



Technische Datenbankdokumentation für Auftragsmanagementsystem

Dersim Kaya

6. Oktober 2025

Inhaltsverzeichnis

| | |
|---|-----------|
| 1 Einführung | 2 |
| 1.1 Projektkonzept | 2 |
| 1.2 Zielsetzung | 2 |
| 2 Datenbankstruktur | 3 |
| 2.1 ER-Modell | 3 |
| 2.2 Entitäten und Attribute | 4 |
| 3 SQL Abfragen | 12 |
| 3.1 Erstellung der Tabellen | 12 |
| 3.2 Testen von Constraints | 17 |
| 3.3 Indexe | 20 |
| 3.4 Prozeduren | 21 |
| 3.4.1 Prozedur zur Rabattberechnung | 21 |
| 3.4.2 Prozedur zur Überprüfung der Verfügbarkeit und Eintragung eines Auftrags | 23 |
| 3.5 Trigger | 26 |
| 4 Zusammenfassung | 29 |

1 Einführung

1.1 Projektkonzept

Dieses Projekt dokumentiert die relationale Datenbank des Auftragsmanagementsystems. Die Dokumentation beschreibt das Datenmodell, die wichtigsten Tabellen und Beziehungen, vorhandene Stored Procedures und Triggers sowie typische Abläufe wie Bestellverarbeitung, Rechnungsstellung und Rabattberechnung. Ziel ist es, Entwicklern, Administratoren und Fachanwendern eine zentrale, verständliche Quelle für Betrieb, Weiterentwicklung und Fehlersuche bereitzustellen.

1.2 Zielsetzung

Die Zielsetzung dieser Dokumentation umfasst die folgenden Punkte:

- **Bereitstellung einer technischen Referenz:** Vollständige Beschreibung der Datenbankstruktur inklusive Tabellen, Spalten, Datentypen, Defaults und Constraints.
- **Erklärung fachlicher Abläufe:** Darstellung der Geschäftsregeln, wie Statuswechsel, automatische Rechnungserstellung und Rabattberechnung.
- **Tests und Beispielskripte:** Anleitungen und Beispielabfragen, um Integrität und erwartetes Verhalten nach Änderungen zu verifizieren.
- **Effiziente Auftragsverwaltung:** Erstellung und Nachverfolgung von Aufträgen.
- **Lagerbestandsverwaltung:** Vermeidung von Fehlbeständen durch Überwachung der Lagerbestände.
- **Automatische Rechnungsstellung:** Vereinfachung der Zahlungsabwicklung durch generierte Rechnungen.
- **Zukunftssicherheit:** Eine flexible Struktur, die erweiterbar ist, um zukünftigen Anforderungen gerecht zu werden.

2 Datenbankstruktur

Die Datenbank besteht aus mehreren Tabellen, die miteinander verknüpft sind, um die referenzielle Integrität zu gewährleisten.

2.1 ER-Modell

Das folgende ER-Modell beschreibt die logische Struktur der Datenbank für das Auftragsmanagement. Es visualisiert die zentralen Entitäten, deren Primärschlüssel, die relevanten Fremdschlüsselbeziehungen sowie die Kardinalitäten zwischen den Entitäten. Das Diagramm dient als Referenz für Architekturentscheidungen und Testszenarien.

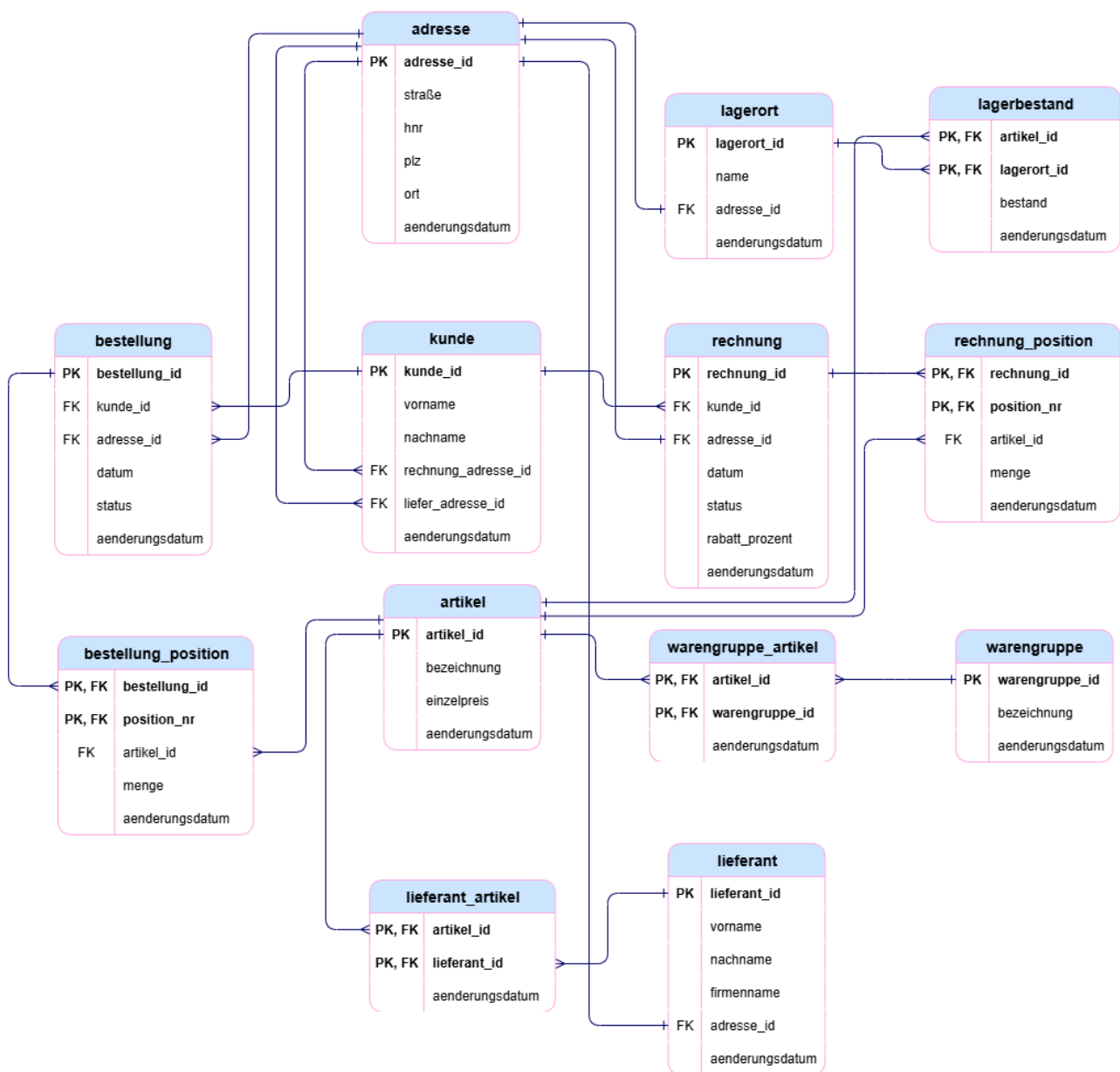


Abbildung 1: ER-Modell

2.2 Entitäten und Attribute

In diesem Abschnitt werden die einzelnen Tabellen der Auftragsmanagement-Datenbank detailliert beschrieben. Jede Tabelle wird mit ihren jeweiligen Spalten aufgeführt, einschließlich Datentypen, Einschränkungen und einer kurzen Beschreibung der Funktionalität der Spalten.

Tabelle 1: Entität adresse

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|--------------|---------------------------------------|----------------------------------|
| adresse_id | SERIAL | PRIMARY KEY | Eindeutige ID für jede Adresse |
| strasse | VARCHAR(100) | NOT NULL | Straßenname der Adresse |
| haus_nr | VARCHAR(10) | NOT NULL | Hausnummer oder Zusatz |
| plz | VARCHAR(10) | NOT NULL | Postleitzahl |
| ort | VARCHAR(100) | NOT NULL | Ortsname / Stadt |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 2: Entität kunde

| Attribut | Datentyp | Constraints | Beschreibung |
|---------------------|--------------|--|--|
| kunde_id | SERIAL | PRIMARY KEY | Eindeutige ID für jeden Kunden |
| vorname | VARCHAR(100) | NOT NULL | Vorname des Kunden |
| nachname | VARCHAR(100) | NOT NULL | Nachname des Kunden |
| rechnung_adresse_id | INT | NOT NULL REFERENCES adresse(adresse_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Rechnungs- adresse (FK) |

Continued on next page

Tabelle 2: Entität kunde (Continued)

| | | | |
|-------------------|-----------|--|-------------------------------------|
| liefer_adresse_id | INT | NOT NULL REFERENCES adresse(adresse_id) ON UPDATE CAS- CADE ON DELETE RE- STRICT | Verweis auf Lieferadresse (FK) |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CUR- RENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 3: Entität bestellung

| Attribut | Datentyp | Constraints | Beschreibung |
|---------------|-------------|--|---|
| bestellung_id | SERIAL | PRIMARY KEY | Eindeutige ID für jede Be- stellung |
| kunde_id | INT | NOT NULL REFERENCES kun- de(kunde_id) ON UPDATE CAS- CADE ON DELETE RE- STRICT | Verweis auf den Kunden (FK) |
| adresse_id | INT | NOT NULL REFERENCES adresse(adresse_id) ON UPDATE CAS- CADE ON DELETE RE- STRICT | Verweis auf Lieferadresse (FK) |
| status | VARCHAR(50) | NOT NULL | Bestellstatus (z. B. 'offen', 'versandt', 'storniert') |
| datum | DATE | NOT NULL DEFAULT CUR- RENT_DATE | Bestelldatum |

Continued on next page

Tabelle 3: Entität bestellung (Continued)

| | | | |
|-----------------|-----------|---------------------------------------|----------------------------------|
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |
|-----------------|-----------|---------------------------------------|----------------------------------|

Tabelle 4: Entität rechnung

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|--------------|---|---|
| rechnung_id | SERIAL | PRIMARY KEY | Eindeutige ID für jede Rechnung |
| kunde_id | INT | NOT NULL REFERENCES kunde(kunde_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Kunden (FK) |
| adresse_id | INT | NOT NULL REFERENCES adresse(adresse_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Rechnungsadresse (FK) |
| datum | DATE | NOT NULL DEFAULT CURRENT_DATE | Rechnungsdatum |
| status | VARCHAR(50) | NOT NULL | Rechnungsstatus (z. B. 'offen', 'bezahlt', 'storniert') |
| rabatt_prozent | NUMERIC(5,2) | DEFAULT 0 | Rabatt in Prozent |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 5: Entität artikel

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|---------------|---------------------------------------|----------------------------------|
| artikel_id | SERIAL | PRIMARY KEY | Eindeutige ID für jeden Artikel |
| bezeichnung | VARCHAR(200) | NOT NULL | Artikelbezeichnung |
| einzelpreis | NUMERIC(10,2) | NOT NULL | Preis pro Stück |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 6: Entität rechnung_position

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|-----------|--|---|
| rechnung_id | INT | NOT NULL REFERENCES rechnung(rechnung_id) ON UPDATE CASCADE ON DELETE CASCADE | Verweis auf Rechnung (FK), Teil des PK |
| position_nr | INT | NOT NULL | Positionsnummer innerhalb der Rechnung, Teil des PK |
| artikel_id | INT | NOT NULL REFERENCES artikel(artikel_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Artikel (FK) |
| menge | INT | NOT NULL | Anzahl der Artikel in dieser Position |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 7: Entität lagerort

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|--------------|--|----------------------------------|
| lagerort_id | SERIAL | PRIMARY KEY | Eindeutige ID für jeden Lagerort |
| name | VARCHAR(200) | NOT NULL | Bezeichnung des Lagerorts |
| adresse_id | INT | NOT NULL REFERENCES adresse(adresse_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Adresse (FK) |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 8: Entität lagerbestand

| Attribut | Datentyp | Constraints | Beschreibung |
|-------------|----------|---|--|
| artikel_id | INT | NOT NULL REFERENCES artikel(artikel_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Artikel (FK), Teil des PK |
| lagerort_id | INT | NOT NULL REFERENCES lagerort(lagerort_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Lagerort (FK), Teil des PK |
| bestand | INT | NOT NULL | Aktueller Bestand am Lagerort |

Continued on next page

Tabelle 8: Entität lagerbestand (Continued)

| | | | |
|-----------------|-----------|---------------------------------------|----------------------------------|
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |
|-----------------|-----------|---------------------------------------|----------------------------------|

Tabelle 9: Entität bestellung_position

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|-----------|--|---|
| bestellung_id | INT | NOT NULL REFERENCES bestellung(bestellung_id) ON UPDATE CASCADE ON DELETE CASCADE | Verweis auf Bestellung (FK), Teil des PK |
| position_nr | INT | NOT NULL | Positionsnummer innerhalb der Bestellung, Teil des PK |
| artikel_id | INT | NOT NULL REFERENCES artikel(artikel_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Artikel (FK) |
| menge | INT | NOT NULL | Bestellmenge dieser Position |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 10: Entität warengruppe

| Attribut | Datentyp | Constraints | Beschreibung |
|----------------|----------|-------------|------------------------------------|
| warengruppe_id | SERIAL | PRIMARY KEY | Eindeutige ID für jede Warengruppe |

Continued on next page

Tabelle 10: Entität warengruppe (Continued)

| | | | |
|-----------------|--------------|---------------------------------------|----------------------------------|
| bezeichnung | VARCHAR(200) | NOT NULL | Name der Warengruppe |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 11: Entität warengruppe_artikel

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|-----------|--|---|
| artikel_id | INT | NOT NULL REFERENCES artikel(artikel_id) ON UPDATE CASCADE ON DELETE CASCADE | Verweis auf Artikel (FK), Teil des PK |
| warengruppe_id | INT | NOT NULL REFERENCES warengruppe(warengruppe_id) ON UPDATE CASCADE ON DELETE CASCADE | Verweis auf Warengruppe (FK), Teil des PK |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 12: Entität lieferant

| Attribut | Datentyp | Constraints | Beschreibung |
|--------------|--------------|-------------|-------------------------------------|
| lieferant_id | SERIAL | PRIMARY KEY | Eindeutige ID für jeden Lieferanten |
| vorname | VARCHAR(100) | | Optionaler Vorname |
| nachname | VARCHAR(100) | | Optionaler Nachname |

Continued on next page

Tabelle 12: Entität lieferant (Continued)

| | | | |
|-----------------|--------------|--|----------------------------------|
| firmenname | VARCHAR(200) | NOT NULL | Firmenname des Lieferanten |
| adresse_id | INT | NOT NULL REFERENCES adresse(adresse_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Adresse (FK) |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

Tabelle 13: Entität lieferant_artikel

| Attribut | Datentyp | Constraints | Beschreibung |
|-----------------|-----------|--|---|
| artikel_id | INT | NOT NULL REFERENCES artikel(artikel_id) ON UPDATE CASCADE ON DELETE RESTRICT | Verweis auf Artikel (FK), Teil des PK |
| lieferant_id | INT | NOT NULL REFERENCES lieferant(lieferant_id) ON UPDATE CASCADE ON DELETE CASCADE | Verweis auf Lieferant (FK), Teil des PK |
| aenderungsdatum | TIMESTAMP | NOT NULL DEFAULT CURRENT_TIMESTAMP | Zeitstempel der letzten Änderung |

3 SQL Abfragen

3.1 Erstellung der Tabellen

Dieser Abschnitt enthält die SQL Anweisungen zum Anlegen aller Tabellen des Datenmodells inklusive Primärschlüssel, Fremdschlüssel, Defaultwerte und Constraints.

Listing 1: adresse

```
CREATE TABLE adresse (  
    adresse_id SERIAL PRIMARY KEY,  
    strasse VARCHAR(100) NOT NULL,  
    haus_nr VARCHAR(10) NOT NULL,  
    plz VARCHAR(10) NOT NULL,  
    ort VARCHAR(100) NOT NULL,  
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP  
);
```

Listing 2: kunde

```
CREATE TABLE kunde (  
    kunde_id SERIAL PRIMARY KEY,  
    vorname VARCHAR(100) NOT NULL,  
    nachname VARCHAR(100) NOT NULL,  
    rechnung_adresse_id INT NOT NULL,  
    liefer_adresse_id INT NOT NULL,  
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    CONSTRAINT fk_kunde_rechnung_adresse FOREIGN KEY(  
        rechnung_adresse_id  
        REFERENCES adresse(adresse_id)  
        ON UPDATE CASCADE  
        ON DELETE RESTRICT,  
    CONSTRAINT fk_kunde_liefer_adresse FOREIGN KEY(liefer_adresse_id)  
        REFERENCES adresse(adresse_id)  
        ON UPDATE CASCADE  
        ON DELETE RESTRICT  
);
```

Listing 3: bestellung

```
CREATE TABLE bestellung (  
    bestellung_id SERIAL PRIMARY KEY,  
    kunde_id INT NOT NULL,  
    adresse_id INT NOT NULL,
```

```

status VARCHAR(50) NOT NULL,
datum DATE NOT NULL DEFAULT CURRENT_DATE,
aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT fk_bestellung_kunde FOREIGN KEY(kunde_id)
    REFERENCES kunde(kunde_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT,
CONSTRAINT fk_bestellung_adresse FOREIGN KEY(adresse_id)
    REFERENCES adresse(adresse_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
);

```

Listing 4: rechnung

```

CREATE TABLE rechnung (
    rechnung_id SERIAL PRIMARY KEY,
    kunde_id INT NOT NULL,
    adresse_id INT NOT NULL,
    datum DATE NOT NULL DEFAULT CURRENT_DATE,
    status VARCHAR(50) NOT NULL,
    rabatt_prozent NUMERIC(5,2) DEFAULT 0,
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_rechnung_kunde FOREIGN KEY (kunde_id)
        REFERENCES kunde(kunde_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT fk_rechnung_adresse FOREIGN KEY (adresse_id)
        REFERENCES adresse(adresse_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

```

Listing 5: artikel

```

CREATE TABLE artikel (
    artikel_id SERIAL PRIMARY KEY,
    bezeichnung VARCHAR(200) NOT NULL,
    einzelpreis NUMERIC(10,2) NOT NULL,
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

```

Listing 6: rechnung_position

```
CREATE TABLE rechnung_position (  
    rechnung_id INT NOT NULL,  
    position_nr INT NOT NULL,  
    artikel_id INT NOT NULL,  
    menge INT NOT NULL,  
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    CONSTRAINT pk_rechnung_position PRIMARY KEY (rechnung_id,  
        position_nr),  
    CONSTRAINT fk_rechnung_position_rechnung FOREIGN KEY (rechnung_id)  
        REFERENCES rechnung(rechnung_id)  
        ON UPDATE CASCADE  
        ON DELETE CASCADE,  
    CONSTRAINT fk_rechnung_position_artikel FOREIGN KEY (artikel_id)  
        REFERENCES artikel(artikel_id)  
        ON UPDATE CASCADE  
        ON DELETE RESTRICT  
);
```

Listing 7: lagerort

```
CREATE TABLE lagerort (  
    lagerort_id SERIAL PRIMARY KEY,  
    name VARCHAR(200) NOT NULL,  
    adresse_id INT NOT NULL,  
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    CONSTRAINT fk_lagerort_adresse FOREIGN KEY (adresse_id)  
        REFERENCES adresse(adresse_id)  
        ON UPDATE CASCADE  
        ON DELETE RESTRICT  
);
```

Listing 8: lagerbestand

```
CREATE TABLE lagerbestand (  
    artikel_id INT NOT NULL,  
    lagerort_id INT NOT NULL,  
    bestand INT NOT NULL,  
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    CONSTRAINT pk_lagerbestand PRIMARY KEY (artikel_id, lagerort_id),  
    CONSTRAINT fk_lagerbestand_artikel FOREIGN KEY (artikel_id)  
        REFERENCES artikel(artikel_id)  
        ON UPDATE CASCADE
```

```

        ON DELETE RESTRICT,
    CONSTRAINT fk_lagerbestand_lagerort FOREIGN KEY (lagerort_id)
        REFERENCES lagerort(lagerort_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

```

Listing 9: bestellung_position

```

CREATE TABLE bestellung_position (
    bestellung_id INT NOT NULL,
    position_nr INT NOT NULL,
    artikel_id INT NOT NULL,
    menge INT NOT NULL,
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_bestellung_position PRIMARY KEY (bestellung_id,
        position_nr),
    CONSTRAINT fk_bestellung_position_bestellung FOREIGN KEY (
        bestellung_id)
        REFERENCES bestellung(bestellung_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT fk_bestellung_position_artikel FOREIGN KEY (artikel_id)
        REFERENCES artikel(artikel_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

```

Listing 10: warengruppe

```

CREATE TABLE warengruppe (
    warengruppe_id SERIAL PRIMARY KEY,
    bezeichnung VARCHAR(200) NOT NULL,
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

```

Listing 11: warengruppe_artikel

```

CREATE TABLE warengruppe_artikel (
    artikel_id INT NOT NULL,
    warengruppe_id INT NOT NULL,
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_warengruppe_artikel PRIMARY KEY (artikel_id,
        warengruppe_id),
);

```



```

        CONSTRAINT fk_warengruppe_artikel_artikel FOREIGN KEY (artikel_id)
            REFERENCES artikel(artikel_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE,
        CONSTRAINT fk_warengruppe_artikel_warengruppe FOREIGN KEY (
            warengruppe_id)
            REFERENCES warengruppe(warengruppe_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
    );

```

Listing 12: lieferant

```

CREATE TABLE lieferant (
    lieferant_id SERIAL PRIMARY KEY,
    vorname VARCHAR(100),
    nachname VARCHAR(100),
    firmenname VARCHAR(200) NOT NULL,
    adresse_id INT NOT NULL,
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_lieferant_adresse FOREIGN KEY (adresse_id)
        REFERENCES adresse(adresse_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

```

Listing 13: lieferant_artikel

```

CREATE TABLE lieferant_artikel (
    artikel_id INT NOT NULL,
    lieferant_id INT NOT NULL,
    aenderungsdatum TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_lieferant_artikel PRIMARY KEY (lieferant_id,
        artikel_id),
    CONSTRAINT fk_lieferant_artikel_artikel FOREIGN KEY (artikel_id)
        REFERENCES artikel(artikel_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT fk_lieferant_artikel_lieferant FOREIGN KEY (
        lieferant_id)
        REFERENCES lieferant(lieferant_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

```

```
);
```

3.2 Testen von Constraints

Dieser Abschnitt beschreibt die geplanten Tests zur Überprüfung der Datenbank-Constraints und erklärt kurz Zweck und Ablauf: Es werden INSERT/UPDATE/DELETE-Beispiele ausgeführt, um Fremdschlüssel, Primärschlüssel (inklusive zusammengesetzter PK), NOT NULL-Beschränkungen, ON DELETE-Verhalten und CHECK-Constraints gezielt zu prüfen; erfolgreiche und absichtlich fehlschlagende Fälle sind dokumentiert, damit man erwartete Fehlermeldungen und die Integritätsfehler nachvollziehen kann.

Listing 14: kunde und adresse

```
INSERT INTO adresse (strasse, haus_nr, plz, ort, aenderungsdatum)
VALUES ('Teststrasse', '1', '60311', 'Frankfurt', CURRENT_TIMESTAMP);

INSERT INTO kunde (vorname, nachname, rechnung_adresse_id,
    liefer_adresse_id, aenderungsdatum)
VALUES ('Max', 'Mustermann', 1, 1, CURRENT_TIMESTAMP);
```

Listing 15: artikel, warengruppe, lieferant

```
INSERT INTO artikel (bezeichnung, einzelpreis, aenderungsdatum)
VALUES ('Widget', 9.99, CURRENT_TIMESTAMP);

INSERT INTO warengruppe (bezeichnung, aenderungsdatum)
VALUES ('Zubehör', CURRENT_TIMESTAMP);

INSERT INTO warengruppe_artikel (artikel_id, warengruppe_id,
    aenderungsdatum)
VALUES (1, 1, CURRENT_TIMESTAMP);

INSERT INTO lieferant (vorname, nachname, firmenname, adresse_id,
    aenderungsdatum)
VALUES ('Erika', 'Beispiel', 'Beispiel GmbH', 1001, CURRENT_TIMESTAMP);

INSERT INTO lieferant_artikel (lieferant_id, artikel_id, aenderungsdatum)
VALUES (1, 1, CURRENT_TIMESTAMP);
```

Listing 16: lagerort und lagerbestand

```
INSERT INTO lagerort (lagerort_id, name, adresse_id, aenderungsdatum)
```

```
VALUES (1, 'Hauptlager Frankfurt', 1, CURRENT_TIMESTAMP);
```

```
INSERT INTO lagerbestand (artikel_id, lagerort_id, bestand,  
    aenderungsdatum)
```

```
VALUES (3001, 6001, 50, CURRENT_TIMESTAMP);
```

Listing 17: bestellung und rechnung

```
INSERT INTO bestellung (kunde_id, adresse_id, status, datum,  
    aenderungsdatum)
```

```
VALUES (1, 1, 'offen', CURRENT_DATE, CURRENT_TIMESTAMP);
```

```
INSERT INTO rechnung (kunde_id, adresse_id, status, datum, aenderungsdatum  
    )
```

```
VALUES (1, 1, 'offen', CURRENT_DATE, CURRENT_TIMESTAMP);
```

Listing 18: Fremdschlüssel-Tests

```
-- Erfolgreiche Positionszeile (FKs existieren)
```

```
INSERT INTO bestellung_position(bestellung_id, position_nr, artikel_id,  
    menge, aenderungsdatum)
```

```
VALUES (1, 2, 1, 3, CURRENT_TIMESTAMP);
```

```
-- Fehler: FK-Verletzung auf bestellung_id
```

```
INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id,  
    menge, aenderungsdatum)
```

```
VALUES (7001, 2, 3999, 1, CURRENT_TIMESTAMP);
```

```
-- Fehler: rechnung_id existiert nicht
```

```
INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge  
    , aenderungsdatum)
```

```
VALUES (8999, 1, 3001, 2, CURRENT_TIMESTAMP);
```

Listing 19: Primärschlüssel-Tests

```
-- Erfolgreiche erste Position
```

```
INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge  
    , aenderungsdatum)
```

```
VALUES (2, 1, 1, 2, CURRENT_TIMESTAMP);
```

```
-- Fehler: gleicher zusammengesetzter PK (rechnung_id, position_nr)
```

```
INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge  
    , aenderungsdatum)
```

```
VALUES (2, 1, 1, 5, CURRENT_TIMESTAMP);
```

Listing 20: NOT NULL-Tests

```
INSERT INTO bestellung (bestellung_id, kunde_id, adresse_id, status, datum
, aenderungsdatum)
VALUES (2, 1, 1, NULL, CURRENT_DATE, CURRENT_TIMESTAMP);
```

Listing 21: ON DELETE-Regeln testen

```
-- CASCADE: Lösche Rechnung, verknüpfte Positionen sollten mitgelöscht
werden
INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge
, aenderungsdatum)
VALUES (2, 2, 1, 1, CURRENT_TIMESTAMP);

DELETE FROM rechnung WHERE rechnung_id = 2;

-- Prüfen: keine Positionen zur gelöschten Rechnung vorhanden
SELECT COUNT(*) AS anzahl_pos
FROM rechnung_position
WHERE rechnung_id = 2;

-- RESTRICT: Löschen eines Artikels, der referenziert wird, sollte
fehlschlagen
DELETE FROM artikel WHERE artikel_id = 1;
```

Listing 22: CHECK-Constraints hinzufügen und testen

```
-- Beispiel-Checks
ALTER TABLE bestellung_position
ADD CONSTRAINT chk_bestpos_menge_pos
CHECK (menge > 0);

ALTER TABLE lagerbestand
ADD CONSTRAINT chk_lagerbestand_bestand_nonneg
CHECK (bestand >= 0);

-- Test: Verletzung der Checks (menge > 0)
INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id,
menge, aenderungsdatum)
VALUES (7001, 4, 3001, 0, CURRENT_TIMESTAMP);

-- Test: Verletzung des Lagerbestands-Checks (bestand >= 0)
UPDATE lagerbestand SET bestand = -1 WHERE artikel_id = 1 AND lagerort_id
= 1;
```

3.3 Indexe

Listing 23: Erstellen von Indexen

```
-- bestellung
CREATE INDEX IF NOT EXISTS idx_bestellung_kunde_id    ON bestellung(
    kunde_id);
CREATE INDEX IF NOT EXISTS idx_bestellung_adresse_id  ON bestellung(
    adresse_id);
CREATE INDEX IF NOT EXISTS idx_bestellung_status     ON bestellung(status)
;
CREATE INDEX IF NOT EXISTS idx_bestellung_datum      ON bestellung(datum);

-- rechnung
CREATE INDEX IF NOT EXISTS idx_rechnung_kunde_id     ON rechnung(kunde_id)
;
CREATE INDEX IF NOT EXISTS idx_rechnung_adresse_id   ON rechnung(
    adresse_id);
CREATE INDEX IF NOT EXISTS idx_rechnung_status      ON rechnung(status);
CREATE INDEX IF NOT EXISTS idx_rechnung_datum       ON rechnung(datum);

-- rechnung_position
CREATE INDEX IF NOT EXISTS idx_rechnung_position_rechnung_id  ON
    rechnung_position(rechnung_id);
CREATE INDEX IF NOT EXISTS idx_rechnung_position_artikel_id   ON
    rechnung_position(artikel_id);

-- bestellung_position
CREATE INDEX IF NOT EXISTS idx_bestellung_position_bestellung_id  ON
    bestellung_position(bestellung_id);
CREATE INDEX IF NOT EXISTS idx_bestellung_position_artikel_id      ON
    bestellung_position(artikel_id);

-- lager
CREATE INDEX IF NOT EXISTS idx_lagerbestand_lagerort_id  ON lagerbestand(
    lagerort_id);
CREATE INDEX IF NOT EXISTS idx_lagerort_adresse_id      ON lagerort(
    adresse_id);

-- warengruppe
CREATE INDEX IF NOT EXISTS idx_warengruppe_artikel_warengruppe_id  ON
```

```

warengruppe_artikel(warengruppe_id);

-- lieferant
CREATE INDEX IF NOT EXISTS idx_lieferant_adresse_id ON lieferant(
    adresse_id);
CREATE INDEX IF NOT EXISTS idx_lieferant_artikel_artikel_id ON
    lieferant_artikel(artikel_id);

```

3.4 Prozeduren

3.4.1 Prozedur zur Rabattberechnung

Die Prozedur `berechne_rabatt` berechnet für eine gegebene Rechnung die Positionssumme (Summe Menge × Einzelpreis), bestimmt anhand einer einfachen Regel einen Rabatt von 10% falls die Positionssumme über 249 liegt (ansonsten 0%), berechnet daraus den gerundeten Rabattbetrag und den Endbetrag, und speichert optional den Rabattprozentsatz in der Tabelle `rechnung`; abschließend gibt die Prozedur eine NOTICE mit Rechnungs-ID, Positionssumme, Rabattprozentsatz, Rabattbetrag und Endbetrag aus.

Listing 24: Prozedur zur Rabattberechnung

```

CREATE OR REPLACE PROCEDURE berechne_rabatt(
    IN p_rechnung_id INT,
    IN p_persist_rabatt BOOLEAN DEFAULT TRUE
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_positionssumme NUMERIC(14,2);
    v_rabatt_prozent INT := 0;
    v_rabatt_betrag NUMERIC(14,2);
    v_endbetrag NUMERIC(14,2);
BEGIN
    -- Positionssumme berechnen (benutze vorhandene betrag oder menge
    -- * einzelpreis)
    SELECT COALESCE(SUM(rp.menge * a.einzelpreis),0)::NUMERIC(14,2)
    INTO v_positionssumme
    FROM rechnung_position rp
    JOIN artikel a ON a.artikel_id = rp.artikel_id
    WHERE rp.rechnung_id = p_rechnung_id;

    -- Rabattregel: 10% nur wenn > 249

```

```

IF v_positionssumme > 249 THEN
v_rabatt_prozent := 10;
ELSE
v_rabatt_prozent := 0;
END IF;

v_rabatt_betrag := ROUND(v_positionssumme * (v_rabatt_prozent
    /100.0), 2);
v_endbetrag := ROUND(v_positionssumme - v_rabatt_betrag, 2);

IF p_persist_rabatt THEN
UPDATE rechnung
SET rabatt_prozent = v_rabatt_prozent,
aenderungsdatum = CURRENT_TIMESTAMP
WHERE rechnung_id = p_rechnung_id;
END IF;

RAISE NOTICE 'Rechnung %: positionssumme=%, rabatt_prozent=%,
    rabatt_betrag=%, endbetrag=%',
p_rechnung_id, v_positionssumme, v_rabatt_prozent, v_rabatt_betrag
    , v_endbetrag;
END;
$$;

```

Listing 25: Test von berechne_rabatt

```

-- adresse
INSERT INTO adresse (adresse_id, strasse, haus_nr, plz, ort,
    aenderungsdatum)
VALUES (1001, 'Teststrasse', '1', '60311', 'Frankfurt', CURRENT_TIMESTAMP)
    ;

-- kunde
INSERT INTO kunde (kunde_id, vorname, nachname, rechnung_adresse_id,
    liefer_adresse_id, aenderungsdatum)
VALUES (2001, 'Max', 'Mustermann', 1001, 1001, CURRENT_TIMESTAMP);

-- artikel
INSERT INTO artikel (artikel_id, bezeichnung, einzelpreis, aenderungsdatum
    )
VALUES

```

```

        (3001, 'Widget', 9.99, CURRENT_TIMESTAMP),
        (3002, 'Gadget', 149.50, CURRENT_TIMESTAMP);

-- rechnung
INSERT INTO rechnung (rechnung_id, kunde_id, adresse_id, datum, status,
    aenderungsdatum)
VALUES (8001, 2001, 1001, CURRENT_DATE, 'offen', CURRENT_TIMESTAMP);

-- rechnung_position
INSERT INTO rechnung_position (rechnung_id, position_nr, artikel_id, menge
    , aenderungsdatum)
VALUES
    (8001, 1, 3001, 10, CURRENT_TIMESTAMP),
    (8001, 2, 3002, 2, CURRENT_TIMESTAMP);

-- Prozedur aufrufen
CALL berechne_rabatt(8001);

```

3.4.2 Prozedur zur Überprüfung der Verfügbarkeit und Eintragung eines Auftrags

Die Prozedur `erstelle_bestellung` legt eine neue Bestellung an und stellt dabei die Bestandslogik sicher: Sie prüft zunächst die verfügbare Gesamtmenge des gewünschten Artikels über alle Lagerorte und wirft bei Unterdeckung eine aussagekräftige Exception. Bei ausreichendem Bestand erstellt sie die Bestellung mit Status „offen“, fügt die erste Positionszeile hinzu und reduziert den Lagerbestand am Lagerort mit dem höchsten aktuellen Bestand. Abschließend meldet sie per NOTICE die neu angelegte Bestellnummer sowie den aktualisierten Gesamtbestand des Artikels.

Listing 26: Prozedur zum Anlegen von Bestellungen

```

CREATE OR REPLACE PROCEDURE erstelle_bestellung(
    IN p_kunde_id INT,
    IN p_adresse_id INT,
    IN p_artikel_id INT,
    IN p_menge INT
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_bestand INT;
    v_bestellung_id INT;

```


BEGIN

-- Verfügbarkeit prüfen

```
SELECT SUM(bestand) INTO v_bestand
FROM lagerbestand
WHERE artikel_id = p_artikel_id;
```

IF v_bestand **IS NULL OR** v_bestand < p_menge **THEN**

```
RAISE EXCEPTION 'Artikel % nicht in ausreichender Menge verfügbar
  (Bestand: %, benötigt: %)',
```

```
p_artikel_id, COALESCE(v_bestand,0), p_menge;
```

END IF;

-- Bestellung anlegen

```
INSERT INTO bestellung (kunde_id, adresse_id, status, datum,
  aenderungsdatum)
```

```
VALUES (p_kunde_id, p_adresse_id, 'offen', CURRENT_DATE,
  CURRENT_TIMESTAMP)
```

```
RETURNING bestellung_id INTO v_bestellung_id;
```

-- Position einfügen

```
INSERT INTO bestellung_position (bestellung_id, position_nr,
  artikel_id, menge, aenderungsdatum)
```

```
VALUES (v_bestellung_id, 1, p_artikel_id, p_menge,
  CURRENT_TIMESTAMP);
```

-- Lagerbestand reduzieren

```
UPDATE lagerbestand
```

```
SET bestand = bestand - p_menge,
```

```
aenderungsdatum = CURRENT_TIMESTAMP
```

```
WHERE artikel_id = p_artikel_id
```

```
AND lagerort_id = (
```

```
SELECT lagerort_id
```

```
FROM lagerbestand
```

```
WHERE artikel_id = p_artikel_id
```

```
ORDER BY bestand DESC
```

```
LIMIT 1
```

```
);
```

-- aktuellen Bestand

```
SELECT SUM(bestand) INTO v_bestand
```

```

FROM lagerbestand
WHERE artikel_id = p_artikel_id;

RAISE NOTICE 'Bestellung % erfolgreich angelegt. Neuer Bestand von
Artikel %: % Stück',
v_bestellung_id, p_artikel_id, v_bestand;
END;
$$;

```

Listing 27: Test von erstelle_bestellung

```

INSERT INTO adresse (adresse_id, strasse, haus_nr, plz, ort,
aenderungsdatum)
VALUES (1001, 'Teststrasse', '1', '60311', 'Frankfurt', CURRENT_TIMESTAMP)
ON CONFLICT (adresse_id) DO NOTHING;

INSERT INTO kunde (kunde_id, vorname, nachname, rechnung_adresse_id,
liefer_adresse_id, aenderungsdatum)
VALUES (2001, 'Max', 'Mustermann', 1001, 1001, CURRENT_TIMESTAMP)
ON CONFLICT (kunde_id) DO NOTHING;

INSERT INTO artikel (artikel_id, bezeichnung, einzelpreis, aenderungsdatum
)
VALUES (3001, 'Widget', 9.99, CURRENT_TIMESTAMP)
ON CONFLICT (artikel_id) DO NOTHING;

INSERT INTO lagerort (lagerort_id, name, adresse_id, aenderungsdatum)
VALUES (6001, 'Hauptlager Frankfurt', 1001, CURRENT_TIMESTAMP)
ON CONFLICT (lagerort_id) DO NOTHING;

INSERT INTO lagerbestand (artikel_id, lagerort_id, bestand,
aenderungsdatum)
VALUES (3001, 6001, 100, CURRENT_TIMESTAMP)
ON CONFLICT (artikel_id, lagerort_id) DO UPDATE
SET bestand = EXCLUDED.bestand;

CALL erstelle_bestellung(2001, 1001, 3001, 5);

```

3.5 Trigger

Der Trigger `trg_auto_rechnung_from_bestellung` erzeugt automatisch eine Rechnung, sobald der Bestellstatus auf "Zugestellt" wechselt. Dabei legt die Triggerfunktion `trg_auto_rechnung_from_bestellung` eine neue Rechnung für den betreffenden Kunden und die Lieferadresse an, kopiert alle Bestellpositionen in die zugehörigen Rechnungspositionen, ruft die Prozedur `berechne_rabatt` zur Berechnung des Rabatts auf, setzt den Rechnungsstatus auf "erstellt".

Listing 28: `trg_auto_rechnung_from_bestellung`

```
CREATE OR REPLACE FUNCTION trg_auto_rechnung_from_bestellung()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_rechnung_id INT;
BEGIN
    -- Nur reagieren, wenn Status neu 'Zugestellt' ist
    IF NEW.status = 'zugestellt' AND (OLD.status IS DISTINCT FROM 'zugestellt') THEN

        -- 1) Rechnung anlegen (ohne gesamtbetrag)
        INSERT INTO rechnung (kunde_id, adresse_id, datum, status,
            aenderungsdatum)
        VALUES (NEW.kunde_id, NEW.adresse_id, CURRENT_DATE, 'offen',
            CURRENT_TIMESTAMP)
        RETURNING rechnung_id INTO v_rechnung_id;

        -- 2) Positionen aus Bestellung übernehmen
        INSERT INTO rechnung_position (rechnung_id, position_nr,
            artikel_id, menge, aenderungsdatum)
        SELECT
            v_rechnung_id,
            bp.position_nr,
            bp.artikel_id,
            bp.menge,
            CURRENT_TIMESTAMP
        FROM bestellung_position bp
        WHERE bp.bestellung_id = NEW.bestellung_id;
```

```

-- 3) Rabatt-Procedure aufrufen (setzt rabatt_prozent in
rechnung)
CALL berechne_rabatt(v_rechnung_id);

-- 4) Rechnungstatus aktualisieren
UPDATE rechnung
SET status = 'erstellt',
aenderungsdatum = CURRENT_TIMESTAMP
WHERE rechnung_id = v_rechnung_id;

RAISE NOTICE 'Rechnung % erstellt aus Bestellung % (
Trigger).', v_rechnung_id, NEW.bestellung_id;
END IF;

RETURN NEW;
END;
$$;

-- Trigger an Tabelle bestellung hängen
CREATE OR REPLACE TRIGGER trg_auto_rechnung_from_bestellung
AFTER UPDATE OF status ON bestellung
FOR EACH ROW
WHEN (NEW.status = 'zugestellt' AND OLD.status IS DISTINCT FROM '
zugestellt')
EXECUTE FUNCTION trg_auto_rechnung_from_bestellung();

```

Listing 29: Test von trg_auto_rechnung_from_bestellung

```

INSERT INTO adresse (strasse, haus_nr, plz, ort) VALUES ('Teststrasse', '1'
, '60311', 'Frankfurt') RETURNING adresse_id;

INSERT INTO kunde (vorname, nachname, rechnung_adresse_id,
liefer_adresse_id)
VALUES ('Max', 'Mustermann', 1, 1) RETURNING kunde_id;

INSERT INTO artikel (bezeichnung, einzelpreis) VALUES ('Widget', 9.99), ('
Gadget', 149.50);

INSERT INTO bestellung (kunde_id, adresse_id, status) VALUES (1, 1, 'offen')
RETURNING bestellung_id;

```

```
INSERT INTO bestellung_position (bestellung_id, position_nr, artikel_id,
    menge)
VALUES (1,1,1,10), (1,2,2,2);

-- Trigger auslösen: Statuswechsel auf 'zugestellt'
UPDATE bestellung
SET status = 'zugestellt', aenderungsdatum = CURRENT_TIMESTAMP
WHERE bestellung_id = 1;

-- Kontrolle: erzeugte Rechnung und Positionen prüfen
SELECT * FROM rechnung WHERE rechnung_id = (SELECT MAX(rechnung_id) FROM
    rechnung);
```

4 Zusammenfassung

Die Dokumentation beschreibt ein kompaktes Subsystem für Bestellwesen und Abrechnung, das Bestellungen anlegt, Lagerbestände verwaltet, bei Lieferung automatisch Rechnungen erzeugt und Rabattregeln anwendet.

Die Kernkomponenten sind:

- **berechne_rabatt**

Eine Prozedur zur Ermittlung der Positionssumme einer Rechnung, Anwendung einer Schwellenwertregel (10 % Rabatt bei Positionssumme > 249), Berechnung von Rabattbetrag und Endbetrag.

- **erstelle_bestellung**

Eine Prozedur zur Verfügbarkeitsprüfung im Lager, Anlage einer Bestellung und zugehöriger Positionen sowie Reduktion des Lagerbestands; bei unzureichendem Bestand wird eine Exception ausgelöst.

- **trg_auto_rechnung_from_bestellung**

Ein Trigger und eine Triggerfunktion, die beim Statuswechsel einer Bestellung auf zugestellt automatisch eine Rechnung anlegt, Bestellpositionen in Rechnungspositionen kopiert, die Rabattprozedur aufruft und den Rechnungsstatus auf `erstellt` setzt.

- **Test- und Setup-Skripte**

Test- und Setup-Skripte liefern Beispielaufrufe, mit denen sich das Verhalten reproduzierbar prüfen lässt.

Die Komponenten bilden zusammen einen durchgängigen Ablauf von Bestellung über Lieferstatus bis zur automatischen Rechnungserstellung inklusive Rabattberechnung und Bestandsführung.