

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
<div><div></div>Installing MySQL on Windows</div>
<div><div></div>Provision a MySQL Server (Google Cloud Platform)</div>
<div><div></div>Introduction to Relational Databases</div>
<div><div></div>Data Design</div>
<div><div></div>Data Definition Language (DDL)</div>
<div><div></div>Entity-Relationship Diagrams</div>
<div><div></div>Data Manipulation Language (DML)</div>
<div><div></div>Data Query Language using SELECT</div>
<div><div></div>Aggregate Functions</div>
<div><div></div>Nested Queries</div>
<div><div></div>Joins</div>
<div><div></div>Data Normalisation</div>
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS

Entity-Relationship Diagrams

Contents

- [Overview](#)
- [Concept](#)
- [Design](#)
 - [Entity Types & Entities](#)
 - [Attributes](#)
 - [Relationships](#)
 - [Roles](#)
 - [Cardinality](#)
 - [Notation](#)
 - [Many-to-many relationships with MySQL](#)
- [Tutorial](#)
- [Exercises](#)

Overview

An **Entity-Relationship Diagram (ERD)** (sometimes referred to as an **Entity-Relationship Model**) is a type of abstract data model.

When used in the context of a database, an ERD allows you to map out the relationships between your tables.

Concept

In an Agile environment which demands the use of a Relational Database Management System (RDBMS), they are generally used at the beginning of a project when designing the structure of the tables involved.

They allow you to clearly see how every piece of data you store in your database will relate to every other piece of data.

Usually, these are designed following the requirements-gathering stage of the development process.

When making an ERD, the most common questions which arise are:

- what data will the database be storing?
- who will be accessing the system?
- what will the data be used for?

ERDs can be made irrespective of whether you:

- already have data to model (**table-first** modelling)
- don't have any data to begin with at all, and will make it later (**objects-first** modelling)

Design

ERDs tend to be done in two different styles, which are functionally-equivalent:

- Chen** - a *conceptual* model, named after Peter Chen, who first devised ERDs for databases in 1976
- UML** - a *logical* model, using the *Unified Modelling Language* standard, which is also commonly used for class diagrams

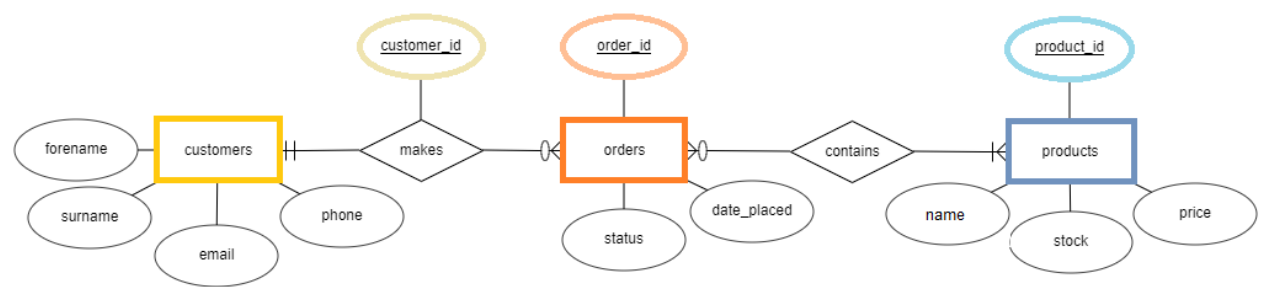
Regardless of style, ERDs will typically contain the following components:

- entity types** - in a relational database, these would be our *tables*

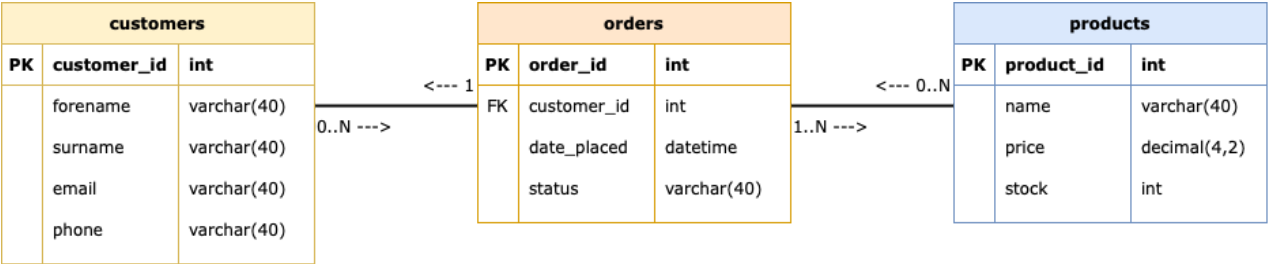
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

- **relationships** - the way that each entity links to each other - these are represented by our **primary** and **foreign keys**
- **attributes** - the way that an entity can be described - in this case, the *columns* in our tables

Traditional ERDs which use the **Chen** style split out entity types and attributes into their own cells, but combine attributes which share a relationship:



ERDs which use the **UML** style keep entity types and their attributes together:



Entity Types & Entities

Entities are objects which can store information - such as a **customer**, an **order**, or a **product**.

Entity types are classes of objects, and are plural - such as **customers**, **orders** or **products** - these are our tables.

The first record in our **customers** entity type is a **customer** entity.

Depending on the visual layout of an ERD, entity types are usually represented by rectangles or by tables, as seen above.

An entity type can be either *strong* or *weak* - denoted by whether one entity type *depends* on another to exist:

- a **strong** entity type has a *primary key* and does not depend on another entity type to exist
- a **weak** entity type has a *foreign key* and depends on another entity type to exist

For instance an **accounts** entity type would be *strong*, because each account should have its own unique id number.

On the other hand, a **favourites** entity type would be weak, because it relies on having an account in the first place.

Attributes

An **attribute** is a property of an entity, or something that can be used to describe an entity - in this case, attributes are the columns in our tables.

There are several different ways to represent attributes:

- **simple** - an attribute which cannot be split down, such as a *forename*
- **composite** - an attribute which can be split down, such as a *name* into *forename*, *middle names* and *surname*
- **derived** - an attribute which is determined by another attribute, such as an *age* which is determined by a *date of birth*

Attributes can also be either **single-value** or **multi-value**:

- *home phone number* and *work phone number* fields would both be single-value
- a *phone number* field which is captured twice would be multi-value

In relational databases, *multi-value* attributes are far less common.

Relationships

Connections between entity types are modelled using *relationships*.

Roles

The *Chen* model above emphasises the naming of these relationships, known as **roles**. (You can use them with UML style, too.)

Roles are written in natural language to explain the nature of a relationship.

For instance, our **customers** and **orders** tables are connected because a **customer makes orders**, or **orders are made by customers**.

We can use a matrix to work out these roles:

X	customers	orders	products
customers	-	make	-
orders	are made by	-	contain
products	-	are part of	-

Reading from the left column, we can see that, for instance, **customers make orders**.

Cardinality

Relationships in ERDs are modelled by using **cardinality**, which shows how each side of a relationship affects the other.

Think of a relationship as a kind of "balance" between two entity types.

Relationships are either *mandatory* or *optional* in nature:

- a **mandatory** relationship requires **at least one** - for instance, an **order** will contain *at least one* **product**
- an **optional** relationship does not have this requirement, so could theoretically be zero - for instance, a **customer** *might not have made any* **orders**




Relationships also refer to quantity:

- a *mandatory* relationship will be **one-to-one**, **one and only one**, or **one-to-many**
- an *optional* relationship will be **zero or one** or **zero-to-many**

There are also non-specific **one** and **many** relationships.

Notation

Relationship notation can differ greatly, depending on the style of ERD used.

relationship	Chen	UML
one-to-one		-
one and only one		1..1
one		1

relationship	Chen	UML
many		N
one-to-many		1..N
zero or one		0..1
zero-to-many		0..N

Many-to-many relationships with MySQL

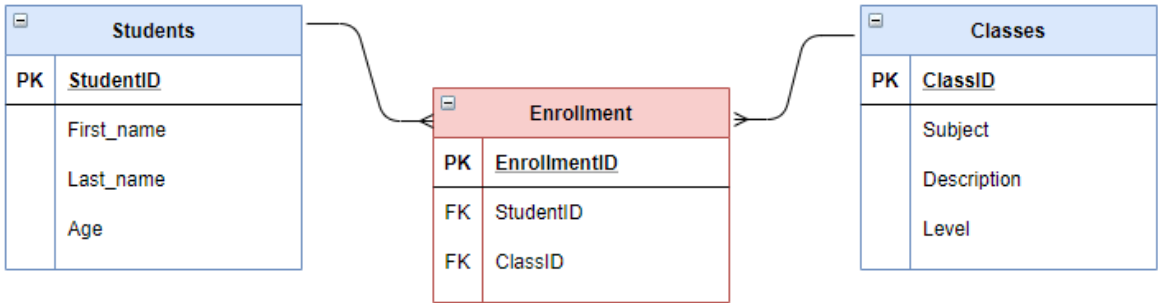
Many-to-many relationships occur when multiple fields in one table are associated with multiple fields in another table.

For example, a student can enrol in multiple classes and a class can include multiple students.

MySQL does not support direct many-to-many relationships between tables.

As such, in order to implement a many-to-many relationship between your tables, you need to create a third table called a "middleman", "child table" or "join table".

In other words, it is possible to split a many-to-many relationship into two one-to-many relationships:



Enrollment is our child table and it contains the values of the primary keys from both the **Students** and **Classes** tables.

The value of the foreign key stored in the **StudentID** field corresponds to each student in **Students**.

The value of the foreign key stored in the **ClassID** field corresponds to each class in **Classes**.

As such, we know which students are included in a particular class.

Tutorial

There is no tutorial for this module.

Exercises

Try to make an ERD for your games shop database. Feel free to mix and match notation styles as you prefer.

(note: if you have not worked out your tables yet, refer back to the [Data Definition Language](#) module for context)

