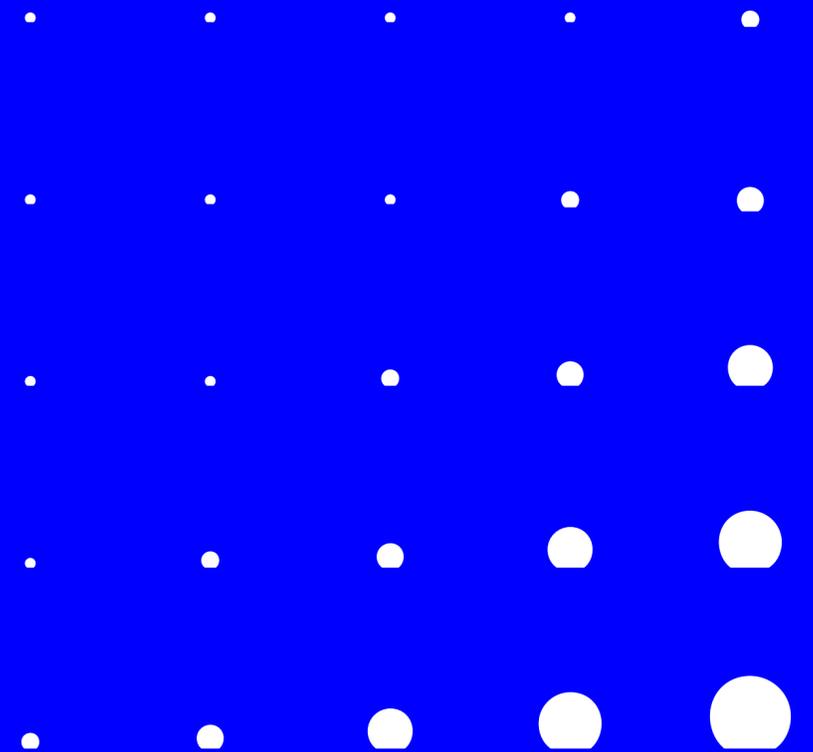


## Transaction Control Language (TCL) und Stored Procedures



# Warum Transaktionen?



# Anforderungen an eine Transaktion: ACID

- Atomarität (Atomicity):
  - Eine Transaktion wird entweder ganz oder gar nicht ausgeführt.
  - Transaktionen sind also „unteilbar“
  - Wenn eine atomare Transaktion abgebrochen wird, ist das System unverändert.
- Konsistenz (Consistency):
  - Nach Ausführung der Transaktion muss der Datenbestand in einer konsistenten Form sein, wenn er es bereits zu Beginn der Transaktion war.

# Anforderungen an eine Transaktion: ACID

- Isolation (Isolation):
  - Bei gleichzeitiger Ausführung mehrerer Transaktionen dürfen sich diese nicht gegenseitig beeinflussen.
- Dauerhaftigkeit (Durability):
  - Die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben.
  - Die Effekte von Transaktionen dürfen also nicht verloren gehen oder mit der Zeit verblassen.
  - Eine Verschachtelung von Transaktionen ist wegen dieser Eigenschaft streng genommen nicht möglich, da ein Zurücksetzen der äußeren Transaktion die Dauerhaftigkeit einer inneren, bereits ausgeführten Transaktion verletzen würde.

# Ablauf, Blockierung & Deadlock einer Transaktion

- Eine Transaktion in MariaDB beginnt damit, dass Sie das Auto-Commit auf false setzen.
  - Im Anschluß daran kommen die SQL-Statements, die zu der Transaktion gehören.
  - Die Transaktion wird dann erfolgreich abgeschlossen mit **COMMIT** ;  
Dabei wird auch direkt eine neue Transaktion gestartet.
  - Ein Abbruch ist mit dem SQL-Kommando **ROLLBACK** ; möglich.
- Wenn eine Transaktion aufgrund einer anderen Transaktion nicht ausgeführt werden kann, spricht man von einer Blockierung.
- Wird die erste Transaktion durch die zweite und gleichzeitig die zweite durch die erste blockiert, so spricht man von einem Deadlock (Verklemmung).

# Umsetzung von Transaktionen in Java mit JDBC

The screenshot shows the phpMyAdmin interface for a MySQL database named 'bank'. The current view is the 'Beziehungsansicht' (Relationship View) for the 'konto' table. The table structure is as follows:

#	Name	Typ	Kollation	Attribute	Null	Standard	Kommentare	Extra
<input type="checkbox"/>	1 ID	bigint(20)			Nein	kein(e)		AUTO_INCREMENT
<input type="checkbox"/>	2 inhaber	varchar(50)	utf8mb4_general_ci		Nein	kein(e)		
<input type="checkbox"/>	3 guthaben	decimal(6,2)			Nein	kein(e)		

Below the table structure, a query is executed: `SELECT * FROM `konto``. The result shows 2 rows:

	ID	inhaber	guthaben
<input type="checkbox"/>	1	Person X	5000.00
<input type="checkbox"/>	2	Person Y	2000.00

# Umsetzung von Transaktionen in Java mit JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Transaktion_01 {

    public static void main(String[] args) {
        // Datenbank-Parameter:
        String url = "jdbc:mysql://localhost:3306/bank"; // IP, TCP-Port und Datenbankname
        String user = "root"; // MySQL-Benutzername
        String password = ""; // Passwort dieses Benutzers
        // Verbindung herstellen:
        Connection conn = null;
        PreparedStatement abhebenPS = null;
        PreparedStatement einzahlenPS = null;
        try {
            // Verbindung zur Datenbank aufbauen
            conn = DriverManager.getConnection(url, user, password);
            conn.setAutoCommit(false); // Transaktionsmodus aktivieren

            // 1000 EUR von Person X (ID = 1) abziehen
            String abhebenSQL = "UPDATE konto SET guthaben = guthaben - ? WHERE ID = ?";
            abhebenPS = conn.prepareStatement(abhebenSQL);
            abhebenPS.setBigDecimal(1, new java.math.BigDecimal("1000.00"));
            abhebenPS.setInt(2, 1);
            int betragAbgezogen = abhebenPS.executeUpdate();
```

**Hier passiert noch nichts!**

# Umsetzung von Transaktionen in Java mit JDBC

```
// 1000 EUR zu Person Y (ID = 2) hinzufügen
String einzahlenSQL = "UPDATE konto SET guthaben = guthaben + ? WHERE ID = ?";
einzahlenPS = conn.prepareStatement(einzahlenSQL);
einzahlenPS.setBigDecimal(1, new java.math.BigDecimal("1000.00"));
einzahlenPS.setInt(2, 2);
int betragEingezahlt = einzahlenPS.executeUpdate();

// Transaktion prüfen und abschließen
if (betragAbgezogen == 1 && betragEingezahlt == 1) {
    conn.commit(); // Alles erfolgreich -> Änderungen übernehmen
    System.out.println("☑ Transaktion erfolgreich: 1000 EUR von Person X an Person Y überwiesen.");
} else {
    conn.rollback(); // Fehler -> Alle Änderungen rückgängig machen
    System.out.println("⚠ Transaktion fehlgeschlagen: Rollback durchgeführt.");
}
}
```

**Irgendwas ist falsch gelaufen:  
Alles rückgängig machen!**

**Bestätigt die aktuelle Transaktion  
und startet direkt eine neue Transaktion!**

# Umsetzung von Transaktionen in Java mit JDBC

✓ Zeige Datensätze 0 - 1 (2 insgesamt, Die Abfrage dauerte 0,0001 Sekunden.)

```
SELECT * FROM `konto`
```

Messen [ [Inline bearbeiten](#) ] [ [Bearbeiten](#) ] [ [SQL erklären](#) ] [ [PHP-Code erzeugen](#) ] [ [Aktualisieren](#) ]

Alles anzeigen | Anzahl der Datensätze:  Zeilen filtern:

Zusätzliche Optionen

				ID	inhaber	guthaben
<input type="checkbox"/>	Bearbeiten	Kopieren	Löschen	1	Person X	4000.00
<input type="checkbox"/>	Bearbeiten	Kopieren	Löschen	2	Person Y	3000.00

# Stored Procedures

# Was ist eine Stored Procedure?

- Eine gespeicherte Prozedur ist eine Funktion eines DBMS, mit der ganze Abläufe von Anweisungen vom Datenbank-Client aufgerufen werden können.
- Sie ist somit ein eigenständiger Befehl, der eine Abfolge gespeicherter Befehle ausführt.
- Gespeicherte Prozeduren werden im Data-Dictionary der jeweiligen Datenbank gespeichert.
- Mitunter wird dadurch die Leistung gesteigert, da weniger Daten zwischen Client und dem DBS ausgetauscht werden müssen und das DBMS häufig auf leistungsfähigeren Servern läuft.

# Sprache für eine Stored Procedure

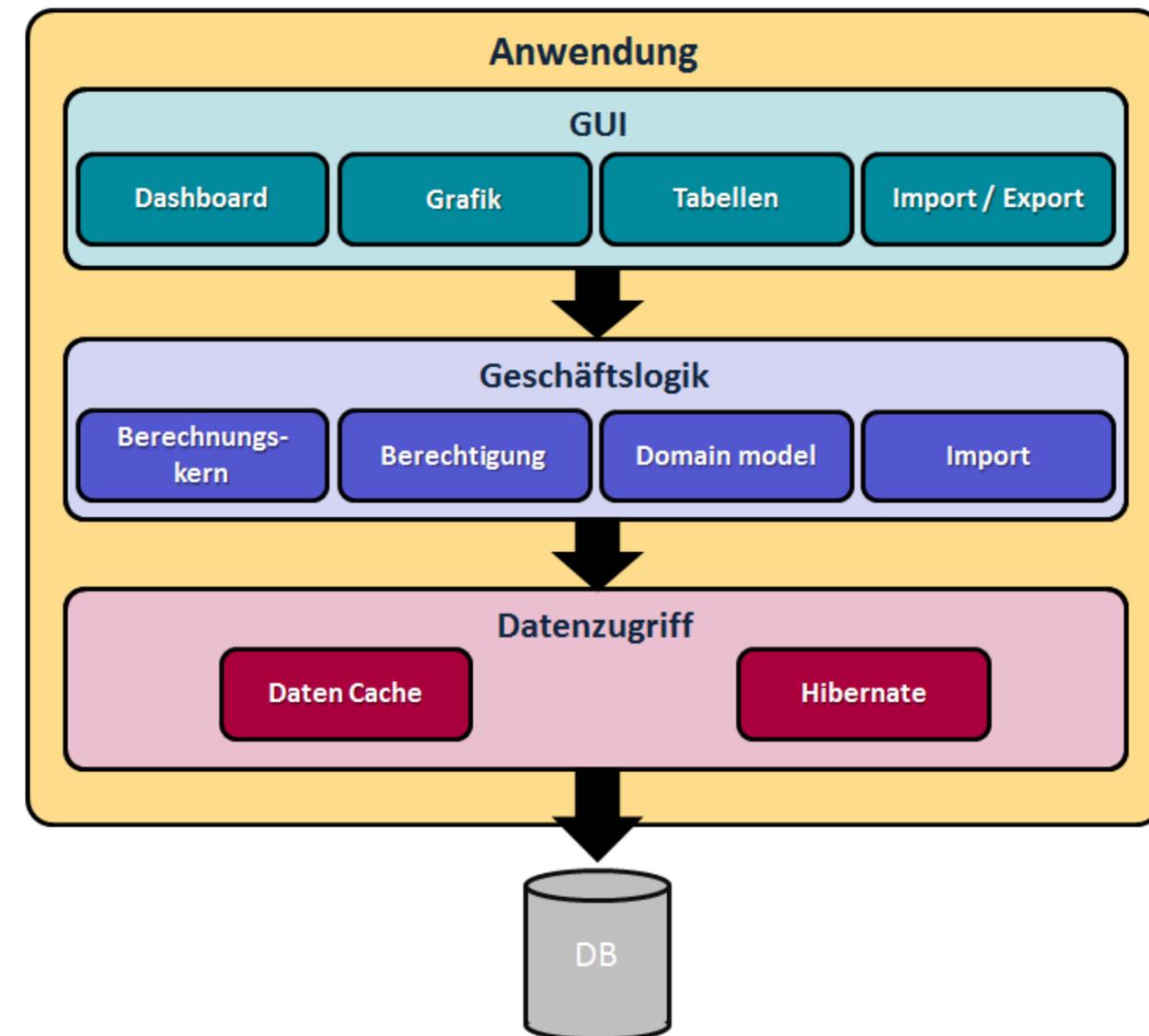
- Neben der gewöhnlichen Syntax der Abfragesprache wie SQL können in gespeicherten Prozeduren auch zusätzliche Befehle zur Ablaufsteuerung oder Auswertung von Bedingungen hinzugefügt werden.
- Damit können sie mit Makrosprachen bestimmter Anwendungsprogramme verglichen werden.
- Oft wird das verwendete SQL um herstellerspezifische Funktionen erweitert.
- Auch der Einsatz anderer Programmiersprachen wie etwa Java oder C# ist inzwischen teilweise möglich.

# Sprache für eine Stored Procedure

Datenbank	Stored Procedure-Sprache	Besonderheiten
MySQL / MariaDB	SQL/PSM (Persistent Stored Modules)	Nutzt <code>DELIMITER</code> , unterstützt keine <code>OUTER</code> - Transaktionen.
PostgreSQL	PL/pgSQL (Procedural Language/PostgreSQL SQL)	Sehr leistungsfähig, unterstützt komplexe Datentypen und Trigger.
SQL Server (MSSQL)	T-SQL (Transact-SQL)	Erweiterte Fehlerbehandlung ( <code>TRY...CATCH</code> ), leistungsstarke Cursor.
Oracle	PL/SQL (Procedural Language/SQL)	Sehr umfangreich, unterstützt Pakete, Funktionen und Trigger.
IBM Db2	SQL PL (Procedural Language)	Unterstützt Cursor und komplexe Transaktionen.
Firebird	PSQL (Procedural SQL)	Bietet <code>WHEN...DO</code> -Blöcke für Fehlerbehandlung.
SQLite	<b>✗</b> Keine native Stored Procedure- Unterstützung	Wird oft durch externe Skripte ersetzt.

# Was kommt nicht in eine Stored Procedure?

- Auch wenn die Zugriffe ggf. extrem schnell sind, sollte man typische Geschäftslogik NICHT in Stored Procedures auslagern, da man dadurch das Schichtenmodell verletzt und abhängig von der Datenbank wird.



# Was kommt denn in eine Stored Procedure?

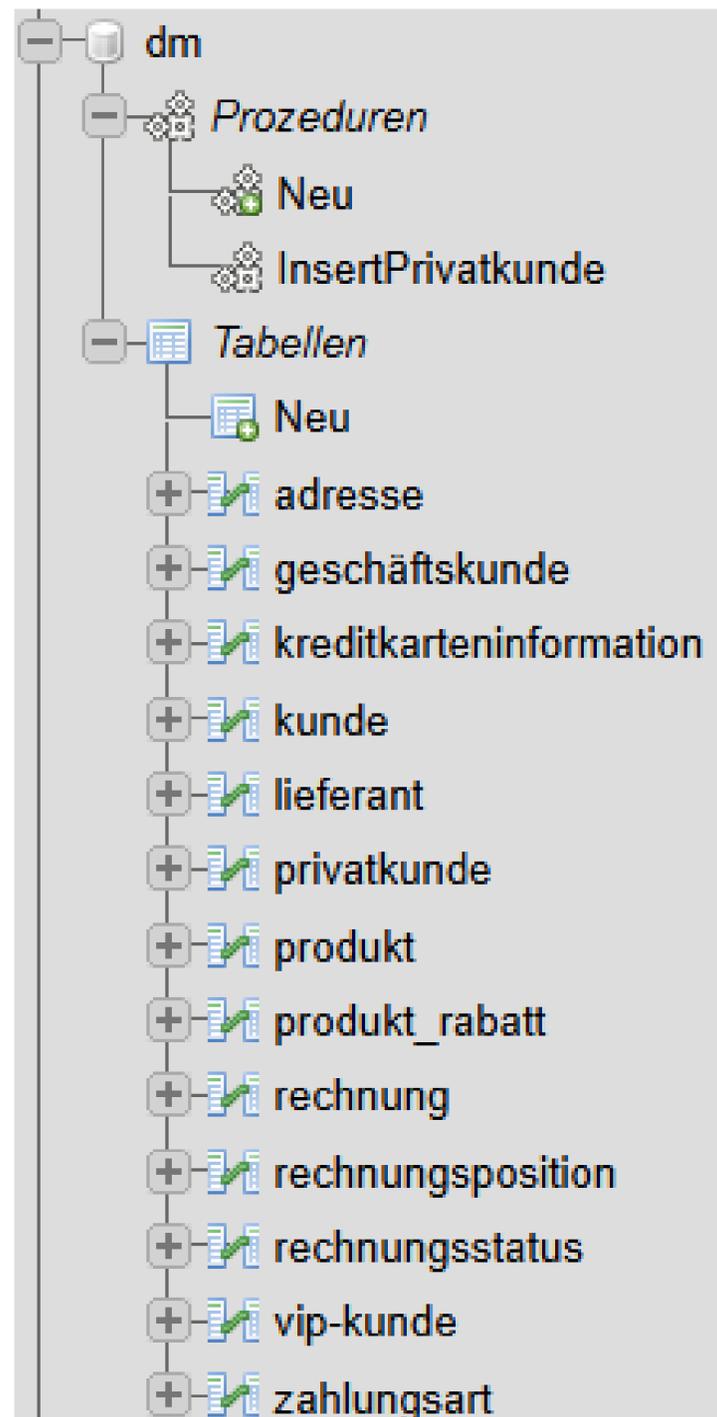
- Kombinierte SQL-Abfragen, die man sehr oft in Zusammenhang verwendet.
- Abfragen, bei denen man zuvor noch Prüfungen vornehmen will.
- Abfragen, bei denen man noch etwas anderes anstoßen will.
  - Beispielsweise ein Logging von Aktionen, immer wenn etwas Bestimmtes gemacht wird.

# Beispiel für eine Stored Procedure in unserer Aufgabe...

- Automatisiertes Einfügen eines neuen Privatkunden als speziellen Kunden!

```
DELIMITER //
CREATE PROCEDURE InsertPrivatkunde (
    IN kundenummer BIGINT(20),
    IN id_adresse BIGINT(20),
    IN kreditkarteninformation BIGINT(20),
    IN vorname VARCHAR(50),
    IN nachname VARCHAR(50)
)
BEGIN
    -- Kunde in die Tabelle 'kunde' einfügen
    INSERT INTO kunde (kundenummer, id_adresse, kreditkarteninformation)
    VALUES (kundenummer, id_adresse, kreditkarteninformation);
    -- Privatkunde mit derselben Kundenummer hinzufügen
    INSERT INTO privatkunde (kundenummer, vorname, nachname)
    VALUES (kundenummer, vorname, nachname);
END //
DELIMITER ;
```

# Übersicht in phpMyAdmin



Prozeduren-Name

Typ

Parameter

**PROCEDURE**

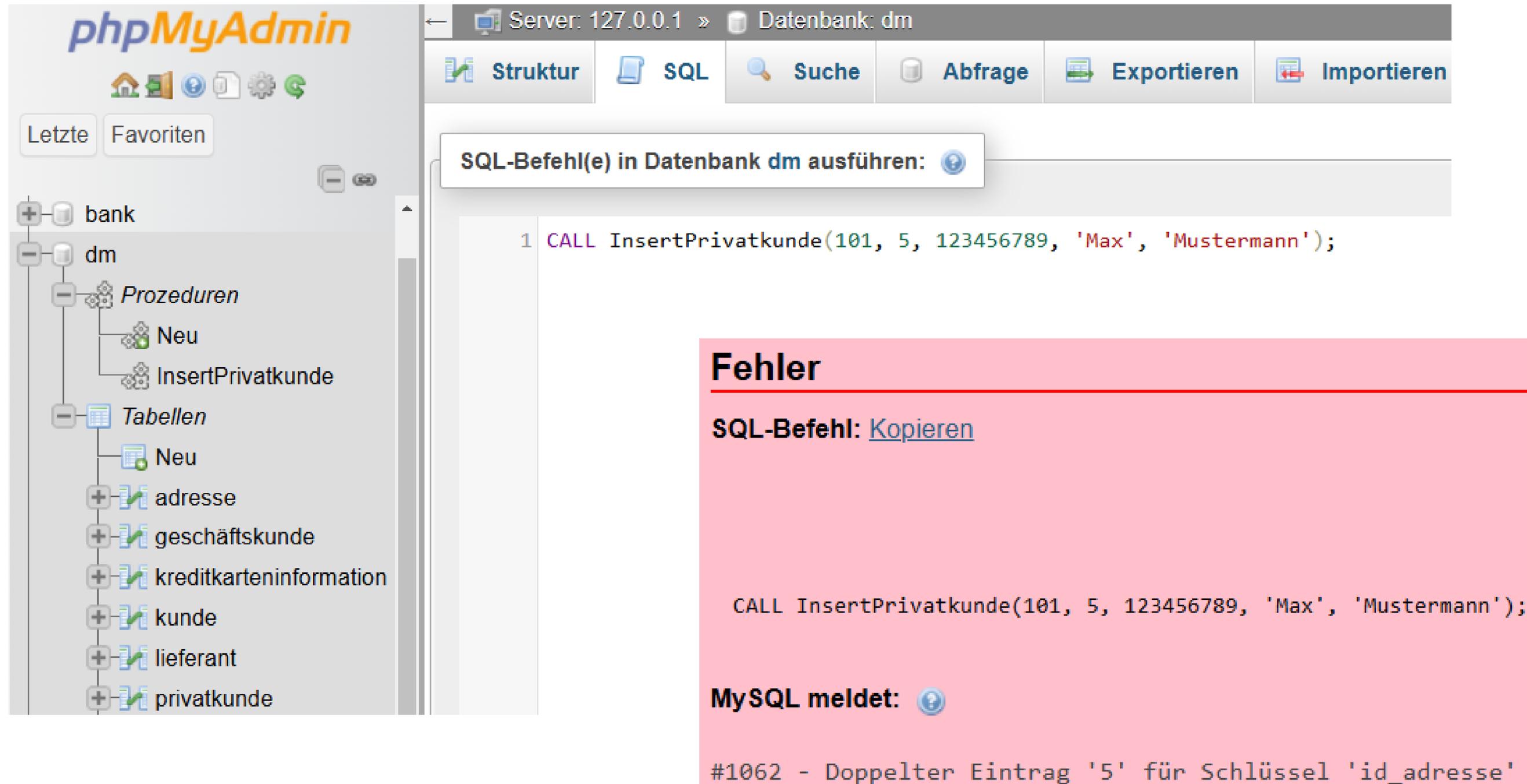
	Richtung	Name	Typ	Länge/Werte
‡	IN	<input type="text" value="kundennummer"/>	<input type="text" value="BIGINT"/>	<input type="text" value="20"/>
‡	IN	<input type="text" value="id_adresse"/>	<input type="text" value="BIGINT"/>	<input type="text" value="20"/>
‡	IN	<input type="text" value="kreditkarteninformation"/>	<input type="text" value="BIGINT"/>	<input type="text" value="20"/>
‡	IN	<input type="text" value="vorname"/>	<input type="text" value="VARCHAR"/>	<input type="text" value="50"/>
‡	IN	<input type="text" value="nachname"/>	<input type="text" value="VARCHAR"/>	<input type="text" value="50"/>

Definition

```
BEGIN
  -- Kunde in die Tabelle 'kunde' einfügen
  INSERT INTO kunde (kundennummer, id_adresse, kreditkarteninformation)
  VALUES (kundennummer, id_adresse, kreditkarteninformation);

  -- Privatkunde mit derselben Kundennummer hinzufügen
  INSERT INTO privatkunde (kundennummer, vorname, nachname)
  VALUES (kundennummer, vorname, nachname);
END
```

# Aufrufen der Prozedur via Java/JDBC oder direkt via phpMyAdmin



The screenshot shows the phpMyAdmin interface for a database named 'dm'. The left sidebar displays the database structure, including a 'Prozeduren' (Procedures) folder containing 'Neu' and 'InsertPrivatkunde', and a 'Tabellen' (Tables) folder containing 'adresse', 'geschäftskunde', 'kreditkarteninformation', 'kunde', 'lieferant', and 'privatkunde'. The main area shows the 'SQL' tab with a single query: `1 CALL InsertPrivatkunde(101, 5, 123456789, 'Max', 'Mustermann');`. Below the query, a red error box is displayed with the following content:

**Fehler**  
SQL-Befehl: [Kopieren](#)

`CALL InsertPrivatkunde(101, 5, 123456789, 'Max', 'Mustermann');`

**MySQL meldet:** [?](#)

#1062 - Doppelter Eintrag '5' für Schlüssel 'id\_adresse'

# Wie kommt der Fehler zustande?

- Wenn Sie einen neuen Privatkunden anlegen, müssen Sie zunächst (wie bei der Vererbung in Java) einen neuen Kunden anlegen; das wurde berücksichtigt!
- Dieser Kunde braucht aber eine Adresse und zwischen kunde und adresse besteht eine 1:1-Beziehung; eine bestehende Adresse kann man also nicht einem zweiten Kunden zuordnen.
- Sie müssen die Stored Procedure so modifizieren, dass auch noch die nötigen Adressdaten eingegeben werden müssen als Input-Parameter, so dass mit dem neuen Kunden automatisch auch eine neue Adresse dieses Kunden angelegt wird.

# Beispiel für eine Stored Procedure in unserer Aufgabe...

- Logging, wenn man einen Kunden löscht!
1. Erstellung einer separaten Log-Tabelle:

```
CREATE TABLE IF NOT EXISTS kunden_log (  
    log_id INT AUTO INCREMENT PRIMARY KEY,  
    kundenummer BIGINT(20),  
    loeschzeit TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    aktion VARCHAR(255)  
);
```

# Beispiel für eine Stored Procedure in unserer Aufgabe...

2. Erstellung eines Triggers, der automatisch einen Eintrag in diese neue Tabelle erstellt, wenn ein Kunde gelöscht wird:

```
DELIMITER //  
CREATE TRIGGER Kunde_Loeschen_Protokoll  
BEFORE DELETE ON kunde  
FOR EACH ROW  
BEGIN  
    INSERT INTO kunden_log (kundennummer, aktion)  
    VALUES (OLD.kundennummer, 'Kunde gelöscht');  
END //  
DELIMITER ;
```