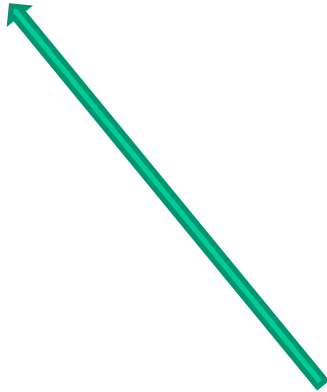


Table 1

INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
JOIN
```

INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
JOIN Customers
```

Table 2



INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
JOIN Customers  
ON
```

INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
JOIN Customers  
ON Orders.CustomerID = Customers.CustomerID
```



common field

INNER JOIN

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
JOIN Customers
ON Orders.CustomerID = Customers.CustomerID;
```

ORDERS

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CUSTOMERS

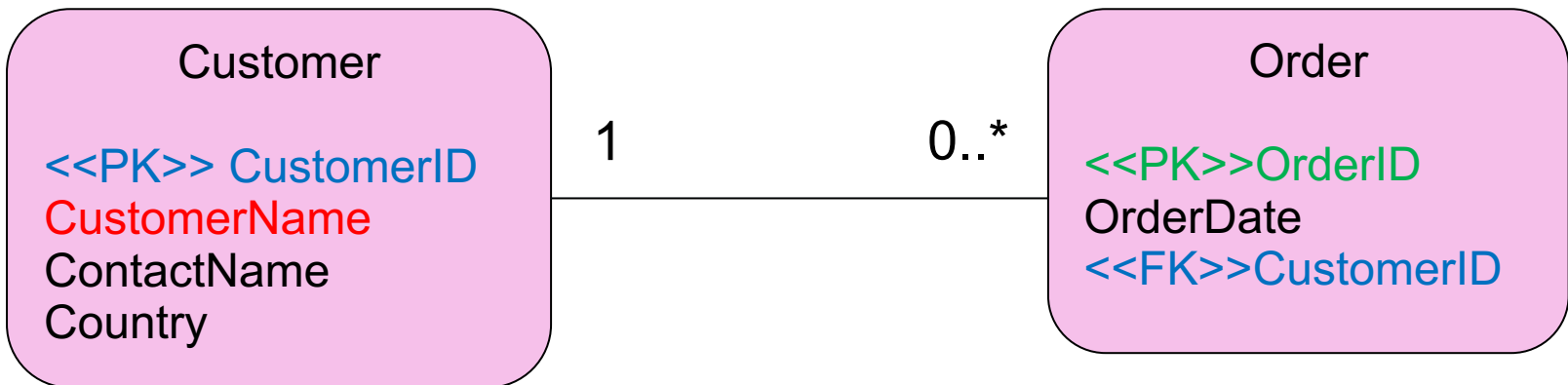
CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

OrderID	CustomerName	OrderDate
10248	Wilman Kala	1996-07-04
10249	Tradição Hipermercados	1996-07-05
10250	Hanari Carnes	1996-07-08
10251	Victuailles en stock	1996-07-08
10252	Suprêmes délices	1996-07-09
10253	Hanari Carnes	1996-07-10
10254	Chop-suey Chinese	1996-07-11
10255	Richter Supermarkt	1996-07-12

Navigating through a DB using JOINS

- To sum up: If we need to get order date with a customer name we get there by JOINing with the Customer table, through a common place.

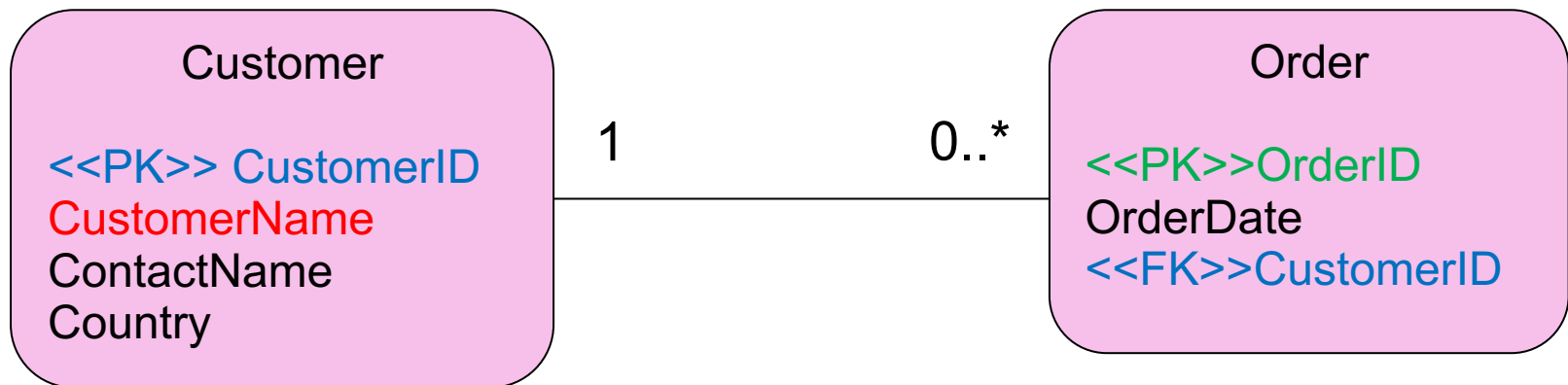
```
SELECT OrderID, CustomerName  
FROM Order JOIN Customer  
ON Order.CustomerID = Customer.CustomerID;
```



Navigating DB tables using JOINS

The syntax is not important right now. The point is that we navigate through a data structure using JOINS between PKs and FKs.

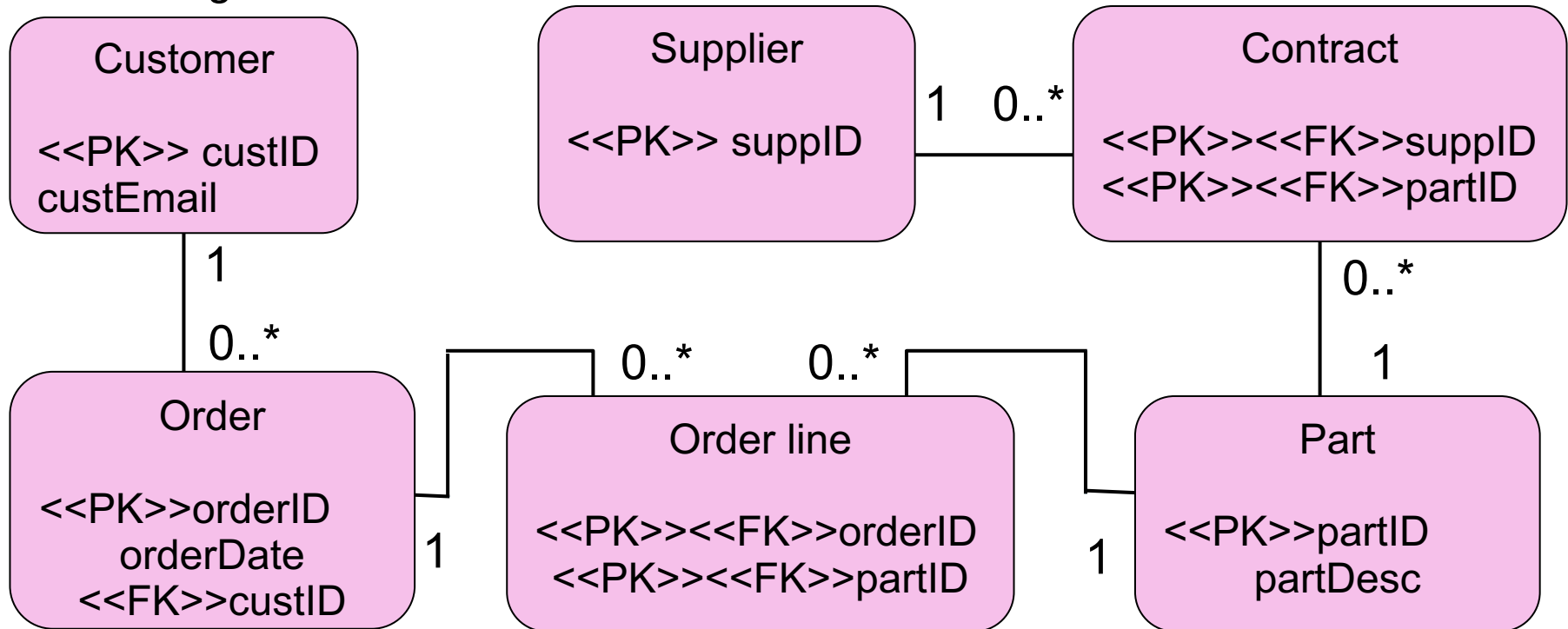
So, here: we start with the **green data item** (orderId), go through the **blue data items** (PKs and FKs) to arrive at the **red data items** (the customer's name).



Navigating a Complex DB

Looking at Spareparts from last week's lab, let's say we need to track down **customer email addresses** for all customers who has received goods that we sourced from a particular supplier since July 23.

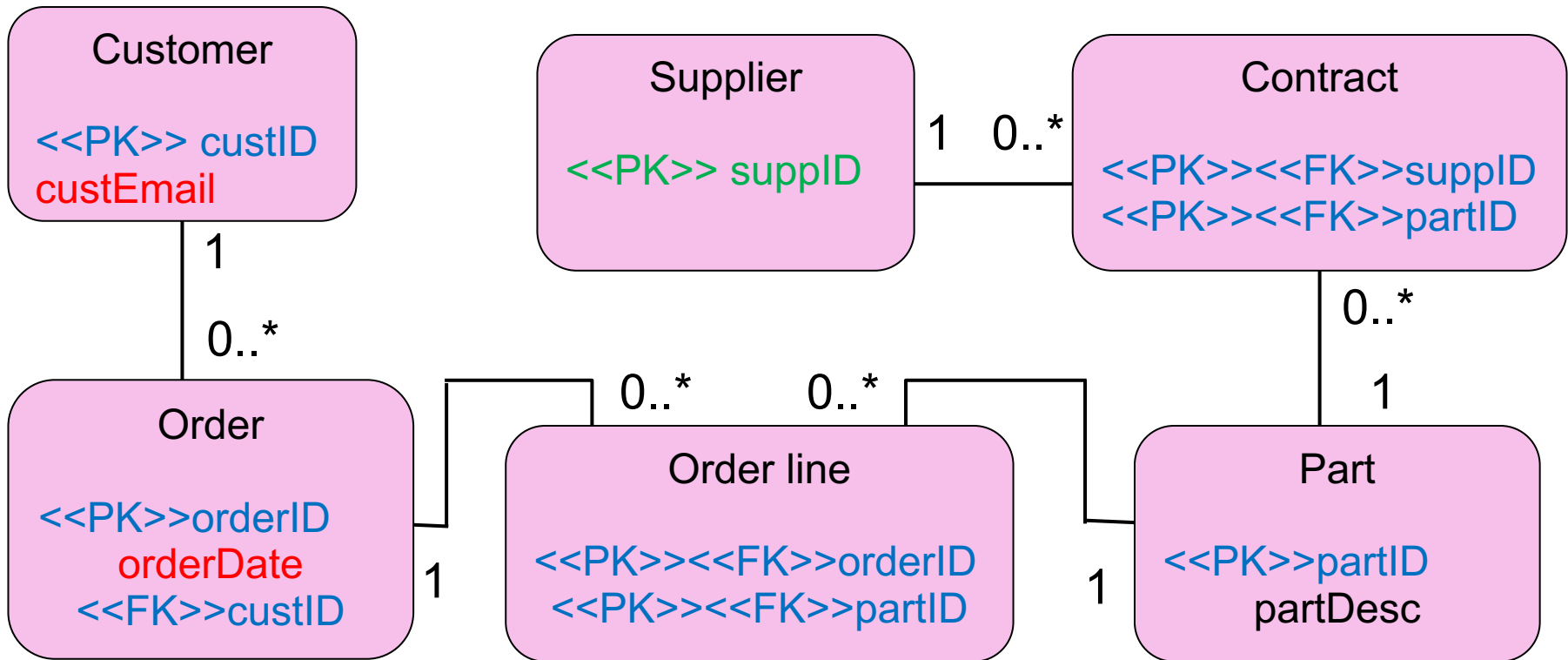
We have a **supplier ID**. What tables would we need to read in the below DB Design?



Navigating a Complex DB

Answer: All of them!!

Again: start with **green** move through the **blue data items** and end up at the **red data items**



Navigating using JOINS

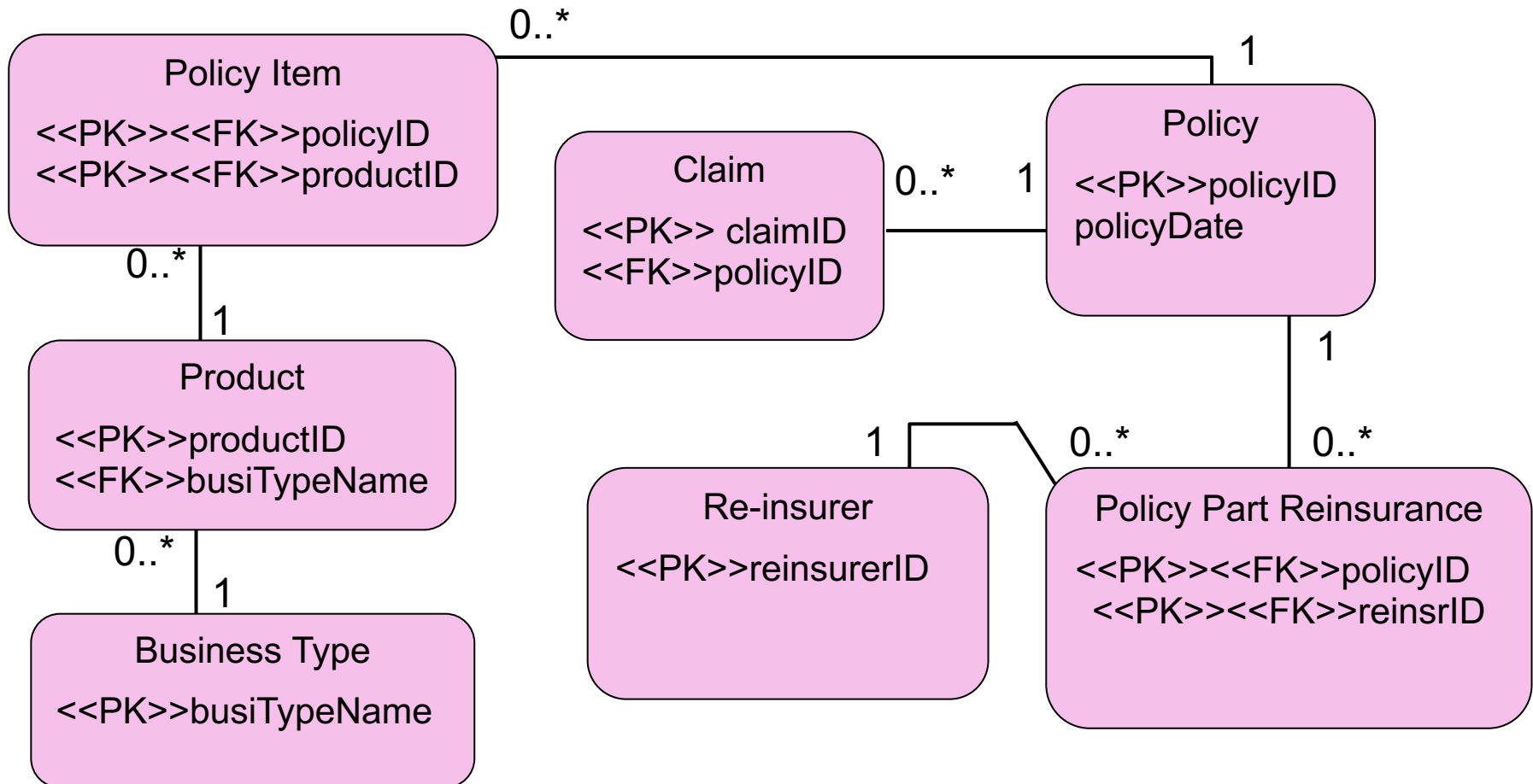
This would involve a complicated query, which we would be unlikely to need to run very often. Example syntax is given below. But... the syntax is not the point.

The point is that we will be assessing your understanding of complex ERDs by asking questions like those above.

```
SELECT custEmail
FROM Supplier
JOIN Contract USING suppID
JOIN Part USING partID
JOIN OrderLine USING partID
JOIN Order USING orderID
JOIN Customer USING custID
WHERE suppID = 004 & orderDate > '01-JUL-2023'
```

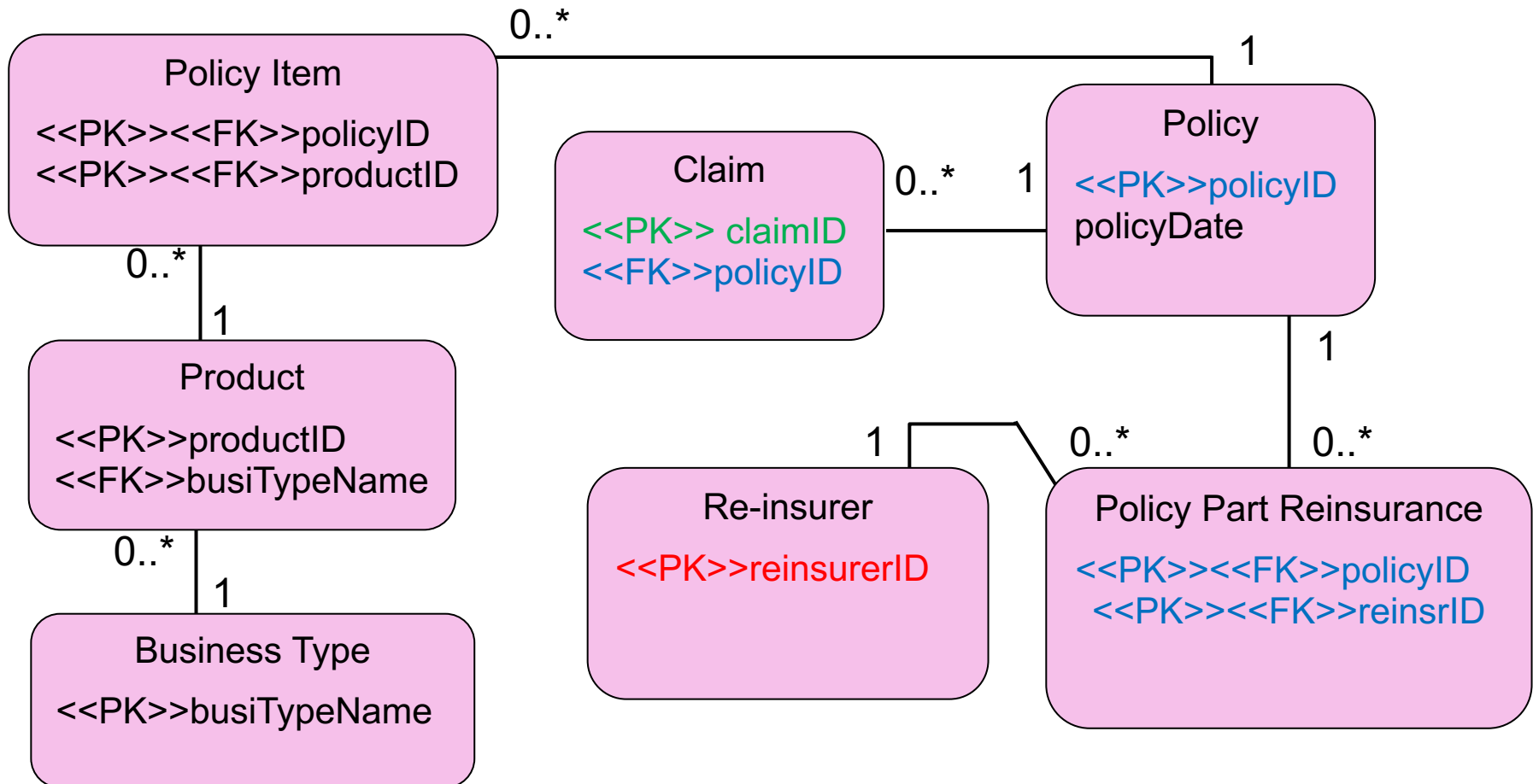
Further Navigation Question 1

Looking at the below insurance ERD, let's say we have a claimID and we need to find out the re-insurer(s) for that claim. What tables would we need to read in the below DB Design?



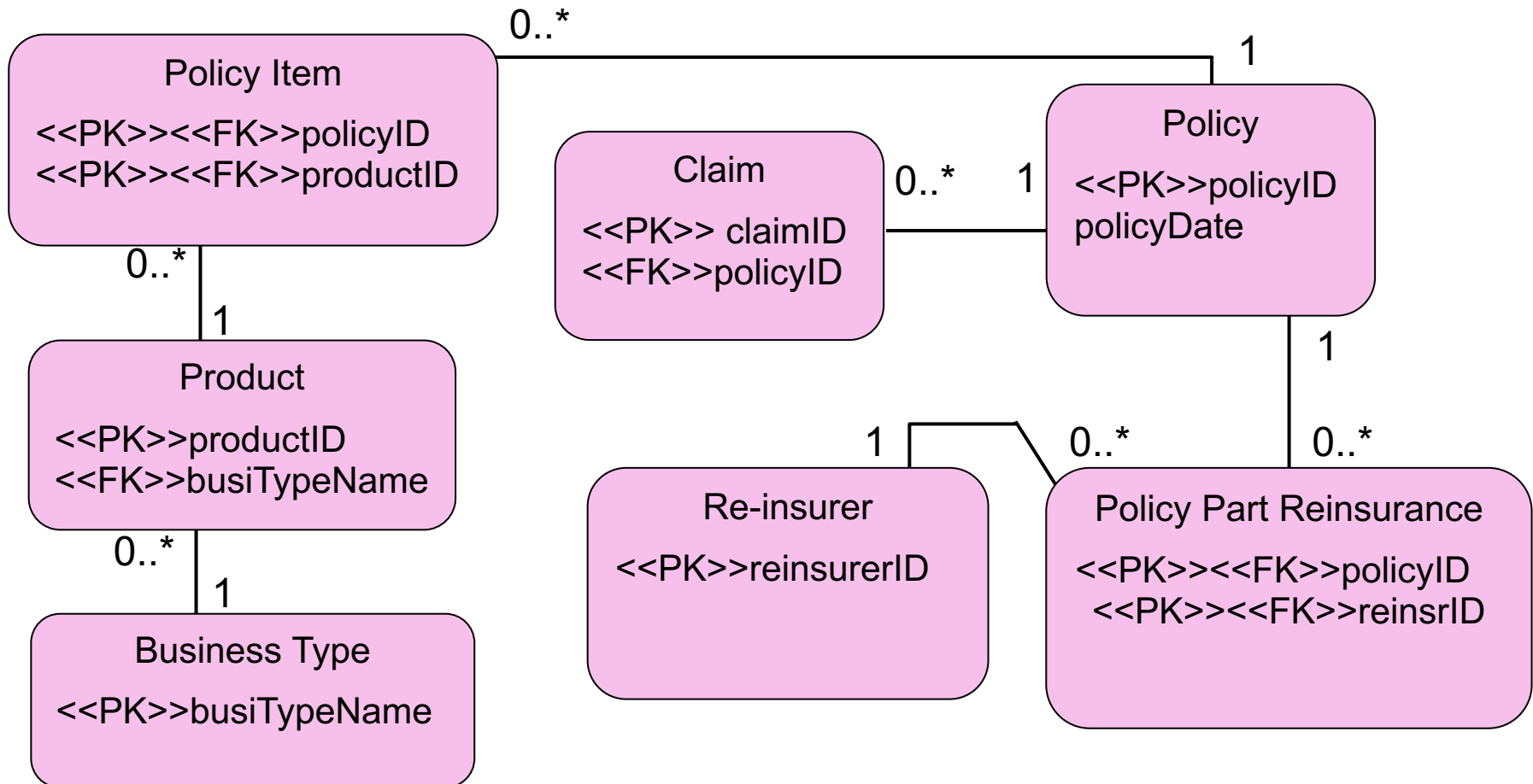
Further Navigation Solution 1

Answer: Claim, Policy, Policy Part Reinsurance and Re-insurer



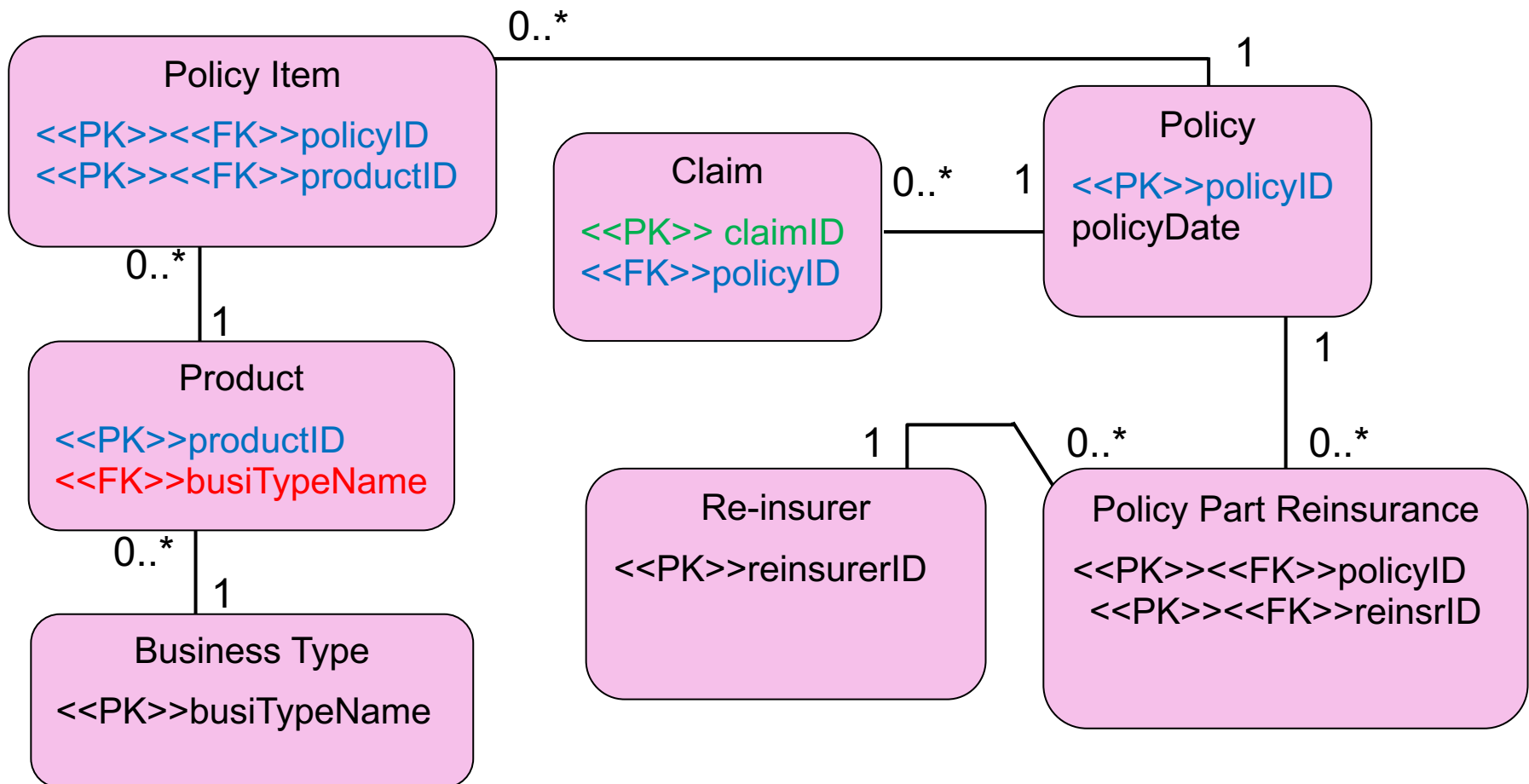
Further Navigation Question 2

Looking at the below insurance ERD, let's say we have a claimID and we need to find out all the business type(s) for that claim. What tables would we need to read in the below DB?



Further Navigation Solution 2

Answer: Claim, Policy, Policy Item, Product



Summary

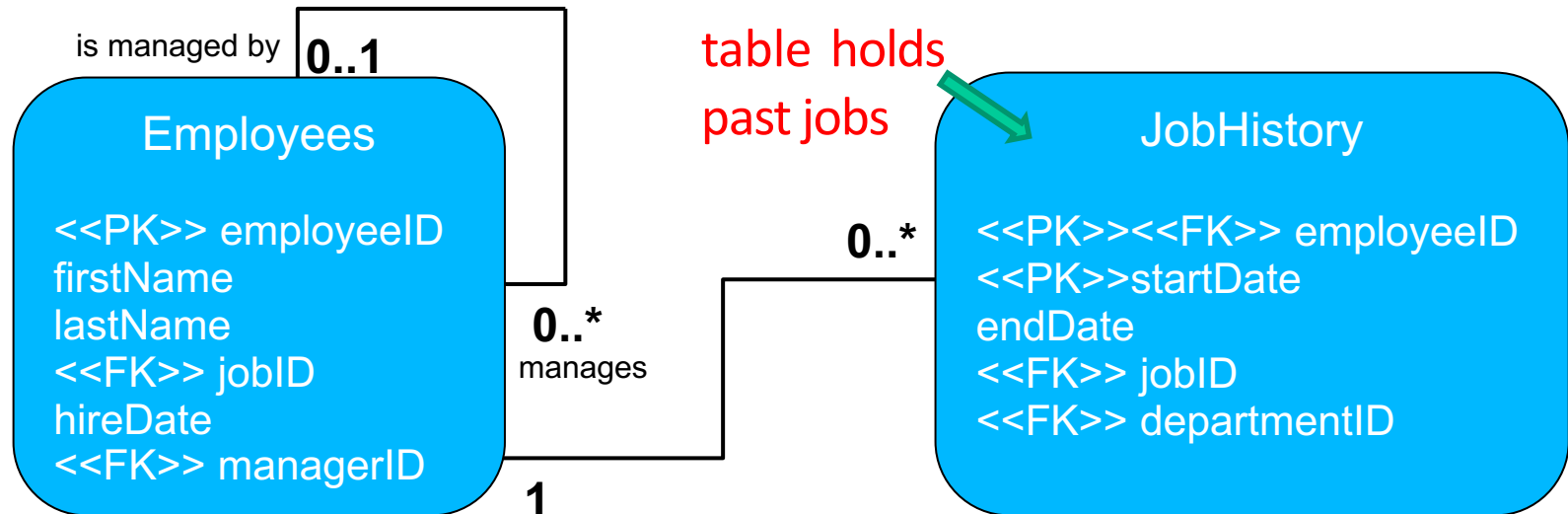
- Many to many relationships
 - Why are they a problem?
- Weak entities
 - What exactly is a weak entity?
- Binary and Unary Relationships
- Navigating through tables

Extra Material

JOIN clause – common “place”

- In the below, manager_id is an alias for employee number, so there will be matches between employee_id and manager_id.
- So, e.g., the manager_ID FK in employees can be joined to employee_ID in job_history to find the start dates of all managers' past jobs in the organisation.

```
SELECT DISTINCT(manager_id), job_history.start_date
FROM employees JOIN job_history
ON employees.manager_id = job_history.employee_id;
```



References

Whiteley, D. (2013), 'Introduction to Information Systems', Palgrave Macmillan

Coronel, C., Blewett, C., Crockett, K., & Morris, S. (2020). *Database principles : fundamentals of design, implementation, and management* (Third edition). Cengage.