**KMS+\_02\_LE\_12\_01**

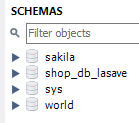
1. Einleitung

*Dieses Dokument beschreibt die Erstellung und Verwaltung einer Shop-Datenbank, inklusive Tabellen, Daten, Abfragen, Views, Transaktionen und Indizes.*

2. Datenbanken, Tabellen und Abfragen

* **Datenbank erstellen und verwenden:**

CREATE DATABASE Shop\_DB\_Lasave;



* **Wechsel zur Datenbank 'Shop\_DB\_Lasave':**

USE Shop\_DB\_Lasave;

* **Erstellung der Tabelle 'Kunden':**

CREATE TABLE Kunden (

kunden\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

vorname VARCHAR(50),

nachname VARCHAR(50),

strasse VARCHAR(50),

hausnummer VARCHAR(10),

postleitzahl VARCHAR(10),

stadt VARCHAR(50),

telefonnummer VARCHAR(20),

email VARCHAR(50)

);

* **Erstellung der Tabelle 'Lieferanten':**

CREATE TABLE Lieferanten (

lieferanten\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(50),

strasse VARCHAR(50),

hausnummer VARCHAR(10),

postleitzahl VARCHAR(10),

stadt VARCHAR(50),

telefonnummer VARCHAR(20),

email VARCHAR(50)

);

* **Erstellung der Tabelle 'Artikel':**

CREATE TABLE Artikel (

artikel\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

bezeichnung VARCHAR(100),

beschreibung TEXT,

preis DECIMAL(10,2),

lagerbestand INT

);

* **Hinzufügung von zwei Kunden aus Argentinien in der Tabelle 'Kunden':**

INSERT INTO Kunden (vorname, nachname, strasse, hausnummer, postleitzahl, stadt, telefonnummer, email)

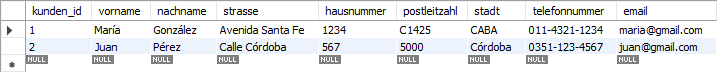
VALUES

('María', 'González', 'Avenida Santa Fe', '1234', 'C1425', 'CABA', '011-4321-1234', 'maria@gmail.com'),

('Juan', 'Pérez', 'Calle Córdoba', '567', '5000', 'Córdoba', '0351-123-4567', 'juan@gmail.com');

* **Überprüfung aller Datensätze in der Tabelle „Kunden“:**

SELECT \* FROM Kunden;



* **Einfügung von zwei Lieferanten aus Argentinien in die Tabelle „Lieferanten“:**

INSERT INTO Lieferanten (name, strasse, hausnummer, postleitzahl, stadt, telefonnummer, email)

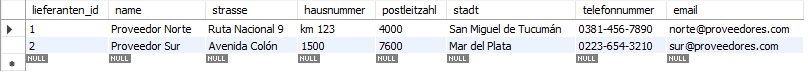
VALUES

('Proveedor Norte', 'Ruta Nacional 9', 'km 123', '4000', 'San Miguel de Tucumán', '0381-456-7890', 'norte@proveedores.com'),

('Proveedor Sur', 'Avenida Colón', '1500', '7600', 'Mar del Plata', '0223-654-3210', 'sur@proveedores.com');

* **Überprüfung aller Datensätze in der Tabelle „Lieferanten“:**

SELECT \* FROM Lieferanten;



* **Einfügung von drei Artikeln aus Argentinien in die Tabelle „Artikel“:**

INSERT INTO Artikel (bezeichnung, beschreibung, preis, lagerbestand)

VALUES

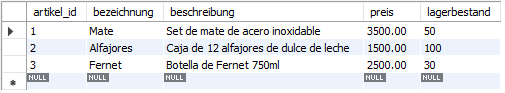
('Mate', 'Set de mate de acero inoxidable', 3500.00, 50),

('Alfajores', 'Caja de 12 alfajores de dulce de leche', 1500.00, 100),

('Ferné', 'Botella de Fernet 750ml', 2500.00, 30);

* **Überprüfung aller Datensätze in der Tabelle „Artikel“:**

SELECT \* FROM Artikel;



* **Erstellung der Tabelle 'Verkauf':**

CREATE TABLE Verkauf (

verkauf\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

kunden\_id INT,

artikel\_id INT,

menge INT,

datum DATE,

FOREIGN KEY (kunden\_id) REFERENCES Kunden(kunden\_id),

FOREIGN KEY (artikel\_id) REFERENCES Artikel(artikel\_id)

);

* **Hinzufügen von Verkäufen mit argentinischen Kunden und Artikeln:**

INSERT INTO Verkauf (kunden\_id, artikel\_id, menge, datum)

VALUES

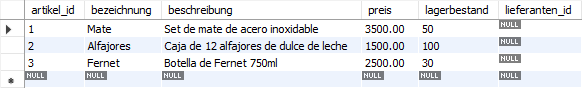
(1, 1, 2, '2024-05-10'), -- María kauft 2 Mates

(2, 2, 1, '2024-05-11'), -- Juan kauft 1 caja de Alfajores

(1, 3, 1, '2024-05-12'); -- María kauft 1 Fernet

* **Hinzufügung der Spalte „lieferanten\_id“ in der Tabelle „Artikel“:**

ALTER TABLE Artikel ADD lieferanten\_id INT;



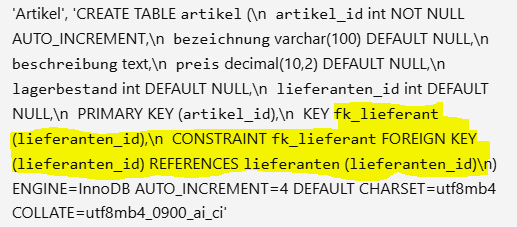
* **Definition des Fremdschlüssels „lieferanten\_id“, der auf „Lieferanten(lieferanten\_id)“ verweist:**

ALTER TABLE Artikel ADD CONSTRAINT fk\_lieferant

FOREIGN KEY (lieferanten\_id) REFERENCES Lieferanten(lieferanten\_id);

* **Überprüfung aller Fremdschlüssel der Tabelle „Artikel“:**

SHOW CREATE TABLE Artikel;



**Kommentar:**

* Die Tabelle Artikel hat einen Fremdschlüssel namens fk\_lieferant.
* Dieser Schlüssel verbindet die Spalte lieferanten\_id der Tabelle Artikel mit der Spalte lieferanten\_id der Tabelle Lieferanten.
* **Zuordnung der Artikel zu den Lieferanten:**

**Beispiele:**

* **Mate gehört zu Lieferant 1:**

UPDATE Artikel SET lieferanten\_id = 1 WHERE artikel\_id = 1;

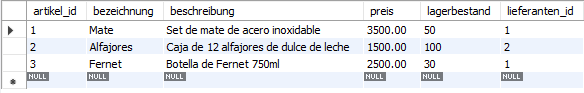
* **Alfajores gehören zu Lieferant 2:**

UPDATE Artikel SET lieferanten\_id = 2 WHERE artikel\_id = 2;

* **Fernet gehört zu Lieferant 1:**

UPDATE Artikel SET lieferanten\_id = 1 WHERE artikel\_id = 3;

* **Überprüfung der Tabelle „Artikel“, in der die Artikel mit ihren aktuellen Preisen, Lagerbeständen und zugeordneten Lieferanten angezeigt werden:**



**Kommentar:**

Nach der Aktualisierung enthält die Spalte lieferanten\_id die ID des entsprechenden Lieferanten.

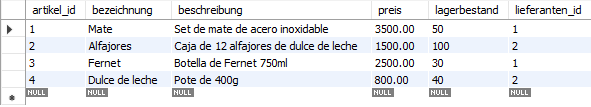
* **Erweiterung des Produktkatalogs in der Tabelle „Artikel“:**

INSERT INTO Artikel (bezeichnung, beschreibung, preis, lagerbestand, lieferanten\_id)

VALUES ('Dulce de leche', 'Pote de 400g', 800.00, 40, 2);

* **Überwachung aller Datensätze in der Tabelle „Artikel“:**

SELECT \* FROM Artikel;



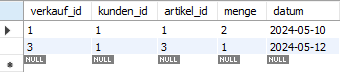
* **Bestandsprüfung eines Artikels:**

SELECT lagerbestand FROM Artikel WHERE artikel\_id = 1;



* **Kunden-Verkaufscheck:**

SELECT \* FROM Verkauf WHERE kunden\_id = 1;



* **Informationen über den Verkauf von Produkten eines bestimmten Lieferanten einholen:**

SELECT Verkauf.\*

FROM Verkauf

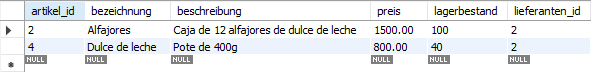
JOIN Artikel ON Verkauf.artikel\_id = Artikel.artikel\_id

WHERE Artikel.lieferanten\_id = 2;



* **Abfrage von Informationen über Produkte unter einem bestimmten Preis:**

SELECT \* FROM Artikel WHERE preis < 2000;



* **Berechnung der Gesamtumsätze, die von allen registrierten Verkäufen erzielt werden:**

SELECT SUM(Artikel.preis \* Verkauf.menge) AS umsatz

FROM Verkauf

JOIN Artikel ON Verkauf.artikel\_id = Artikel.artikel\_id;



3. Erstellen und Verwenden von Views

* **Löschung der View 'KundenVerkaeufe', falls diese bereits existiert:**

DROP VIEW IF EXISTS KundenVerkaeufe;

* **Erstellung einer View, die Kunden, ihre gekauften Artikel, Menge und Kaufdatum zusammenfasst:**

CREATE VIEW KundenVerkaeufe AS

SELECT Kunden.vorname, Kunden.nachname, Artikel.bezeichnung, Verkauf.menge, Verkauf.datum

FROM Verkauf

JOIN Kunden ON Verkauf.kunden\_id = Kunden.kunden\_id

JOIN Artikel ON Verkauf.artikel\_id = Artikel.artikel\_id;

**Kommentar:**

Diese View erleichtert die Berichterstattung und Analyse, ohne dass komplexe JOIN-Abfragen geschrieben werden müssen.

* **Überprüfung der Daten in der View 'KundenVerkaeufe':**

SELECT \* FROM KundenVerkaeufe;



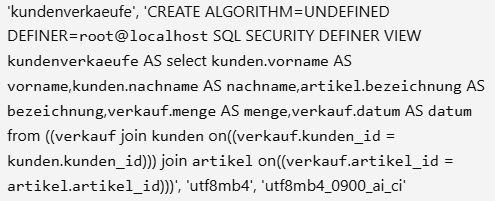
* **Auflistung aller Views in der aktuellen Datenbank:**

SHOW FULL TABLES WHERE Table\_type = 'VIEW';



* **Anzeigen des SQL-Codes zur Erstellung der View:**

SHOW CREATE VIEW KundenVerkaeufe;



**Kommentar:**

Dies ist der Code, den ich geschrieben und als View gespeichert habe.

4. Transaktionen

* **Abfrage des Lagerbestands eines Artikels:**

SELECT lagerbestand FROM Artikel WHERE artikel\_id = 1;



* **Start einer Transaktion zur Reduzierung des Lagerbestands:**

START TRANSACTION;

UPDATE Artikel SET lagerbestand = lagerbestand - 1 WHERE artikel\_id = 1;

INSERT INTO Verkauf (kunden\_id, artikel\_id, menge, datum) VALUES (1, 1, 1, CURDATE());

COMMIT;

* **Überprüfung des aktualisierten Lagerbestands:**

SELECT lagerbestand FROM Artikel WHERE artikel\_id = 1;

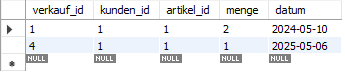


**Kommentar:**

Ich habe festgestellt, dass der Lagerbestand um 1 reduziert worden ist.

* **Überprüfung der erfassten Verkäufe für Artikel ID 1:**

SELECT \* FROM Verkauf WHERE artikel\_id = 1;



**Kommentar:**

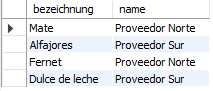
Ich habe überprüft, dass der neue Verkauf mit Datum = CURDATE() eingefügt wurde.

* **Anzeige aller Artikel mit ihren Lieferanten:**

SELECT Artikel.bezeichnung, Lieferanten.name

FROM Artikel

JOIN Lieferanten ON Artikel.lieferanten\_id = Lieferanten.lieferanten\_id;



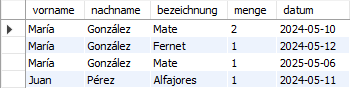
* **Anzeige aller Verkäufe mit Kunden- und Artikeldetails:**

SELECT Kunden.vorname, Kunden.nachname, Artikel.bezeichnung, Verkauf.menge, Verkauf.datum

FROM Verkauf

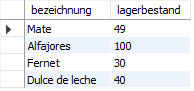
JOIN Kunden ON Verkauf.kunden\_id = Kunden.kunden\_id

JOIN Artikel ON Verkauf.artikel\_id = Artikel.artikel\_id;



* **Anzeige des Lagerbestands aller Artikel:**

SELECT bezeichnung, lagerbestand FROM Artikel;



* **Berechnung des Gesamtumsatzes pro Kunde:**

SELECT Kunden.vorname, Kunden.nachname, SUM(Artikel.preis \* Verkauf.menge) AS umsatz

FROM Verkauf

JOIN Kunden ON Verkauf.kunden\_id = Kunden.kunden\_id

JOIN Artikel ON Verkauf.artikel\_id = Artikel.artikel\_id

GROUP BY Kunden.kunden\_id;



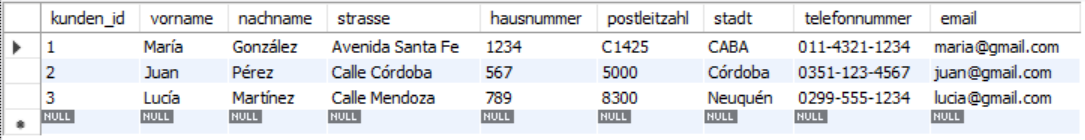
* **Hinzufügen eines neuen Kunden mit gleichzeitiger Verkaufserfassung (Transaktion)**

START TRANSACTION;

* **Einfügen eines neuen Kunden in die Tabelle 'Kunden':**

INSERT INTO Kunden (vorname, nachname, strasse, hausnummer, postleitzahl, stadt, telefonnummer, email)

VALUES ('Lucía', 'Martínez', 'Calle Mendoza', '789', '8300', 'Neuquén', '0299-555-1234', 'lucia@gmail.com');



* **Abrufen der ID des neu eingefügten Kunden:**

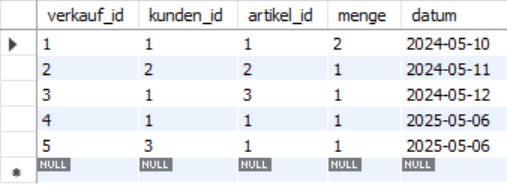
SET @neuer\_kunde\_id = LAST\_INSERT\_ID();

* **Einfügen eines Verkaufs für den neuen Kunden und Abschluss der Transaktion:**

INSERT INTO Verkauf (kunden\_id, artikel\_id, menge, datum)

VALUES (@neuer\_kunde\_id, 1, 1, CURDATE());

COMMIT;



* **Ändern des Artikelpreises und Vorbereitung der Aktualisierung der Verkäufe:**

START TRANSACTION;

* **Bisheriger Preis des Artikels 1:**

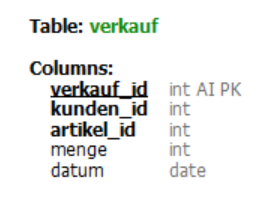


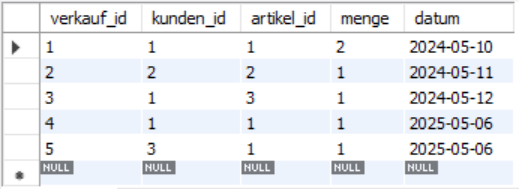
* **Aktualisierung des Artikelpreises:**

UPDATE Artikel SET preis = 4000.00 WHERE artikel\_id = 1;



* **Überprüfung der Tabelle Verkauf vor Hinzufügen der Spalte „verkaufspreis“:**

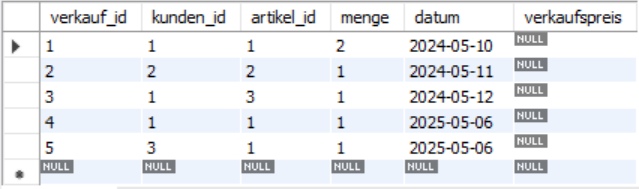




* **Überprüfung der Tabelle Verkauf nach Hinzufügen der Spalte „verkaufspreis“:**

ALTER TABLE Verkauf ADD COLUMN verkaufspreis DECIMAL(10,2);





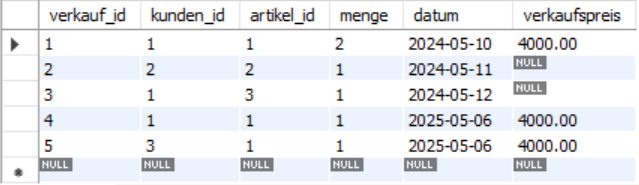
* **Aktualisierung des Verkaufspreises in allen bisherigen Verkäufen:**

UPDATE Verkauf

SET verkaufspreis = (SELECT preis FROM Artikel WHERE Artikel.artikel\_id = Verkauf.artikel\_id)

WHERE artikel\_id = 1;

COMMIT;

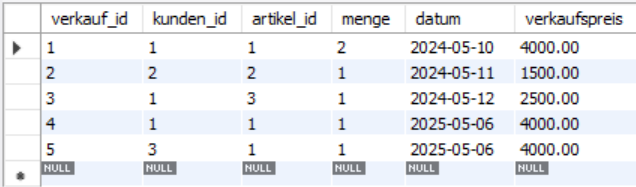


* **Aktualisierung des Verkaufspreises nur für Datensätze mit NULL-Wert:**

UPDATE Verkauf

SET verkaufspreis = (SELECT preis FROM Artikel WHERE Artikel.artikel\_id = Verkauf.artikel\_id)

WHERE verkaufspreis IS NULL;



5. Indizes und Abfrageoptimierung

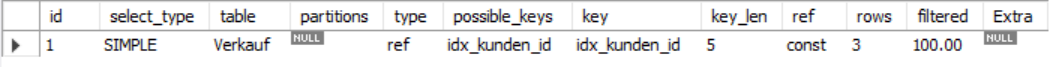
CREATE INDEX idx\_kunden\_id ON Verkauf(kunden\_id);

CREATE INDEX idx\_lieferanten\_id ON Artikel(lieferanten\_id);

CREATE INDEX idx\_artikel\_id ON Verkauf(artikel\_id);

* **Überprüfung des Indexes „idx\_kunden\_id“ mit EXPLAIN:**

EXPLAIN SELECT \* FROM Verkauf WHERE kunden\_id = 1;



**Kommentar:**

Um alle Verkäufe zu finden, bei denen der Kunde die Nummer 1 ist, wurde der Index idx\_kunden\_id verwendet.

* **Überprüfung des Indexes „idx\_lieferanten\_id“ mit EXPLAIN:**

EXPLAIN SELECT \* FROM Artikel WHERE lieferanten\_id = 2;



**Kommentar:**

In der Spalte key erscheint idx\_lieferanten\_id. Das bedeutet, dass das Programm den Index verwendet, den ich über die Spalte lieferanten\_id erstellt habe.

* **Überprüfung des Indexes „idx\_artikel\_id“ mit EXPLAIN:**

EXPLAIN SELECT \* FROM Verkauf WHERE artikel\_id = 1;

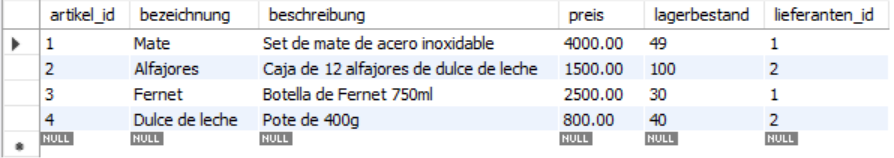


**Kommentar:**

Die Spalte key zeigt idx\_artikel\_id. Das bedeutet, dass MySQL den Index verwendet, den ich für die Spalte artikel\_id erstellt habe.

**Zusätzliche Hinweise:**

* Indizes beschleunigen Abfragen, indem sie die Suche nach bestimmten Datensätzen vereinfachen.
* Ohne Indizes müsste die Datenbank jeden Datensatz durchsuchen (Full Table Scan), was bei großen Datenmengen ineffizient wäre.
* **Start einer Transaktion zur Reduzierung des Lagerbestands und Protokollierung des Verkaufs.**
* Anzeige des aktuellen Lagerbestands und Preises vor der Transaktion:



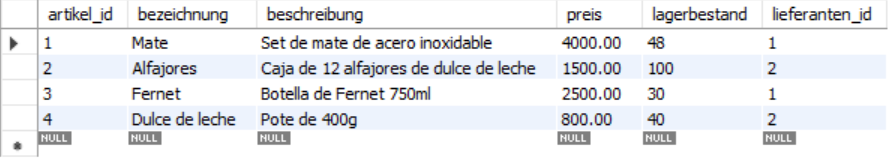
START TRANSACTION;

UPDATE Artikel SET lagerbestand = lagerbestand - 1 WHERE artikel\_id = 1;

INSERT INTO Verkauf (kunden\_id, artikel\_id, menge, datum) VALUES (1, 1, 1, CURDATE());

COMMIT;

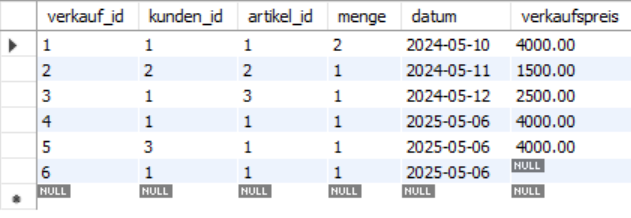
* Überprüfung des Lagerbestands nach der Transaktion:



**Kommentar:**

In der Tat wurde der Bestand des gekauften Artikels nach der Transaktion um 1 (auf 48) reduziert.

* **Überprüfung der Verkaufstabelle nach der Transaktion:**



**Kommentar:**

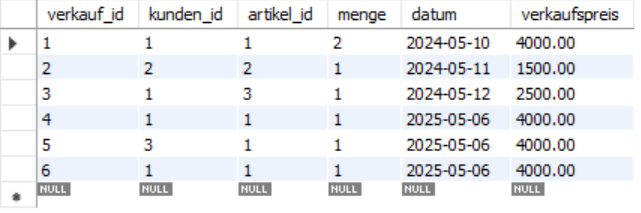
Die Verkaufstabelle zeigt nun alle Verkäufe, einschließlich der aktualisierten Verkaufspreise. Alle Einträge sind konsistent und vollständig.

* **Endgültige Aktualisierung der Verkaufspreise für alle Verkäufe:**

UPDATE Verkauf

SET verkaufspreis = (SELECT preis FROM Artikel WHERE Artikel.artikel\_id = Verkauf.artikel\_id)

WHERE verkaufspreis IS NULL;



**Kommentar:**

* Diese Tabelle zeigt alle Verkäufe mit aktualisierten Verkaufspreisen.
* Der zuletzt hinzugefügte Verkauf (verkauf\_id = 6) wurde erfolgreich ergänzt und enthält jetzt den korrekten Verkaufspreis.
* Alle Datensätze sind nun vollständig und konsistent.

6. Zusammenfassung:

In diesem Projekt wurde eine Datenbank für einen Shop entwickelt.

Sie enthält vier Tabellen: Kunden, Lieferanten, Artikel und Verkauf.

Mithilfe von Fremdschlüsseln wurden die Tabellen verknüpft.

Es wurden Daten aus Argentinien eingefügt und verschiedene SQL-Abfragen, Views und Transaktionen erstellt, um die Funktionalität zu demonstrieren.

**Klarstellung:**

Die Tabelle 'Artikel' wurde zunächst ohne Fremdschlüssel erstellt. Später habe ich mit ALTER TABLE die Spalte 'lieferanten\_id' und die Fremdschlüsselbeziehung hinzugefügt. Dadurch konnte ich den Prozess Schritt für Schritt dokumentieren, wie in den Screenshots zu sehen ist.

**Ende des Arbeitsauftrags**

**--**