Allgemein

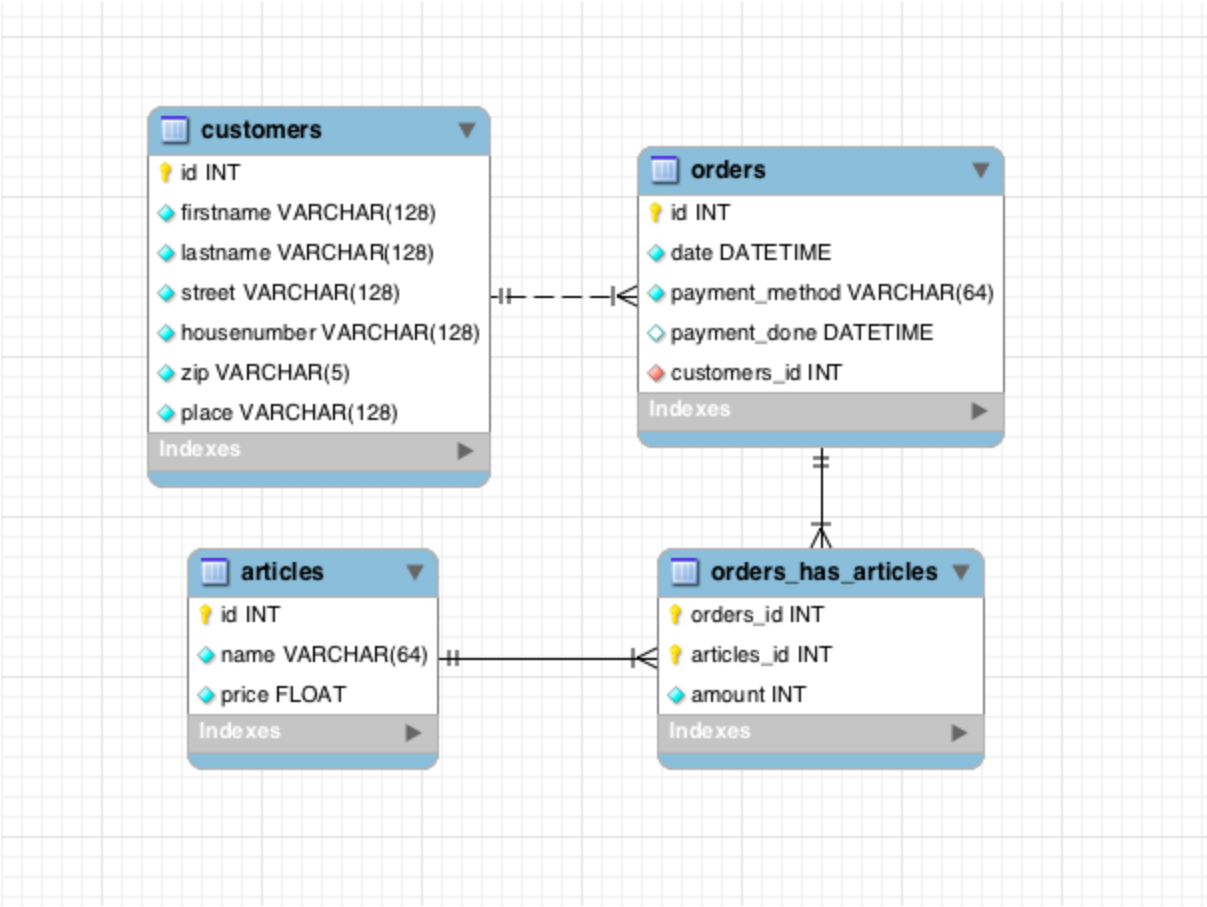
Eine SQL-Datenbank ist eine meist serverseitige Software, die Daten speichern und verwalten kann. Dabei werden diese Daten in Tabellen abgelegt und indiziert.
Der Server reagiert dabei auf Befehle, die entweder direkt vom Endanwender an den Server gerichtet oder z.B. über eine Programmiersprache wie PHP ausgeführt werden.
Durch die Vielzahl von SQL-Server-Herstellern unterscheiden sich die Funktionen und Funktionsweisen der einzelnen Server zum Teil sehr stark voneinander.

Bei SQL-Datenbanken handelt es sich um relationale Datenbanken, deren Entwurf 1970 von Edgar F. Codd erstellt wurde. Seit diesem Zeitpunkt sind relationale Datenbanken der Standard, wenn es darum geht, viele Datensätze zu verarbeiten und zu verwalten.
Grundlage der relationalen Datenbank ist die relationale Algebra.

Die *Tabellen* beschreiben in diesem Modell *Relationen*, wobei die *Zeilen* hier auch *Tupel* genannt werden. Dabei spricht man von einem *Datensatz* oder auch *record*.
Die einzelnen Spalten der Tabelle werden Attribute genannt und beinhalten verschiedene Eigenschaften der Datensätze.
Anhand eines eindeutigen Schlüssels (Primary Key, Primärschlüssel, ID) werden die Datensätze eindeutig gekennzeichnet. Darin ist auch begründet, warum ein Primärschlüssel in einer Relation nur ein einziges Mal vorkommen darf: Mehrere Datensätze dürfen nicht von ein und dem selben Primärschlüssel abhängig sein, da sonst eine eindeutige Referenzierung unmöglich wird.
Wird ein Datensatz gelöscht, so wird dessen ID niemals erneut vergeben.

Datenbanken spielen ihre Stärke allerdings nicht nur dann aus, wenn nur eine einzige Tabelle damit verwendet wird, sondern auch wenn es darum geht, viele Daten miteinander zu verknüpfen.
So können über die Fremdschlüssel (Foreign Keys) in den Attributen Referenzen zu Primärschlüsseln in anderen Tabellen angegeben werden und somit die Daten aus einer anderen Tabelle mit eingebunden werden.

Ein ER-Diagramm



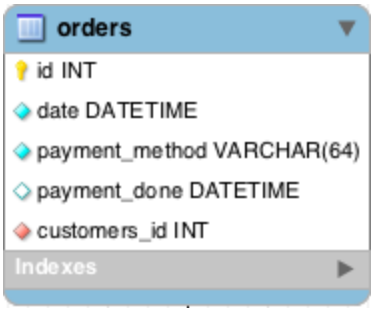
Das oben abgebildete Modell wird auch ER-Diagramm oder Entity Relationship Diagramm genannt. Dabei werden die Tabellen mit ihren Attributen aufgeführt. Primärschlüssel werden mit einem kleinen Schlüssel markiert, oftmals auch unterstrichen.
Weitere Attribute werden darunter aufgelistet.

Anhand eines solchen Modells ist es möglich, eine Datenbank in allen Datenbank-Systemen abzubilden.

Die Tabelle “customers”
Diese Tabelle beschreibt einen Kunden mit den wichtigsten Informationen: Vorname (firstname), Nachname (lastname), Straße (street), Hausnummer (housenumber), Postleitzahl (zip) und Ort (place).

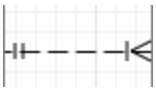
Außerdem hat er eine Kundennummer, die ID. Wird ein neuer Kunde angelegt, müssen nur die oben genannten Informationen angegeben werden, die ID wird beim Einfügen in die Datenbank erzeugt. In den seltensten Fällen sucht man direkt nach der

Kundennummer, meist wird die Kombination aus Vor- und Nachname gewählt, um den Kunden zu identifizieren. Anhand der Kundennummer werden allerdings die Aufträge den Kunden zugewiesen, demnach kann die Kundennummer nicht vernachlässigt werden.



Die Tabelle "orders"
In der Tabelle "orders" werden die Rahmenbedingungen der Aufträge festgelegt, etwa das Datum (date), die gewünschte Zahlungsmethode (payment_method), der Zahlungseingang (payment_done) und der Fremdschlüssel (customers_id) des Kunden.

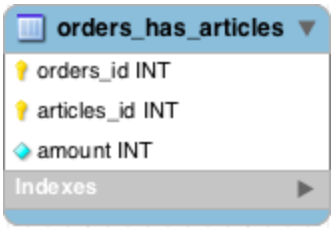
Auch hier gibt es wieder eine ID, in diesem Fall die Auftragsnummer, die den Auftrag eindeutig referenziert.



Die 1:n-Beziehung
Aufgefallen sein dürfte der Pfeil bzw. die Linie zwischen den beiden genannten Tabellen. Hierbei handelt es sich um eine so genannte 1:n-Beziehung, das heißt, dass auf der einen Seite ein Datensatz steht und auf der anderen keiner, einer oder mehrere. Da der Pfeil mit drei Strichen von den Bestellungen ausgeht, heißt dies, dass dort mehrere Aufträge von einem und dem selben Kunden kommen (können). Da es sich hier um das Datenbank-Design dreht, ist dieser Pfeil auch vorhanden, wenn es keine Daten gibt, die sich gegenseitig referenzieren könnten.

Die Tabelle "articles"
In dieser Tabelle findet man die angebotenen Artikel mit Namen (name) und Preisen (price).

Wie bereits aus den vorigen Tabellen bekannt gibt es auch hier einen Primärschlüssel ID, der in diesem Fall die Artikelnummer darstellt.



Die n:m-Beziehung
Da jeder Auftrag nur ein einziges Mal in der Datenbank vorkommen sollte und jeder Artikel ebenfalls nur ein einziges mal gespeichert werden soll, diese allerdings miteinander verknüpft werden müssen, führt man eine neue Tabelle ein, die in jedem Datensatz einmal den Auftrag und einmal den Artikel referenziert.

Besonderheit dieser Tabelle ist der doppelte Primärschlüssel:
Anhand der Auftrags-ID und der Artikel-ID erhält man die bestellte Menge des Artikels in dem entsprechenden Auftrag.
Da ein Auftrag mehrere Artikel beinhalten kann, ein Artikel aber auch in mehreren Aufträgen (dann u.U. mit anderen Mengen) vorhanden sein kann, muss diese Art der Abbildung gewählt werden, um Probleme beim Erfassen der Aufträge zu umgehen.

Regeln

Bei Datenbanken muss man einige Regeln beachten, wenn man effizient mit ihnen arbeiten möchte. Dabei gilt es zum Einen Redundanzen zu vermeiden und zum Anderen Anomalien vorzubeugen. Außerdem helfen sie, den Speicherplatzbedarf der Datenbank klein zu halten.

Diese Regeln nennt man Normalformen. Hierbei unterscheidet man derweil sechs, relevant für unsere Zwecke sind allerdings nur die ersten drei:

erste Normalform (1NF)

Jedes Attribut der Relation muss einen atomaren Wertebereich haben und die Relation muss frei von Wiederholungsgruppen sein.
Kurz: Jedes Feld bzw. jedes Attribut darf nur einen Wert enthalten. Ein Feld “Name”, das “Heinz Müller” beinhaltet, ist nicht zulässig. (atomarer Wertebereich)
Ein Feld “Telefonnummern”, das “0124/123123, 0125/654321, 0126/456456” beinhaltet ist ebenfalls nicht zulässig. (Wiederholungsgruppen)

zweite Normalform (2NF)

=> 1NF wird erfüllt
Kein Nichtschlüsselattribut ist funktional abhängig von einer echten Teilmenge eines Schlüsselkandidaten.
Kurz: Jedes nicht-primäre Attribut ist von allen Schlüsseln abhängig.
Im Beispiel oben ist von der Menge eines Artikels in der Bestellung die Rede. Diese Menge ist nur eindeutig referenzierbar, wenn man Auftrags- und Artikel-ID kennt, demnach ist sie von den beiden Schlüsseln abhängig.

dritte Normalform (3NF)

=> 2NF wird erfüllt
Kein Nichtschlüsselattribut hängt von einem Schlüsselkandidaten transitiv ab.
Kurz: Von einem Nichtschlüssel-Attribut können keine weiteren Daten abhängen, da diese nicht eindeutig referenziert werden würden.
Im Beispiel oben ist ein Artikel abhängig von dem Auftrag, der wiederum vom Kunden aufgegeben wurde. Allerdings sind alle Werte, die andere Relationen einbinden, Primär- respektive Fremdschlüssel, deren Datensätze funktional von diesen abhängig sind.
Die Menge der Artikel zu einem Auftrag sind abhängig vom Auftrag und dem Artikel. Die Menge eines Artikels ohne einen zugehörigen Auftrag oder die Menge eines Artikels in einem Auftrag ohne den zugehörigen Artikel lässt sich demnach nicht (eindeutig) bestimmen.

SQL-Befehle

SELECT

Mit dem `SELECT`-Befehl lassen sich Daten aus einer Tabelle auslesen:

```
SELECT feld1, feld2, feld3
FROM tabelle1;
```

Hier wird eine Abfrage über drei Felder (`feld1`, `feld2`, `feld3`) gemacht, die sich in der Tabelle `tabelle1` befinden. Gefunden werden alle Datensätze in der Tabelle ohne irgendwelche Daten zu sortieren oder auszuschließen.

```
SELECT feld1, feld2, feld3
FROM tabelle1
WHERE feld1 = 'test';
```

In diesem Beispiel werden wieder die drei Felder aus der Tabelle abgefragt, allerdings wird hier eine Bedingung angegeben, die auf `feld1` zutreffen muss: In diesem Feld muss `test` stehen, sonst wird der Datensatz nicht gefunden und dementsprechend auch nicht ausgegeben.
Dabei lassen sich mehrere Bedingungen miteinander verknüpfen:

```
SELECT feld1, feld2, feld3
FROM tabelle1
WHERE feld1 = 'wert1' AND feld2 = 'wert2';
```

```
SELECT feld1, feld2, feld3
FROM tabelle1
WHERE feld1 = 'wert1' OR feld2 = 'wert2';
```

Die erste Abfrage bedingt, dass das erste Feld den ersten Wert beinhaltet und das zweite Feld den zweiten Wert beinhaltet.
Die zweite Abfrage bedingt, dass entweder das erste Feld den ersten Wert beinhaltet oder das zweite Feld den zweiten Wert; beides zusammen ist in dieser Abfrage nicht möglich.

```
SELECT feld1, feld2, feld3
FROM tabelle1
WHERE feld1 > 'wert1';
```

Außerdem lassen sich Vergleiche in der Tabelle ausführen, so zum Beispiel hier, wo man alle Datensätze auswählt, in denen das erste Feld größer ist als der angegebene Wert.

```
SELECT feld1, feld2, feld3
FROM tabelle1
WHERE feld1 LIKE '%wert%';
```

In diesem Beispiel wird im ersten Feld ein Wert gesucht, der ähnlich (`LIKE`) dem übergebenen Wert ist. Dieser steht dabei in Prozentzeichen, was bedeutet, dass beliebig viele Zeichen davor und beliebig viele Zeichen danach folgen können. Möchte man feststellen, ob der Wert am Ende des Feldes steht, so kann man das hintere Prozentzeichen auch weglassen. Ebenso kann man feststellen, ob der Wert am Anfang des Feldes steht, indem man das führende Prozentzeichen weglässt.

```
SELECT feld1, feld2, feld3
FROM tabelle1
ORDER BY feld1;
```

Hier wird eine Sortierreihenfolge über den Befehl `ORDER BY` festgelegt. Standardmäßig wird die Tabelle dann anhand des darauffolgend angegebenen Feldes aufsteigend sortiert. Dies kann aber je nach Server-Hersteller und Konfiguration variieren. Möchte man sichergehen, wie sortiert wird, so sollte man hinter den Feldnamen mit einem Leerzeichen getrennt noch ein `ASC` (ascending, aufsteigend) oder ein `DESC` (descending, absteigend) schreiben.

INSERT

Mit dem `INSERT`-Befehl lassen sich Daten in eine Datenbank-Tabelle einfügen:

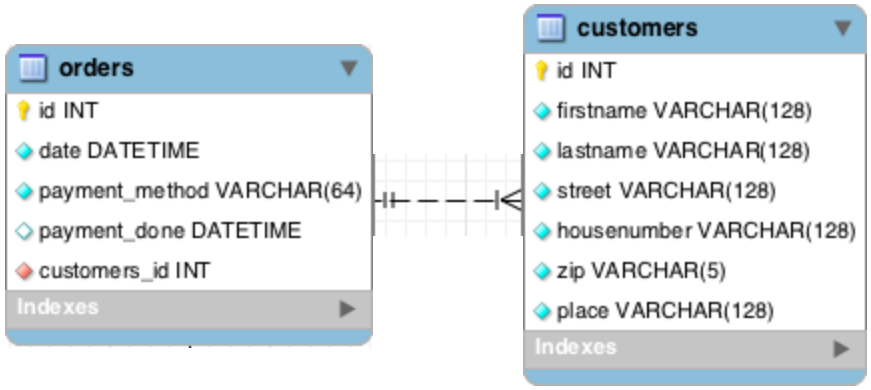
```
INSERT INTO tabelle1 (feld1, feld2, feld3)
VALUES ('wert1', 'wert2', 'wert3');
```

Damit werden mehrere Werte (so genannte `VALUES`) in eine Tabelle eingefügt und dauerhaft gespeichert. Dabei ist die Reihenfolge der Werte zu beachten und so anzugeben, wie in der Parameter-Liste der Tabelle. Es sind meist nicht alle Felder unbedingt notwendig, um einen erfolgreichen Eintrag in eine Datenbank zu machen. So kann man in einer Tabelle, in der ein Primärschlüssel definiert wurde, die Spalte, die diesen beinhaltet, vernachlässigen. Sie wird beim Einfügen des neuen Datensatzes automatisch gefüllt.
Wichtig ist: Bei Fließkommazahlen (z.B. 1,25 und 4,56) ist zu beachten, dass die Kommata durch Punkte ersetzt werden müssen, da sonst in der Liste der einzufügenden Werte unter Umständen ein Element mehr erkannt wird. Durch Kommata werden hier nämlich die Werte getrennt.

JOINS

Mit den so genannten **JOINS** werden zwei oder mehr Tabellen miteinander verknüpft. Hier kommen die Primär- und Sekundärschlüssel ins Spiel. Fremdschlüssel definieren einen eindeutigen Datensatz in einer anderen Tabelle, die in eben jener Tabelle als Primärschlüssel definiert sind. Dadurch ist es möglich, Datensätze einzubinden, ohne darauf angewiesen zu sein. Anhand des Fremdschlüssels könnte man daran anknüpfende Datensätze hinzuziehen, sofern man weitere Informationen benötigt. Allerdings kann man diese auch vollständig ignorieren, sodass die Geschwindigkeit für weitere Abfragen auf andere Tabellen ausbleiben.

“Joins” können auf alle Arten von Beziehungen ausgeführt werden, solange entsprechende Fremdschlüssel definiert wurden.



Die Tabelle **orders** hat den Fremdschlüssel customers_id und referenziert den Primärschlüssel id in der Tabelle **customers**. Wie bereits oben beschrieben, handelt es sich hierbei um eine 1:n-Beziehung, da mehrere Aufträge vom selben Kunden stammen können und ein Kunde mehrere Aufträge haben kann.
Es können auch Kunden existieren, die keinen Auftrag abgegeben haben. Allerdings kann es keinen Auftrag ohne einen Kunden geben.

Möchte man nun alle Aufträge eines bestimmten Kunden erhalten, muss man zuerst die Kundennummer herausfinden und dann in den Aufträgen nach dieser Kundennummer in der Spalte **customers_id** suchen. Dieses Verfahren ist langwierig und fehleranfällig, sofern die Kundennummer händisch übertragen und nicht kopiert wird.

Um zu einem Kunden alle Aufträge zu erhalten, setzen wir erneut **SELECT** ein, um Daten aus der Datenbank zu lesen:

```
SELECT customers.id, customers.firstname, customers.lastname,
orders.id FROM customers
INNER JOIN orders ON customers.id = orders.customers_id;
```

Bei dieser Abfrage werden nun die folgenden Informationen über den Kunden und dessen Aufträge aus der Datenbank gelesen: Kundennummer, Vorname, Nachname und Auftrags-ID. Wichtig ist bei einem **JOIN**, dass trotzdem sich die Abfrage über mehrere Tabellen erstreckt, eine Tabelle angegeben werden muss, die als Datenquelle für das `SELECT` dient. Sonst funktioniert die Abfrage nicht, da die Quell-Tabelle, mit der die “gejointen” Daten verknüpft werden sollen, schlichtweg nicht auftauchen würde.

JOINs erweitern also die SELECT-Abfragen, um mehrere Tabellen in die Suche und Auswertung einzubeziehen.

Unter den verschiedenen JOIN-Arten versteht man die Verknüpfung der Daten untereinander, so müssen in einem **INNER JOIN** alle Datensätze der “linken” und “rechten” Tabelle entsprechende Wertepaare haben (Primär- und Fremdschlüssel).
LEFT JOINs heben diese Limitierung auf und binden alle Daten der “linken” Tabelle ein, auch wenn in der “rechten” Tabelle keine entsprechenden Werte vorliegen.
RIGHT JOINs funktionieren ähnlich wie die LEFT JOINS, nur werden alle Daten der “rechten” Tabelle eingebunden, auch wenn in der “linken” Tabelle nicht alle Wertepaare übereinstimmen.
FULL OUTER JOINs werden dazu verwendet, zwei Tabellen in ihrer Gesamtheit miteinander zu verknpüfen, ungeachtet dessen, ob die Wertepaare “links” oder “rechts” vorkommen.

Die Bezeichnungen “links” und “rechts” beziehen sich hier auf die Angabe der Werte in der so genannten **ON**-Clause des JOINs. Die Tabelle, in der die Spalte vorkommt, die links vom Gleichheitszeichen steht, ist die “linke”, die rechts ist die “rechte”.
Somit kann man auf LEFT JOINs oder RIGHT JOINs verzichten, indem man die Gleichung einfach umdreht.

Übungen

SELECT

Es ist eine Tabelle **articles** gegeben:

id	name	price
1	Apfel	0.89
2	Tomate	0.49
3	Gurke	0.2
4	Flasche Wasser	1.49
5	Chips	2.39

1. Aus der Tabelle soll der Preis des Artikels ausgegeben werden, der "Tomate" heißt.
2. Aus der Tabelle sollen alle Namen und Preise von Artikeln ausgegeben werden, deren Preis kleiner als 1 ist.
3. Aus der Tabelle sollen die Artikel gewählt werden, die "Chips" heißen und deren Preis kleiner als 2 ist.
4. Aus der Tabelle sollen alle Artikel nach dem Alphabet sortiert werden – in absteigender Reihenfolge.
5. Die Befehle `WHERE` und `ORDER BY` lassen sich auch zusammenführen. In welcher Reihenfolge ergibt die Aneinanderreihung mehr Sinn und warum? Denke dabei auch an große Datenbestände!

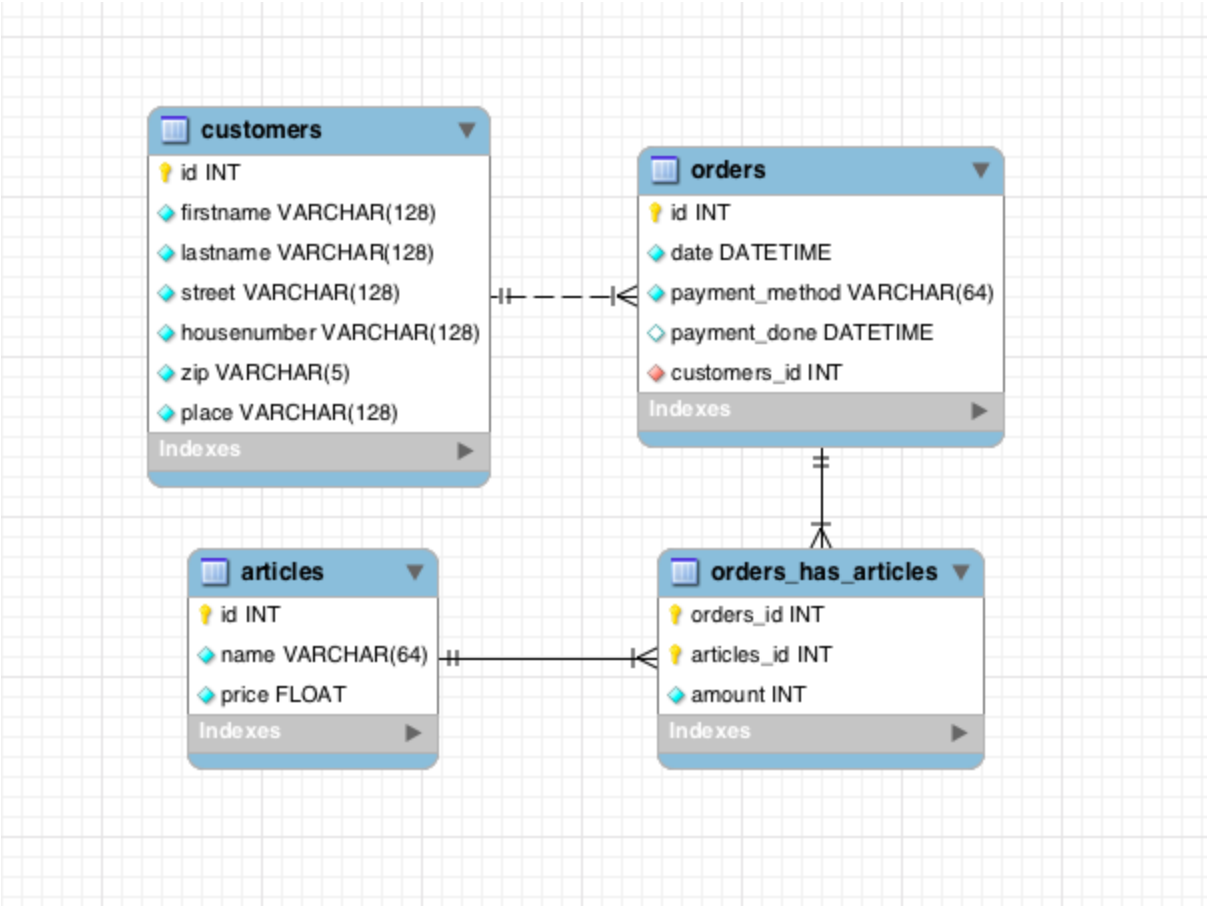
INSERT

Aus der vorherigen Aufgabe ist die Tabelle **articles** gegeben.

1. In die Tabelle sollen die folgenden Artikel eingetragen werden:
   a. Birnen zum Preis von 0,39 Euro
   b. Zwetschgen zum Preis von 1,11 Euro
   c. Erdbeeren zum Preis von 1,40 Euro
2. Es sind die folgenden `INSERT`-Befehle gegeben. Prüfe sie auf Vollständigkeit und Korrektheit
   a.
   ```
   INSERT INTO articles (name, price)
   VALUES ('Rotbeeren', 0.39);
   ```
   b.
   ```
   INERT INTO articles (name, price)
   VALUES ('Stachelbeeren, 2,50);
   ```
   c.
   ```
   INSERT INTO articles (name, price)
   VALUES ('Ei', 0,27);
   ```

JOIN

Es ist folgendes ER-Diagramm mit den Tabellen **customers**, **orders**, **articles** und **orders_has_articles** gegeben.



1. Erläutere kurz den Aufbau der Datenbank und den Tabellen. Gehe dabei auch auf die Verknüpfungen der einzelnen Tabellen untereinander ein.
2. Erläutere den folgenden SQL-Befehl:

```
SELECT customers.id, customers.firstname, customers.lastname,
orders.id FROM customers
INNER JOIN orders ON customers.id = orders.customers_id;
```

3. Eine Besonderheit dieser Tabelle stellt die n:m-Beziehung von der Tabelle **orders** zur Tabelle **articles** dar. Warum ist diese Verbindung sinnvoll und notwendig?
4. Notiere einen SQL-Befehl, der zu einem Kunden mit dem Vornamen Michael und dem Nachnamen Peters alle Auftrags-IDs ausgibt.
5. Notiere einen SQL-Befehl, der alle Artikelnamen, sowie deren Preise und Mengen des Auftrags mit der ID 45 ausgibt.