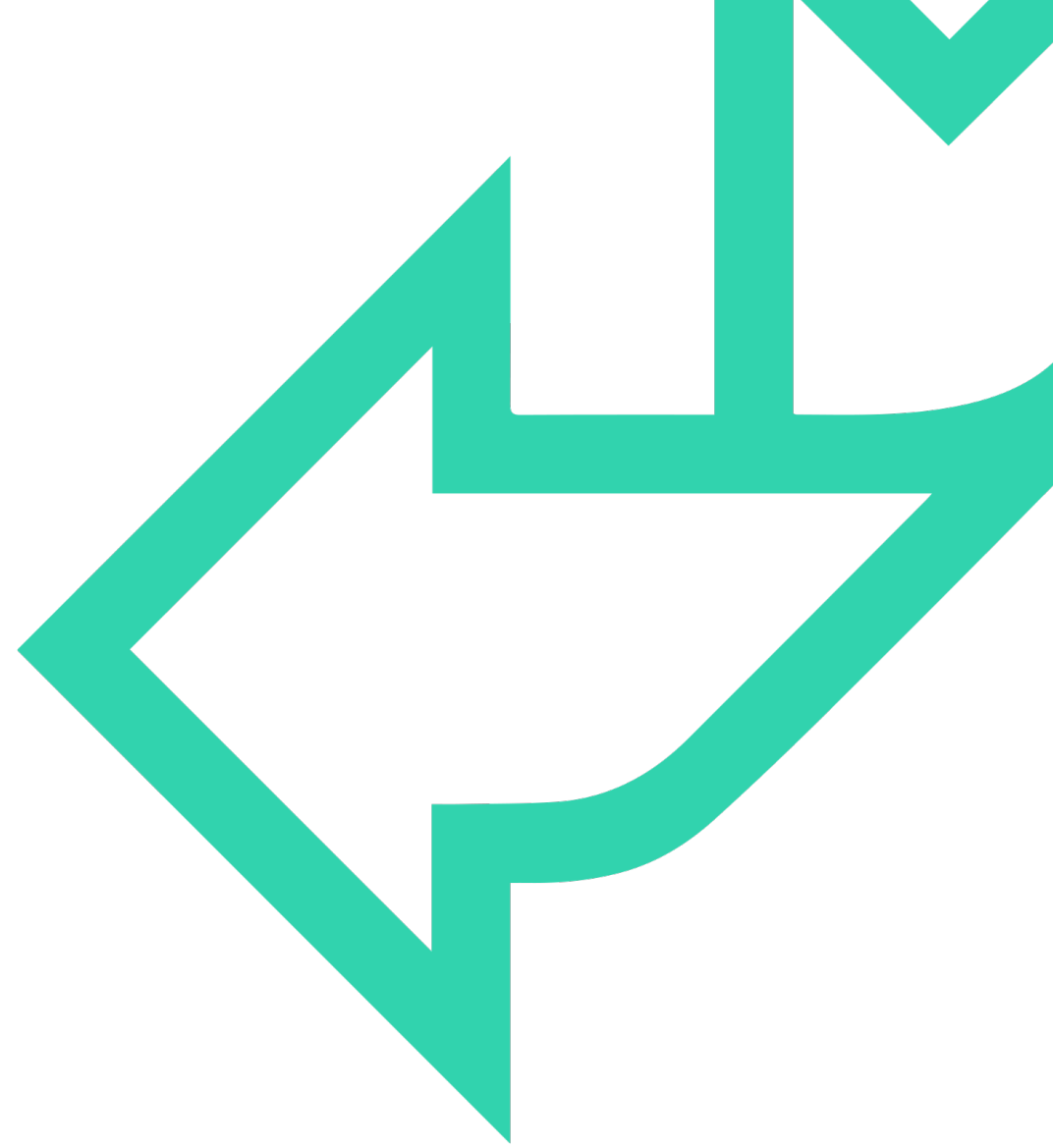




# Data definition

## Module 3: Databases





# Databases

## Module 3: Contents

- Designing a schema.
- Data Definition Language.
- Creating tables.
- Entity Relationship Diagrams.
- Exploring entity relationships.





# Objectives

## Data definition

**Describe what the schema of a database is.**

- How are schema implemented?

**Explore the syntax used to create a table through Data Definition Language.**

- How can we create the tables we need for our database?

**Create an ERD based on database tables and keys.**

- What will the structure of our database look like?

**Create the GAME database, implementing keys and constraints.**

- How can we begin to build out our database?



# Designing a schema

Databases

Module 3: Data Definition

# QA Why do we need a schema?

**Every database itself is built from a well-structured schema:**

- The database schema describes the structure of the database.
- Tables, fields, and their relationships are all part of the schema.

**Before we start entering and using the data, we need to define the schema:**

- To do this, we can start using **Structured Query Language (SQL)** for the first time.

**SQL allows us to create our databases and tables, as well as populate it with data and ask it stuff:**

- However, for now, we use SQL's specific schematic tool for creation, which is **Data Definition Language**.

# QA DDL: Data Definition Language

**Data Definition Language – or DDL – is used to create and manipulate the database schema:**

- There are three key words used for this: **CREATE**, **ALTER** and **DROP**.

**The first way we start using DDL is by creating our entire database:**

```
CREATE DATABASE IF NOT EXISTS gamedb;
```

# QA DDL: Creating tables

**Of course, all our tables can also be created through DDL:**

- The syntax for this breaks down into two steps: first create the table, then the fields.

```
CREATE TABLE table_name (  
    column_name1 data_type(size) constraint_name,  
    column_name2 data_type(size) constraint_name,  
    PRIMARY KEY (field_name),  
    FOREIGN KEY (other_field) REFERENCES table_name(primary_key)  
);
```

**auto\_increment**

**Strings (written words) +1**

**Numerics – can have mathematics applied to it.**



We start with the CREATE TABLE statement followed by the name of the table you wish to make.

We then start listing the fields we want in the table. This needs the name of the field (column), the data type (int, varchar, Boolean, etc.), and any data constraints (more on this later).

```
CREATE TABLE table_name (  
    column name1 data type(size) constraint name,  
    column_name2 data_type(size) constraint_name,  
    PRIMARY KEY (field_name),  
    FOREIGN KEY (other_field) REFERENCES table_name(primary_key)  
);
```

We can also set a primary key, which acts as a unique identifier for a table record.

We can also set a foreign key that links a field from this table to the primary key field of another table.



# QA Table constraints

**We can add constraints to our data to further improve data validation.**

Data types is a simple method of validation, but we can make this more complex.

**Constraints include:**

- Unique.
- Not null.
- Primary and foreign keys.
- Default.

# QA NOT NULL

**NOT NULL** is a constraint that ensures a column cannot have a **NULL** value:

- I.e., the absence of any data
- If you put one space into a field that is not null, that is technically compliant.

```
CREATE TABLE mytable (  
    name varchar(100) NOT NULL,  
);
```

# QA Unique

**The unique constraint ensures that each column must have a unique value:**

- For example, a rank in a list – no one record can share the same ranking.

```
CREATE TABLE mytable (  
    name varchar(100) NOT NULL,  
    rank int NOT NULL,  
    UNIQUE(rank)  
);
```

# QA Default

## We can also have a default value for columns:

- If no value is specified during data entry, this is the value that will be used.

```
CREATE TABLE mytable (  
    name varchar(100) NOT NULL,  
    rank int NOT NULL,  
    status varchar(10) DEFAULT 'unranked',  
    UNIQUE(rank)  
);
```

# QA Primary keys

**Primary keys are a combination of NOT NULL and UNIQUE:**

- Must have a unique identity for identifying the row quickly and easily.
- A table can only have ONE primary key, and most tables should have one.
- A primary key can be made up of two fields, known as a composite key.

```
CREATE TABLE mytable (  
    person_ID int NOT NULL,  
    name varchar(100) NOT NULL,  
    rank int NOT NULL,  
    status varchar(10) DEFAULT 'unranked',  
    UNIQUE(rank),  
    PRIMARY KEY(person_ID)  
);
```

# QA Foreign keys

**Foreign keys ensure data for this column matches data in another:**

- A foreign key in a table will point to the primary key in another table.
- It prevents invalid data because it must match the data in the table it's pointing to and prevents destroying links between tables.

```
CREATE TABLE mytable (  
    person_ID int NOT NULL,  
    name varchar(100) NOT NULL,  
    rank int NOT NULL,  
    status varchar(10) DEFAULT 'unranked',  
    UNIQUE(rank),  
    PRIMARY KEY(person_ID)  
);
```

```
CREATE TABLE mysecondtable (  
    account_ID int NOT NULL,  
    fk_person_ID int NOT NULL,  
    account_type varchar(100) NOT NULL,  
    FOREIGN KEY(fk_person_ID) REFERENCES mytable(person_ID)  
);
```

# QA Creating tables for our GAME database



## Outcome:

- Create two tables in the **GAME database** based on the tables shown below.
- Ensure the fields for a primary key.



## Steps:

**10 minutes, solo**

- Use DDL to make these two tables, based on what you know so far:

**orders**

ORDER ID	CUSTOMER ID	PRODUCT ID	PLACED	TOTAL
1	1	4	2019-08-06	45.99
2	2	3	2019-08-14	37.99

**customers**

CUSTOMER ID	NAME	ADDRESS	EMAIL	PASSWORD
1	SIMON	256 BYTE STREET	SI@MAIL.CO.UK	*****
2	MARKUS	47 RED TIE ROAD	MARKUS47@POST.COM	*****
3	EMMA	63 NUMBER LANE	EM@LETTER.BOX	*****



## customers

CUSTOMER ID	NAME	ADDRESS	EMAIL	PASSWORD
1	SIMON	256 BYTE STREET	SI@MAIL.CO.UK	*****
2	MARKUS	47 RED TIE ROAD	MARKUS47@POST.COM	*****
3	EMMA	63 NUMBER LANE	EM@LETTER.BOX	*****

primary key 🔑

```
CREATE TABLE customers (  
  customer_id int NOT NULL AUTO_INCREMENT,  
  name varchar(100) NOT NULL,  
  address varchar(100) NOT NULL,  
  email varchar(100) NOT NULL,  
  password varchar(50) NOT NULL,  
  PRIMARY KEY (customer_id)  
);
```

## orders

ORDER ID	CUSTOMER ID	PRODUCT ID	PLACED	TOTAL
1	1	4	2019-08-06	45.99
2	2	3	2019-08-14	37.99

foreign key 🔑

```
CREATE TABLE orders (  
  order_id int NOT NULL AUTO_INCREMENT,  
  customer_id int NOT NULL,  
  placed date,  
  total dec(7,2),  
  PRIMARY KEY (order_id),  
  FOREIGN KEY (customer_id) REFERENCES customers (customer_id)  
);
```





# Entity Relationship Diagrams

Databases

Module 3: Data Definition



# QA Designing a database: ERDs

**Before we start creating a database, we should first model it and test it against any requirements:**

- For this, we'll use an **Entity Relationship Diagram (ERD)**.

**We will need to factor in several considerations when modelling our database:**

- What sort of data will it store?
- Who will access and using it?
- What will the data we store be used for?

# QA ERDs

**ERDs are used to show the design of a database scheme with names, columns, and relationships:**





- Think in terms of the end user. Your user stories should logically include the relevant data.
- Don't include extraneous information.

**ERDs relate information through entities:**

- **Entity type:** A class of objects which share characteristics, e.g. customers, orders.
- **Entity:** An instance of the entity type, e.g. an object, person, event, abstraction.
- The entity type **customers** might have entities **Jeff, Geoff, Djeph**.

## We can give further context to a relationship between entities:

- We use **cardinality** to discuss the number of instances for a relationship between records.

SYMBOL	MEANING
	ONLY 1
	0 OR 1
	1-TO-MANY
	0-TO-MANY

# QA Creating an ERD

**Broadly, you should follow these steps when designing an ERD:**

1. Identify the Entity.
2. Identify the Attributes.
3. Identify the Primary Keys.
4. Identify the relationships.
5. Identify the cardinality.
6. Draw a Draft.

# QA Create the GAME database's ERD



## Outcome:

- Create an ERD for our GAME database.



## Steps:

### 10 minutes, pairs

- Create an ERD for our GAME database.
- Base it on the tables we've already used: **customers**, **orders**, and **games**.
- Go through each of the creation steps we looked at just now.



## Identify the entity

- We already know what these entities are: **customers**, **orders**, and **games**.

customers

orders

games

## Identify the attributes

- These are the fields we've already got for each of the tables.
- We'll also include some unique identifier fields as well, as it's good practice.

customers

customer ID  
name  
email  
password

orders

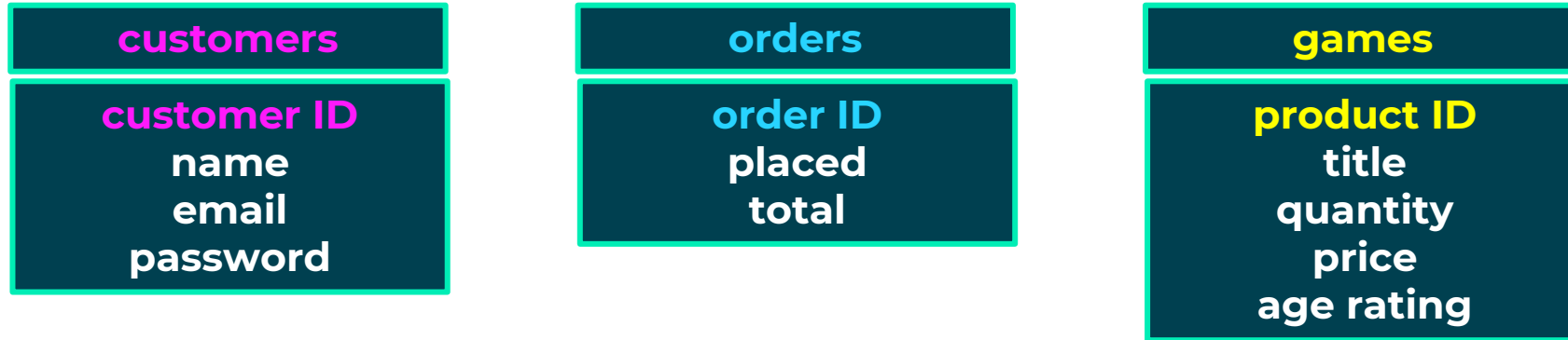
order ID  
placed  
total

games

product ID  
title  
quantity  
price  
age rating

## Identify the primary keys

- We've made fields that are dedicated to being primary keys: all the **ID** fields.



## Identify the relationships and cardinality

- Based on these entities, we can make a simple matrix for this:

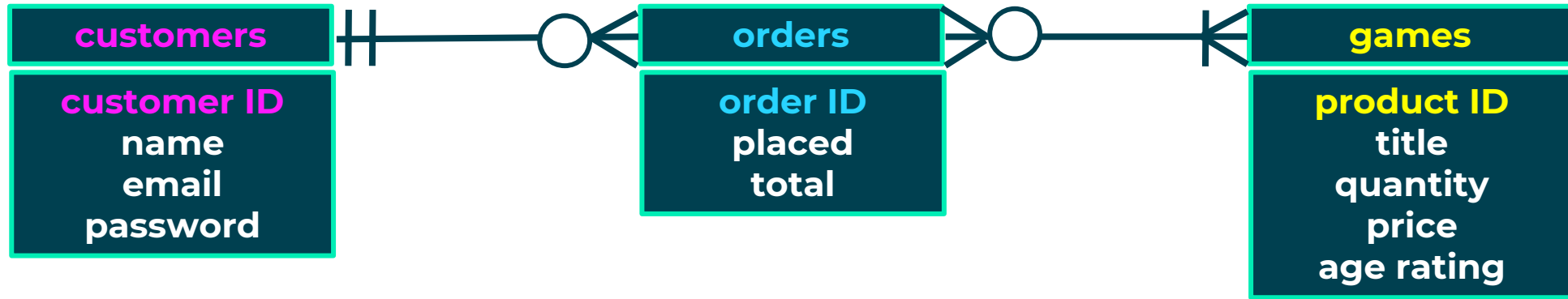
	CUSTOMERS	ORDERS	GAMES
CUSTOMERS	-	0-TO-MANY	N/A
ORDERS	ONLY 1	-	1-TO-MANY
GAMES	N/A	0-TO-MANY	-





## Draw out the draft

- We've made fields that are dedicated to being primary keys: all the **ID** fields.



# QA **Many-to-many (m2m) relationships**

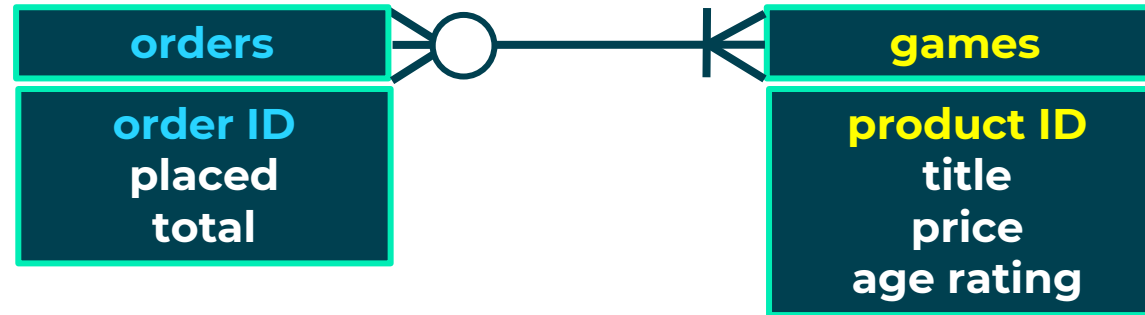
**Whilst we can model m2m relationships with ERDs, we can't make them in MySQL,**

- Because keys have constraints on records, it just isn't possible.

**The best way to model m2m relationships in MySQL is through some kind of middleman.**

- › We will need to have a table in the middle which is dedicated to handling the m2m relationship.

Let's consider this alongside the order and product table within our GAME database:



An order can contain 1-to-many games, and a game can be part of 0-to-many orders:



# QA Creating more tables for the GAME database



## Outcome:

- Add some more tables to our GAME database, ensuring the fields for a primary key.



## Steps:

**10 minutes, solo**

- Create two tables in the GAME database based on the tables shown here:

games				
PRODUCT ID	TITLE	QUANTITY	PRICE	AGE RATING
1	SHOOT THE COOL GUN 9	8965	£79.99	18
2	GUNBLADERS XXII	546	£64.99	15
3	PAINT DRYING SIMULATOR	435	£37.99	3
4	SITAR HERO	456	£45.99	12

orderline		
ORDER ID	CUSTOMER ID	QTY_ORDERED
1	1	1
2	2	1



# Creating tables

**games**

PRODUCT ID	TITLE	QUANTITY	PRICE	AGE RATING
1	SHOOT THE COOL GUN 9	8965	£79.99	18
2	GUNBLADERS XXII	546	£64.99	15
3	PAINT DRYING SIMULATOR	435	£37.99	3
4	SITAR HERO	456	£45.99	12

```
CREATE TABLE games (  
  product_id int NOT NULL AUTO_INCREMENT,  
  title varchar(100) NOT NULL,  
  quantity int NOT NULL,  
  price decimal NOT NULL,  
  age_rating int NOT NULL,  
  PRIMARY KEY (product_id)  
);
```

**orderline**

ORDER ID	CUSTOMER ID	QTY_ORDERED
1	1	1
2	2	1

```
CREATE TABLE orderline (  
  order_id int NOT NULL,  
  product_id int NOT NULL,  
  qty_ordered int,  
  FOREIGN KEY (order_id) REFERENCES orders (order_id),  
  FOREIGN KEY (product_id) REFERENCES games (product_id)  
);
```



# Summary

## Databases: Module 3

### **Begin using SQL to build out a database:**

- CREATE TABLE is used, followed by field names, their data type, and any constraints we need.

### **Describe what the schema of a database is:**

- A schema is the structure of the database and uses SQL's Data Definition Language (DDL) to implement it.

### **Create an ERD based on database tables and keys:**

- We created an ERD for our GAME database that included entities, attributes, and cardinal relationships.

### **Create the GAME database, implementing keys and constraints:**

- We created four tables, using primary and foreign keys as well as constraints, to start off our GAME database.



# Thank you for listening

Any questions?