Meine Datenbank dient zur Verwaltung von Verkäufen in einem Shop. Sie enthält Informationen über Kunden, Lieferanten, Artikel, Verkäufe, Zahlungen und Rabatte. Die Struktur ist so aufgebaut, dass sie die Beziehungen zwischen diesen Elementen effizient verwalten kann.

Zunächst wurde eine neue Datenbank mit dem Namen Shop\_DB\_ihr name erstellt:

create DATABASE Shop\_DB\_Akbarifar;

use shop\_DB\_Akbarifar;

Meine Datenbank ist so aufgebaut, dass sie alle wichtigen Geschäftsprozesse eines Shops abbildet – von der Artikelverwaltung über Lieferanten und Verkäufe bis hin zu Zahlungen und Rabatten. Dadurch können alle Vorgänge effizient und übersichtlich verwaltet werden.

**Die Datenbank enthält die folgenden Tabellen:**

**kunden (Kundenverwaltung)**

**Diese Tabelle speichert Informationen über Kunden wie Name, Adresse, Telefonnummer und E-Mail.**

**Warum? Damit wir wissen, wer etwas gekauft hat.**

CREATE TABLE kunden (

kunden\_id INT AUTO\_INCREMENT PRIMARY KEY,

vorname VARCHAR(50) NOT NULL,

nachname VARCHAR(50) NOT NULL,

straße VARCHAR(100),

hausnummer VARCHAR(10),

postleitzahl VARCHAR(10),

stadt VARCHAR(50),

telefonnummer VARCHAR(20) UNIQUE, -- Keine doppelten Telefonnummern

email VARCHAR(100) UNIQUE NOT NULL, -- E-Mail muss vorhanden sein

kunden\_typ ENUM('Privatkunde', 'Firma') DEFAULT 'Privatkunde' -- Optional: Kundentyp

);

**lieferanten (Lieferantenverwaltung)**

**Hier werden Informationen über Lieferanten gespeichert, die die Artikel an den Shop liefern.**

**Warum? Damit wir wissen, von wem die Artikel kommen.**

CREATE TABLE lieferanten (

lieferanten\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

straße VARCHAR(100),

hausnummer VARCHAR(10),

postleitzahl VARCHAR(10),

stadt VARCHAR(50),

telefonnummer VARCHAR(20) UNIQUE, -- Keine doppelten Telefonnummern

email VARCHAR(100) UNIQUE NOT NULL -- E-Mail muss vorhanden sein

);

**artikel (Artikelverwaltung)**

**Enthält die grundlegenden Informationen über Produkte, wie Name und Beschreibung.**

**Warum? Weil verschiedene Lieferanten den gleichen Artikel zu unterschiedlichen Preisen anbieten können.**

CREATE TABLE artikel (

artikel\_id INT AUTO\_INCREMENT PRIMARY KEY,

bezeichnung VARCHAR(100) NOT NULL,

beschreibung TEXT

);

**artikel\_lieferant (Artikel-Zuordnung zu Lieferanten)**

**Diese Tabelle verbindet Artikel mit den Lieferanten, die sie verkaufen, inklusive Preis und Lagerbestand.**

**Warum? Damit wir den Preis und die Verfügbarkeit eines Artikels von verschiedenen Lieferanten nachvollziehen können.**

CREATE TABLE artikel\_lieferant (

artikel\_lieferant\_id INT AUTO\_INCREMENT PRIMARY KEY,

artikel\_id INT,

lieferanten\_id INT,

preis DECIMAL(10,2) NOT NULL, -- Preis des Artikels von diesem Lieferanten

lagerbestand INT NOT NULL, -- Lagerbestand des Artikels bei diesem Lieferanten

FOREIGN KEY (artikel\_id) REFERENCES artikel(artikel\_id) ON DELETE CASCADE,

FOREIGN KEY (lieferanten\_id) REFERENCES lieferanten(lieferanten\_id) ON DELETE CASCADE

);

**verkauf (Verkaufsverwaltung)**

**Speichert die Verkäufe mit Datum, Kunden-ID und dem Gesamtbetrag der Bestellung.**

**Warum? Weil wir wissen müssen, welche Kunden wann und für welchen Betrag eingekauft haben.**

CREATE TABLE verkauf (

verkauf\_id INT AUTO\_INCREMENT PRIMARY KEY,

kunden\_id INT,

datum DATE NOT NULL,

betrag DECIMAL(10,2) NOT NULL,

status ENUM('Offen', 'Bezahlt', 'Storniert') DEFAULT 'Offen',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (kunden\_id) REFERENCES kunden(kunden\_id) ON DELETE CASCADE,

);

**verkauf\_details (Details zu einem Verkauf)**

**Hier wird gespeichert, welche Artikel in welchem Verkauf enthalten sind, inklusive Menge und Preis.**

**Warum? Damit wir jede Bestellung im Detail nachvollziehen können.**

CREATE TABLE verkauf\_details (

verkauf\_details\_id INT AUTO\_INCREMENT PRIMARY KEY,

verkauf\_id INT,

artikel\_lieferant\_id INT,

menge INT NOT NULL,

preis DECIMAL(10,2) NOT NULL,

FOREIGN KEY (verkauf\_id) REFERENCES verkauf(verkauf\_id) ON DELETE CASCADE,

FOREIGN KEY (artikel\_lieferant\_id) REFERENCES artikel\_lieferant(artikel\_lieferant\_id) ON DELETE CASCADE

);

**zahlung (Zahlungsverwaltung)**

**Speichert die Zahlungen, die Kunden für ihre Einkäufe leisten, mit Zahlungsart und Datum.**

**Warum? Damit wir wissen, welche Verkäufe bezahlt wurden und mit welcher Methode.**

CREATE TABLE zahlung (

zahlung\_id INT AUTO\_INCREMENT PRIMARY KEY,

verkauf\_id INT,

betrag DECIMAL(10,2) NOT NULL, -- Betrag der Zahlung

zahlungsart ENUM('Bar', 'Kreditkarte', 'Überweisung') NOT NULL, -- Zahlungsart

zahlungsdatum DATE NOT NULL, -- Datum der Zahlung

FOREIGN KEY (verkauf\_id) REFERENCES verkauf(verkauf\_id) ON DELETE SET NULL

);

rabatt (Rabatte auf Verkäufe)

Hier werden Rabatte für Verkäufe gespeichert, mit Prozentsatz und Betrag.

Warum? Damit wir sehen können, welche Rabatte angewendet wurden und wie viel gespart wurde.

CREATE TABLE rabatt (

rabatt\_id INT AUTO\_INCREMENT PRIMARY KEY,

verkauf\_id INT,

rabatt\_prozent DECIMAL(5,2) NOT NULL,

betrag DECIMAL(10,2) NOT NULL, -- مقدار تخفیف به یورو

FOREIGN KEY (verkauf\_id) REFERENCES verkauf(verkauf\_id) ON DELETE CASCADE

);

**Nachdem die Tabellen erstellt wurden, wurden Beispielwerte in die Datenbank eingefügt, um die Funktionalität der Abfragen zu testen.**

INSERT INTO kunden (vorname, nachname, straße, hausnummer, postleitzahl, stadt, telefonnummer, email, kunden\_typ)

VALUES

('Majid', 'Akbarifar', 'Auenbruggergasse', '5', '8073', 'Feldkirchen Bei Graz', '068110861672', 'majidakbarifar2@gmail.com', 'Privatkunde'),

('Maryam', 'Jalali', 'Auenbruggergasse', '5', '8073', 'Feldkirchen Bei Graz', '06606722612', 'maryamlalali094@gmail.com', 'Privatkunde'),

('Johann', 'Schmidt', 'Hauptstraße', '12', '8010', 'Graz', '06641234567', 'johann.schmidt@gmail.com', 'Firma'),

('Elisabeth', 'Müller', 'Kaiserstraße', '25', '8020', 'Graz', '06761234567', 'elisabeth.mueller@yahoo.com', 'Privatkunde');

INSERT INTO lieferanten (name, straße, hausnummer, postleitzahl, stadt, telefonnummer, email)

VALUES

('Bio Frisch', 'Marktstraße', '10', '8010', 'Graz', '06641234501', 'info@biofrisch.at'),

('Steirer Hof', 'Hauptplatz', '5', '8020', 'Graz', '06762345602', 'office@steirerhof.at'),

('Naturgut', 'Bauernweg', '8', '8051', 'Graz', '06873456703', 'kontakt@naturgut.at');

INSERT INTO artikel (bezeichnung, beschreibung)

VALUES

('Apfel', 'Frischer roter Apfel'), ('Brot', 'Vollkornbrot 500g'), ('Milch', '1L Bio-Vollmilch'), ('Käse', 'Gouda-Käse 250g'), ('Eier', '10 frische Eier');

INSERT INTO artikel\_lieferant (artikel\_id, lieferanten\_id, preis, lagerbestand)

VALUES

(1, 3, 1.30, 100), (2, 2, 2.50, 50), (3, 3, 1.80, 70), (4, 1, 3.40, 40), (5, 2, 3.00, 60);

INSERT INTO verkauf (kunden\_id, datum, betrag, status)

VALUES

(1, STR\_TO\_DATE('12.02.2024', '%d.%m.%Y'), 3.60, 'Bezahlt'),

(2, STR\_TO\_DATE('14.02.2024', '%d.%m.%Y'), 3.00, 'Offen'),

(3, STR\_TO\_DATE('16.02.2024', '%d.%m.%Y'), 7.20, 'Bezahlt');

INSERT INTO verkauf\_details (verkauf\_id, artikel\_lieferant\_id, menge, preis)

VALUES

(1, 3, 2, 1.80), -- Majid kaufte 2 Milch

(2, 5, 1, 3.00), -- Maryam kaufte 1 Packung Eier

(3, 1, 6, 1.20); -- Johann kaufte 6 Äpfel

INSERT INTO zahlung (verkauf\_id, betrag, zahlungsart, zahlungsdatum)

VALUES

(1, 3.60, 'Kreditkarte', STR\_TO\_DATE('13.02.2024', '%d.%m.%Y')),

(2, 3.00, 'Bar', STR\_TO\_DATE('15.02.2024', '%d.%m.%Y')),

(3, 7.20, 'Überweisung', STR\_TO\_DATE('17.02.2024', '%d.%m.%Y'));

INSERT INTO rabatt (verkauf\_id, rabatt\_prozent, betrag)

VALUES

(1, 10.00, 0.36), -- 10% Rabatt für Majid

(2, 5.00, 0.15), -- 5% Rabatt für Maryam

(3, 15.00, 1.08); -- 15% Rabatt für Johann

**Da die Datenbank und Tabellen erfolgreich erstellt wurden und bereits Daten enthalten.**

**Erstellen Sie eine Abfrage, die alle Kunden aus der Tabelle kunden abruft.**

select \* from kunden

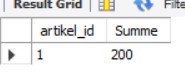
**Erstellen Sie eine Abfrage, die alle Lieferanten aus der Tabelle lieferanten abruft.**

select \* from lieferanten

**Erstellen Sie eine Abfrage, die alle Artikel aus der Tabelle artikel abruft.**

select \* from artikel

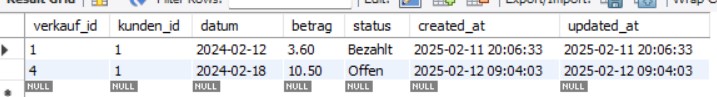
**Abfrage des Lagerbestands eines bestimmten Artikels:**

select artikel\_id,sum(lagerbestand) as Summe from artikel\_lieferant

where artikel\_id = 1

-- group by artikel\_id having artikel\_id = 1

**Abfrage aller Verkäufe eines bestimmten Kunden:**

****select \* from verkauf

where kunden\_id = 1

**Abfrage aller Verkäufe eines bestimmten Lieferanten:**

select verkauf.\* from verkauf

join verkauf\_details on verkauf.verkauf\_id = verkauf\_details.verkauf\_id

join artikel\_lieferant on artikel\_lieferant.artikel\_lieferant\_id = verkauf\_details.verkauf\_details\_id

where lieferanten\_id = 1



**Abfrage aller Artikel, die unter einem bestimmten Preis liegen:**

select artikel.\*,artikel\_lieferant.preis from artikel

join artikel\_lieferant artikel\_lieferant.artikel\_id= artikel.artikel\_id

where preis <= 2.50

**Abfrage zum Ändern des Lagerbestands eines Artikels um eine bestimmte Menge**

update artikel\_lieferant

set lagerbestand = 99

where lieferanten\_id = 3 and artikel\_id=1;

**Abfrage zum Löschen eines Kunden und aller zugehörigen Verkäufe**

delete from kunden

where kunden\_id =3

**Abfrage zum Löschen eines Artikels und aller zugehörigen Verkäufe**

delete from artikel

where artikel\_id =3

**Abfrage aller Verkäufe mit den Informationen zu Kunde, Lieferant und Artikel**

select verkauf.verkauf\_id,kunden.vorname,kunden.nachname,lieferanten.name name\_Lieferant,artikel.bezeichnung,artikel\_lieferant.preis

from verkauf

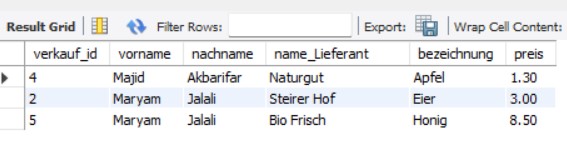
join kunden on verkauf.kunden\_id = kunden.kunden\_id

join verkauf\_details on verkauf\_details.verkauf\_id = verkauf.verkauf\_id

join artikel\_lieferant on artikel\_lieferant.artikel\_lieferant\_id = verkauf\_details.artikel\_lieferant\_id

join lieferanten on lieferanten.lieferanten\_id = artikel\_lieferant.lieferanten\_id

join artikel on artikel.artikel\_id = artikel\_lieferant.artikel\_id



**Abfrage aller Artikel und ihrer Lieferanten**

select lieferanten.name as name\_Lieferant,artikel.bezeichnung,artikel.beschreibung

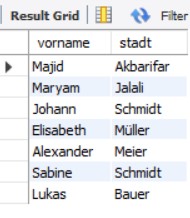
from artikel

join artikel\_lieferant on artikel\_lieferant.artikel\_id= artikel.artikel\_id

join lieferanten lieferanten.lieferanten\_id = artikel\_lieferant.lieferanten\_id

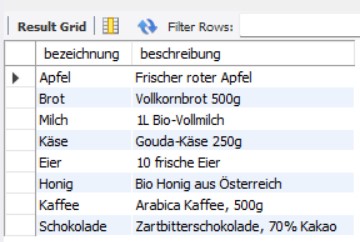


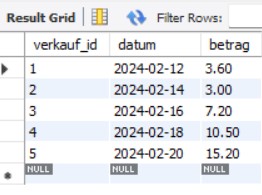
**Ein paar mehr Quary Fragen:**

1. **Die Zeige alle Kunden mit ihren Namen und Städten.**

**select vorname, nachname stadt from kunden**

1. **Zeige alle Artikel mit ihre Beschreibung.**

**select bezeichnung, beschreibung from artikel**

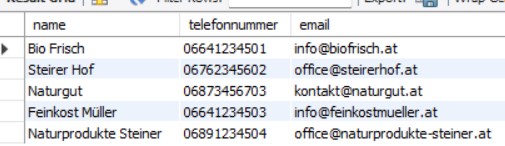
****

1. **Zeige alle Verkäufe mit Datum und Betrag.**

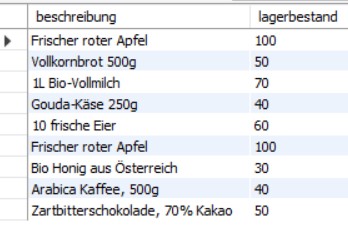
**select verkauf\_id,datum, betrag from verkauf**

1. **Zeige alle Lieferanten mit ihrer Telefonnummer und E-Mail-Adresse.**

**select name,telefonnummer,email from lieferanten**

****

1. **Zeige alle Artikel, die auf Lager sind (Lagerbestand > 0).**

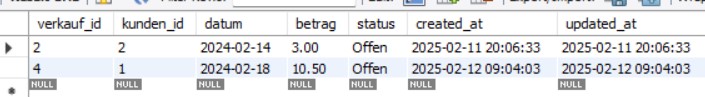
**select artikel.beschreibung,artikel\_lieferant.lagerbestand**

**from artikel\_lieferant**

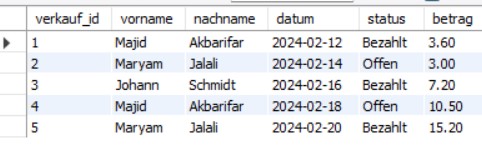
**join artikel on artikel\_lieferant.artikel\_id = artikel.artikel\_id**

**where lagerbestand > 0**

1. **Zeige alle Verkäufe, die noch nicht bezahlt wurden.**

**select \* from verkauf**

**where status = 'Offen'**

****

1. **Zeige alle Verkäufe mit dem Namen des Kunden.**

**select verkauf.verkauf\_id, kunden.vorname,kunden.nachname,**

**verkauf.datum,verkauf.status,verkauf.betrag**

**from verkauf**

**join kunden on verkauf.kunden\_id = kunden.kunden\_id**

****

1. **Zeige alle Artikel, die mindestens einmal verkauft wurden.**

**select distinct artikel.bezeichnung**

**from artikel**

**join artikel\_lieferant on artikel.artikel\_id = artikel\_lieferant.artikel\_id**

**join verkauf\_details on artikel\_lieferant.artikel\_lieferant\_id = verkauf\_details.artikel\_lieferant\_id**

1. **Zeige die gesamte Summe aller Verkäufe.**

**select sum(verkauf.betrag) as Summer\_Verkauf**

**from verkauf;**

1. **Zeige die Anzahl aller Verkäufe.**

**select count(verkauf.verkauf\_id) as Count\_Verkauf**

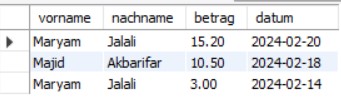
** from verkauf;**

**11-Zeige die letzten 3 Verkäufe nach Datum sortiert (nachname-vorname-betrag und datum)**

**select kunden.vorname,kunden.nachname,verkauf.betrag,verkauf.datum**

**from verkauf**

**join kunden on verkauf.kunden\_id = kunden.kunden\_id**

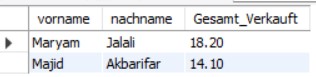
**order by verkauf.datum desc**

**limit 3;**

**12- Zeige die Gesamtsumme der Verkäufe für jeden Kunden**

**select kunden.vorname,kunden.nachname,sum(verkauf.betrag) as Gesamt\_Verkauft**

**from verkauf**

**join kunden on verkauf.kunden\_id = kunden.kunden\_id**

**group by kunden.vorname, kunden.nachname**

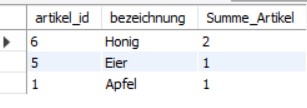
**order by Gesamt\_Verkauft desc**

**13-** **Zeige die meistverkauften Artikel mit der Gesamtmenge**

**select artikel.artikel\_id,artikel.bezeichnung,sum(verkauf\_details.menge) as Summe\_Artikel**

**from verkauf\_details**

**join artikel\_lieferant on verkauf\_details.artikel\_lieferant\_id = artikel\_lieferant.artikel\_lieferant\_id**

**join artikel on artikel.artikel\_id = artikel\_lieferant.artikel\_id**

**group by artikel.bezeichnung,artikel.artikel\_id**

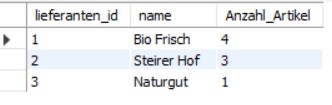
**order by Summe\_Artikel desc**

**14-** **Zeige alle Lieferanten mit der Anzahl der gelieferten Artikel**

**select lieferanten.lieferanten\_id, lieferanten.name, COUNT(artikel\_lieferant.artikel\_id) AS Anzahl\_Artikel**

**from lieferanten**

**join artikel\_lieferant on lieferanten.lieferanten\_id= artikel\_lieferant.lieferanten\_id**

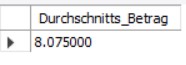
**group by lieferanten.lieferanten\_id, lieferanten.name**

**order by Anzahl\_Artikel desc**

**15- Zeige die Gesamtanzahl aller Artikel im System**

**select count(artikel.artikel\_id) as Summe\_Artikel**

**from artikel**

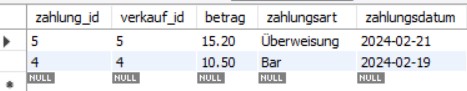
**16- Zeige den durchschnittlichen Betrag pro Verkauf**

**SELECT AVG(verkauf.betrag) as Durchschnitts\_Betrag**

**FROM verkauf;**

**17-** **Zeige alle Zahlungen, die größer als 10 Euro sind**

**SELECT zahlung.zahlung\_id, zahlung.verkauf\_id, zahlung.betrag, zahlung.zahlungsart, zahlung.zahlungsdatum**

**FROM zahlung**

**WHERE zahlung.betrag > 10**

**ORDER BY zahlung.betrag DESC;**

**18- Zeige die Gesamtverkäufe pro Monat**

**SELECT date\_format(verkauf.datum, '%Y-%m') as monat,sum(verkauf.betrag) as Gesamt\_Umsatz**

**from verkauf**

**GROUP BY DATE\_FORMAT(verkauf.datum, '%Y-%m')**

**ORDER BY Monat DESC;**

**19- Zeige alle Artikel, die Rabatte erhalten haben, mit Rabattprozent**

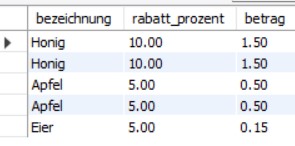
**SELECT artikel.bezeichnung, rabatt.rabatt\_prozent, rabatt.betrag**

**from rabatt**

**join verkauf on rabatt.verkauf\_id = verkauf.verkauf\_id**

**join verkauf\_details on verkauf.verkauf\_id = verkauf\_details.verkauf\_id**

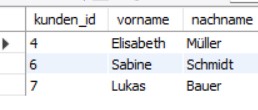
**join artikel\_lieferant on verkauf\_details.artikel\_lieferant\_id = artikel\_lieferant.artikel\_lieferant\_id**

**join artikel on artikel\_lieferant.artikel\_id = artikel.artikel\_id**

**ORDER BY rabatt.rabatt\_prozent DESC;**

**20- Zeige alle Kunden, die noch keine Einkäufe getätigt haben**

**SELECT kunden.kunden\_id, kunden.vorname, kunden.nachname**

**from kunden**

**left join verkauf on kunden.kunden\_id = verkauf.kunden\_id**

**WHERE verkauf.verkauf\_id IS NULL;**

**Erstellen Sie folgende Views und Transaktionen**

1. **Erstellung einer View, die alle Kunden und deren Umsätze listet**

create view v\_kunden\_umsaetze as

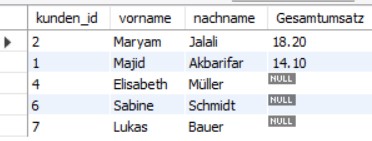
select kunden.kunden\_id, kunden.vorname, kunden.nachname,

SUM(verkauf.betrag) as Gesamtumsatz

from kunden

LEFT JOIN verkauf ON kunden.kunden\_id = verkauf.kunden\_id

GROUP BY kunden.kunden\_id, kunden.vorname, kunden.nachname

****ORDER BY Gesamtumsatz DESC;

SELECT \* FROM v\_kunden\_umsaetze;

1. **Durchführung einer Transaktion, um den Lagerbestand eines Artikels zu reduzieren und einen Verkauf zu protokollieren.**

START TRANSACTION;

UPDATE artikel\_lieferant

SET lagerbestand = lagerbestand - 2

WHERE artikel\_lieferant\_id = 1;

INSERT INTO verkauf (kunden\_id, datum, betrag, status)

VALUES (1, CURDATE(), 20.00, 'Offen');

INSERT INTO verkauf\_details (verkauf\_id, artikel\_lieferant\_id, menge, preis)

VALUES **((SELECT MAX(verkauf\_id) FROM verkauf), 1, 2, 10.00);**

**COMMIT;**

1. **Durchführung einer Transaktion, um einen neuen Kunden und einen neuen Verkauf hinzuzufügen**

START TRANSACTION;

insert into kunden(vorname, nachname, straße, hausnummer, postleitzahl, stadt, telefonnummer, email)

values ('Khatarina', 'Argarate', 'Musterstraße', '10', '1010', 'Graz', '0123456789', 'katarina123@hotmail.com');

insert into verkauf (kunden\_id, datum, betrag, status)

values(LAST\_INSERT\_ID(), CURDATE(), 52.75, 'Offen');

COMMIT;

1. **Durchführung einer Transaktion, um den Preis eines Artikels zu ändern und den Preis in allen zugehörigen Verkäufen zu aktualisieren**

start transaction;

update artikel\_lieferant

set preis = 6.32

where artikel\_lieferant\_id = 1;

update verkauf\_details

set preis = 6.32

where artikel\_lieferant\_id = 1;

commit;

**Überlegen Sie auch wo Sie sinnvoll einen oder mehrere Indizes verwenden sollten um die Abfrage zu beschleunigen.**

**1- Index auf Datum in der Verkaufstabelle**

**-So melden monatliche Umsätze (GROUP BY DATE\_FORMAT(datum, '%Y-%m'))**

**-Zum Filtern von Käufen innerhalb eines Zeitraums (WHERE datum BETWEEN ...)**

**CREATE INDEX index\_verkauf\_datum ON verkauf(datum)**

**2- Zusammengesetzter Index für kunden\_id und datum in der Tabelle „verkauf“**

**So suchen nach Käufen, die ein bestimmter Kunde über einen bestimmten Zeitraum getätigt hat**

**Um die letzten Transaktionen jedes Kunden zu überprüfen**

**Effizienter als zwei separate Indizes für kunden\_id und datum**

**CREATE INDEX index\_kunde\_datum ON verkauf(kunden\_id, datum);**

**3- Index für E-Mails in der Kundentabelle (um doppelte E-Mails zu vermeiden)**

**Schnelle Überprüfung auf doppelte E-Mails bei der Registrierung eines neuen Kunden (INSERT INTO customer)**

**Verhindern Sie, dass doppelte E-Mails in die Datenbank gelangen**

**Beschleunigen Sie Abfragen, die per E-Mail nach Kunden suchen**

**CREATE UNIQUE INDEX index\_email ON kunden(email);**

1. **Kombinierter Index auf artikel\_id und lieferanten\_id auf artikel\_lieferant**

**Für die schnelle Verbindung zwischen Artikel und Lieferanten**

**Bei der Durchführung eines JOIN auf artikel\_id und lieferanten\_id**

**Steigerung der Effizienz von Suchvorgängen, die Informationen über Lieferanten eines Produkts abrufen**

**CREATE INDEX index\_artikel\_lieferant ON artikel\_lieferant(artikel\_id, lieferanten\_id);**