

►► Übungsaufgaben TSQL

- **STORED PROCEDURES**
- **TRIGGER**
- **FUNKTIONEN**
- **CURSOR**
- **DATENBANKVERBINDUNG
MIT JAVA**

1. Einführung

2. Datenbankentwurf

3. Datenbankimplementierung

4. Physische Datenorganisation

5. Anfrageoptimierung

6. Transaktionsverwaltung

7. Datensicherheit und Wiederherstellung

8. Business Intelligence

Syntax

Syntaxerklärung

Variablendeklaration und -definition:

```
Declare @Variablenname int;
```

Variableninitialisierung:

```
Select @Variablenname = 5;
```

...oder in einem Schritt:

```
Declare @Variablenname int = 5;
```

weiteres Beispiel:

```
Declare @Variablenname varchar(7) = 'Konsole';
```

Syntaxerklärung

Ausgabe auf Konsole

```
Print 'Dieser Text wird auf Konsole ausgegeben'
```

Ausgabe auf Konsole mit Variableninhalt und Stringverkettung

```
Declare @Variablenname varchar(7) = 'Konsole';  
Print 'Dieser Text wird auf ' + @Variablenname + '  
ausgegeben';
```

Je nach Datentyp u.U. Casts notwendig

```
Declare @Variablenname int = 5;  
Print 'Das ist die Zahl ' +  
cast(@Variablenname as varchar);
```

Syntaxerklärung

Verzweigung if / else if / else und Anweisungsblöcke

```
Declare @Variablenname int = 2;
```

```
if @variablenname = 1
    Print 'Das ist die Zahl ' +
        cast(@Variablenname as varchar);
else if @Variablenname = 2
    Print 'Das ist die Zahl ' +
        cast(@Variablenname as varchar);
else
    begin
        Print 'Wir befinden uns im Else-Zweig';
    end
```

Syntaxerklärung

Case

```
Declare @Variablenname int = 3;
```

```
Select
```

```
Case
```

```
When @variablenname = 1 Then 'Das ist die Zahl ' +
```

```
Cast(@Variablenname as varchar)
```

```
When @variablenname = 2 Then 'Das ist die Zahl ' +
```

```
Cast(@Variablenname as varchar)
```

```
Else 'Wir befinden uns im Else-Zweig'
```

```
End
```

1. Einführung

2. Datenbankentwurf

3. Datenbankimplementierung

4. Physische Datenorganisation

5. Anfrageoptimierung

6. Transaktionsverwaltung

7. Datensicherheit und Wiederherstellung

8. Business Intelligence

Stored Procedures

Stored Procedures

Gespeicherte Prozeduren gleichen den Prozeduren in anderen Programmiersprachen bezüglich der folgenden Merkmale und Fähigkeiten:

- Annehmen von Eingabeparametern und Zurückgeben mehrerer Werte in Form von Ausgabeparametern an die aufrufende Prozedur oder den aufrufenden Batch.
- Aufnehmen von Programmieranweisungen, die Vorgänge in der Datenbank ausführen, einschließlich des Aufrufs anderer Prozeduren.
- Zurückgeben eines Statuswertes an eine aufrufende Prozedur oder einen aufrufenden Batch, der Erfolg oder Misserfolg (sowie die Ursache für den Fehler) anzeigt.

Hinweis

Wir arbeiten alle in der gleichen Datenbank.

Bitte benennen Sie deshalb aufgrund der Übersichtlichkeit Ihre erstellten Prozeduren, Trigger, Funktionen und Cursor sinnvoll, z.B. durch Voranstellen Ihres Nachnamens.

Beispiel:

<Mein_Nachname>_sp_<Prozedurname>

Stored Procedures – Aufgabe

Erstellen Sie Ihre eigenen Tabellen in der „Spieler2_LAB“-Datenbank.

Rufen Sie dazu die bereits vorhandene Stored Procedure „sp_CreateTables“ wie folgt auf und überprüfen Sie das Ergebnis.

```
exec sp_CreateTables 'IhrNachname'
```

Stored Procedures

Beispielsyntax: **CREATE PROCEDURE**
 dbo.Sample_Procedure
 @param1 int = 0,
 @param2 int

 AS
 --insert code here
 --SELECT @param1,@param2

 RETURN 0

Aufruf: **EXEC** **dbo.Sample_Procedure**

Stored Procedures – Aufgabe 1

Schreiben Sie eine Stored Procedure, die eine schnelle Übersicht der erhaltenen Gesamtstrafen je Spieler aufzeigt.

(Name der Prozedur: „Mein_Nachname_...“ !)

Stored Procedures – Aufgabe 1

1 exec uebersicht_strafe

100 %

Results Messages

	Spieleummer	Nachname	Vorname	Strafe
1	2	Elfers	Rainer	245,00
2	6	Peters	Robert	188,50
3	7	Wiegand	Günther	144,00
4	8	Neuhaus	Berta	173,00
5	27	Kohl	Dagmar	107,50
6	28	Kohl	Claudia	114,50
7	39	Bischof	Dennis	157,50
8	44	Bäcker	Egon	70,00
9	57	Böhmen	Manfred	386,00
10	83	Hofmann	Philipp	68,00
11	95	Müller	Paul	147,00
12	100	Peters	Franz	152,50
13	104	Maurer	Doris	214,00
14	112	Bauer	Irene	109,50

Stored Procedures – Aufgabe 2

Erweitern Sie die in Aufgabe 1 erstellte Stored Procedure um einen Eingabeparameter, so dass Sie die Wahl über die Ausgabe der Gesamtstrafe oder der Einzelstrafen haben. Falls ein unbekannter Parameter übergeben wird, soll eine Fehlermeldung auf Konsole ausgegeben werden.

Stored Procedures – Aufgabe 2

```
1 exec uebersicht_strafe_parameter 'gesamt'
2 exec uebersicht_strafe_parameter 'einzeln'
3 exec uebersicht_strafe_parameter 'x'
```

	Spielenummer	Nachname	Vorname	Strafe
1	2	Elfers	Rainer	245,00
2	6	Peters	Robert	188,50
3	7	Wiegand	Günther	144,00
4	8	Neuhaus	Berta	173,00
5	27	Kohl	Dagmar	107,50
6	28	Kohl	Claudia	114,50
7	39	Bischof	Dennis	157,50
8	44	Bäcker	Egon	70,00
9	57	Böhmen	Manfred	386,00
10	83	Hofmann	Philipp	68,00
11	95	Müller	Paul	147,00
12	100	Peters	Franz	152,50
13	104	Maurer	Doris	214,00
14	112	Bauer	Irene	109,50

	Spielenummer	Nachname	Vorname	Strafe
1	2	Elfers	Rainer	21,50
2	2	Elfers	Rainer	6,50
3	2	Elfers	Rainer	24,00
4	2	Elfers	Rainer	32,50
5	2	Elfers	Rainer	7,50
6	2	Elfers	Rainer	36,50
7	2	Elfers	Rainer	34,00
8	2	Elfers	Rainer	34,00
9	2	Elfers	Rainer	48,50
10	6	Peters	Robert	50,50
11	6	Peters	Robert	37,50
12	6	Peters	Robert	34,00
13	6	Peters	Robert	12,50
14	6	Peters	Robert	16,00

Messages

Sie müssen entweder 'gesamt' für die Gesamtstrafe oder 'einzeln' für eine Liste der Einzelstrafen eingeben

Stored Procedures – Aufgabe 3

Schreiben Sie eine Prozedur, die ein Preisgeld für einen bestimmten Wettkampf anlegt. Wenn bereits ein Preisgeld für diesen Wettkampf vorhanden ist, soll eine Konsolenmeldung erscheinen und kein Eintrag in die Tabelle geschrieben werden.

Stored Procedures – Aufgabe 3

```
1 | exec addpreisgeld 2, 50.00|
```

100 %



Messages

Preisgeld für diese Wettkampf ID bereits vorhanden

```
1 | exec addpreisgeld 3, 50.00
```

100 %



Messages

(1 row(s) affected)
Preisgeld angelegt

1. Einführung

2. Datenbankentwurf

3. Datenbankimplementierung

4. Physische Datenorganisation

5. Anfrageoptimierung

6. Transaktionsverwaltung

7. Datensicherheit und Wiederherstellung

8. Business Intelligence

Trigger

Trigger

Ein Trigger ist eine spezielle Art von gespeicherter Prozedur, die automatisch ausgeführt wird, wenn ein Ereignis auf dem Datenbankserver auftritt.

Beispielsyntax:

```
CREATE TRIGGER TriggerName
ON [dbo].[TableName]
FOR DELETE, INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON
    --insert code here
END
```

Anderes Beispiel:

```
CREATE TRIGGER DDL_Database_Trigger
ON ALL SERVER
FOR ALTER_DATABASE
AS
```


Trigger – Aufgabe 1

Erstellen Sie einen Trigger, der beim Hinzufügen einer neuen Strafe die Anzahl sowie die Gesamtsumme aller Strafen auf der Konsole ausgibt.

Trigger – Aufgabe 1

```
1 | insert into strafe values (99,2,'2004-12-20 00:00:00.000',1,'2004-12-20 00:00:00.000',99.99)
```

100 %

 Messages

Derzeit sind 85 Strafen mit einer Gesamtsumme von 2376.99€ eingetragen

(1 row(s) affected)

Trigger – Aufgabe 2

Erstellen Sie einen Trigger, der beim Löschen von Zeilen aus der Tabelle „wettkampf“ einen Eintrag in die Tabelle „logWettkampf“ schreibt.

In die Logtabelle soll ein Zeitstempel, Hostname, Username, der Aktionstyp sowie der Name der Tabelle von der gelöscht wurde vermerkt werden.

Zusätzlich sollen alle Werte der gelöschten Zeilen in die Logtabelle geschrieben werden.

Trigger – Aufgabe 2

	zeitstempel	Hostname	Username	aktionstyp	tabellenname	id	spielemr	team_id	punktzahlspieler	punktzahlgegner	datum
1	2015-04-30 11:41:02.043	ITPC1074	caurenz	DELETE	wettkampf	373	100	1	5	8	2006-11-24 00:00:00.000
2	2015-05-05 08:19:11.720	ITPC1074	caurenz	DELETE	wettkampf	374	6	2	0	6	2004-01-12 00:00:00.000

Hilfreiche Funktionen:

CURRENT_TIMESTAMP

HOST_NAME()

SUSER_NAME()

Trigger – Aufgabe 2

Verwenden Sie zum Lösen der Aufgabe die „deleted“-Tabelle.

Informieren Sie sich dazu auf der MSDN-Website unter dem Stichwort „CREATE TRIGGER (Transact-SQL)“

bzw. unter

<https://msdn.microsoft.com/de-de/library/ms189799.aspx>

Trigger – Aufgabe 2

Die „deleted“-Tabelle

In der deleted-Tabelle werden Kopien der von DELETE- und UPDATE-Anweisungen betroffenen Zeilen gespeichert. Während der Ausführung einer DELETE- oder UPDATE-Anweisung werden Zeilen aus der Triggertabelle gelöscht und in die deleted-Tabelle übertragen. Die deleted-Tabelle und die Triggertabelle enthalten normalerweise nicht die gleichen Zeilen.

1. Einführung

2. Datenbankentwurf

3. Datenbankimplementierung

4. Physische Datenorganisation

5. Anfrageoptimierung

6. Transaktionsverwaltung

7. Datensicherheit und Wiederherstellung

8. Business Intelligence

Funktionen

Funktionen

Benutzerdefinierte Funktionen können Parameter annehmen, eine Aktion ausführen und das Ergebnis dieser Aktion als Wert zurückgeben.

Der Rückgabewert kann ein Skalarwert (Einzelwert) oder eine Tabelle sein.

Beispielsyntax:

```
CREATE FUNCTION [dbo].[FunctionName]
(
    @param1 int,
    @param2 int
)
RETURNS INT
AS
BEGIN
    --insert code here
    RETURN @param1 + @param2
END
```

Aufruf:

```
SELECT dbo.FunctionName(<param1>,<param2>)
```

Funktionen

Stored Procedures

Stored Procedures are pre-compiled objects which are compiled for the first time and its compiled format is saved, which executes (compiled code) whenever it is called. For more about a stored procedure, please refer to the article [Different types of Stored Procedure](#).

Functions

A function is compiled and executed every time whenever it is called. A function must return a value and cannot modify the data received as parameters. For more about functions, please refer to the article [Different types of Functions](#).

Basic Differences between Stored Procedure and Function in SQL Server

The function must return a value but in **Stored Procedure** it is optional. Even a procedure can return zero or n values.

Functions can have only input parameters for it whereas Procedures can have input or output parameters.

Functions can be called from Procedure whereas Procedures cannot be called from a Function.

Funktionen

Advance Differences between Stored Procedure and Function in SQL Server

The procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it.

Procedures cannot be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement.

Stored Procedures cannot be used in the [SQL](#) statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be.

Functions that return tables can be treated as another rowset. This can be used in JOINS with other tables.

Inline Function can be thought of as views that take parameters and can be used in JOINS and other Rowset operations.

An exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function.

We can use Transactions in Procedure whereas we can't use Transactions in Function.

Funktionen – Aufgabe 1

Schreiben Sie eine benutzerdefinierte Funktion, welche die Zeichen eines Strings zählt.

Die Funktion soll zwei Eingabeparameter haben:

- 1. zu überprüfender Text**
- 2. Switch ob mit oder ohne Leerzeichen gezählt werden soll.**

Rückgabewert ist die Anzahl der gezählten Zeichen.

Hilfreiche Funktionen:

LEN (string_expression)

REPLACE (string_expression , string_pattern , string_replacement)

Funktionen – Aufgabe 1

The screenshot shows a SQL query window with two statements:

```
1 select dbo.zaehlezeichen('Dies ist ein Test',1)
2 select dbo.zaehlezeichen('Dies ist ein Test',0)
```

Below the query window, the Results pane displays the output of the first statement:

	(No column name)
1	17

Below the Results pane, the Messages pane displays the output of the second statement:

	(No column name)
1	14

Funktionen – Aufgabe 2

Schreiben Sie eine benutzerdefinierte Funktion mit 2 Eingabeparametern vom Typ float, die als Ausgabe die Differenz der beiden übergebenen Zahlen auf Konsole ausgibt.

Die Aufgabe soll ohne Verwendung der Funktion ABS() gelöst werden!

Funktionen – Aufgabe 2

1	<code>select dbo.differenz(10,8)</code>
100 %	
Results	Messages
	(No column name)
1	2

1. Einführung

2. Datenbankentwurf

3. Datenbankimplementierung

4. Physische Datenorganisation

5. Anfrageoptimierung

6. Transaktionsverwaltung

7. Datensicherheit und Wiederherstellung

8. Business Intelligence

Cursor

Cursor

SQL-Statements erzeugen ein vollständiges Resultset, manchmal ist es jedoch von Vorteil, wenn die Ergebnisse zeilenweise verarbeitet werden.

Das Öffnen eines Cursors auf einem Resultset ermöglicht das zeilenweise Verarbeiten des Resultsets.

Syntaxbeispiel:

```
Declare @Variable as int;
Declare @CursorName AS CURSOR;

SET @CursorName = CURSOR FOR
SELECT <Feld> FROM <Tabelle>

OPEN @CursorName;
FETCH NEXT FROM @CursorName INTO @Variable

WHILE @@FETCH_STATUS = 0
BEGIN
    --insert Code here
    FETCH NEXT FROM @CursorName INTO @Variable
END

CLOSE @CursorName;
DEALLOCATE @CursorName;
```

Cursor – Aufgabe 1

Schreiben Sie einen Cursor für das Statement

SELECT spielernr, name, vorname

FROM spieler

Der Cursor soll über alle Ergebniszeilen iterieren (=„loop“ bis Ende erreicht)

Tipp: @@FETCH_STATUS =

Rückgabewert	Beschreibung
0	Die FETCH-Anweisung war erfolgreich.
-1	Die FETCH-Anweisung ist fehlgeschlagen, oder die Zeile war außerhalb des Resultsets.
-2	Die abgerufene Zeile fehlt.

Cursor – Aufgabe 1



Messages

2	Elfers	Rainer
6	Peters	Robert
7	Wiegand	Günther
8	Neuhaus	Berta
27	Kohl	Dagmar
28	Kohl	Claudia
39	Bischof	Dennis
44	Bäcker	Egon
57	Böhmen	Manfred
83	Hofmann	Philipp
95	Müller	Paul
100	Peters	Franz
104	Maurer	Doris
112	Bauer	Irene

Cursor – Aufgabe 2

Schreiben Sie einen Cursor, der jeweils die Anzahl der gewonnen und verlorenen Spiele der Spieler mit der Spielernr 2 und 6 in je eine Variable schreibt.

Geben Sie die Variablen anschließend aus.

Cursor – Aufgabe 2

Results		Messages	
Anzahl gewonnener Spiele von Spieler 6			
1	17		
Anzahl verlorener Spiele von Spieler 6			
1	10		
Anzahl gewonnener Spiele von Spieler 2			
1	6		
Anzahl verlorener Spiele von Spieler 2			
1	9		

Datenbankverbindung mit Java-Applikation

Datenbankverbindung mit einer Java-Applikation herstellen

Datenbankverbindung mit einer Java-Applikation herstellen

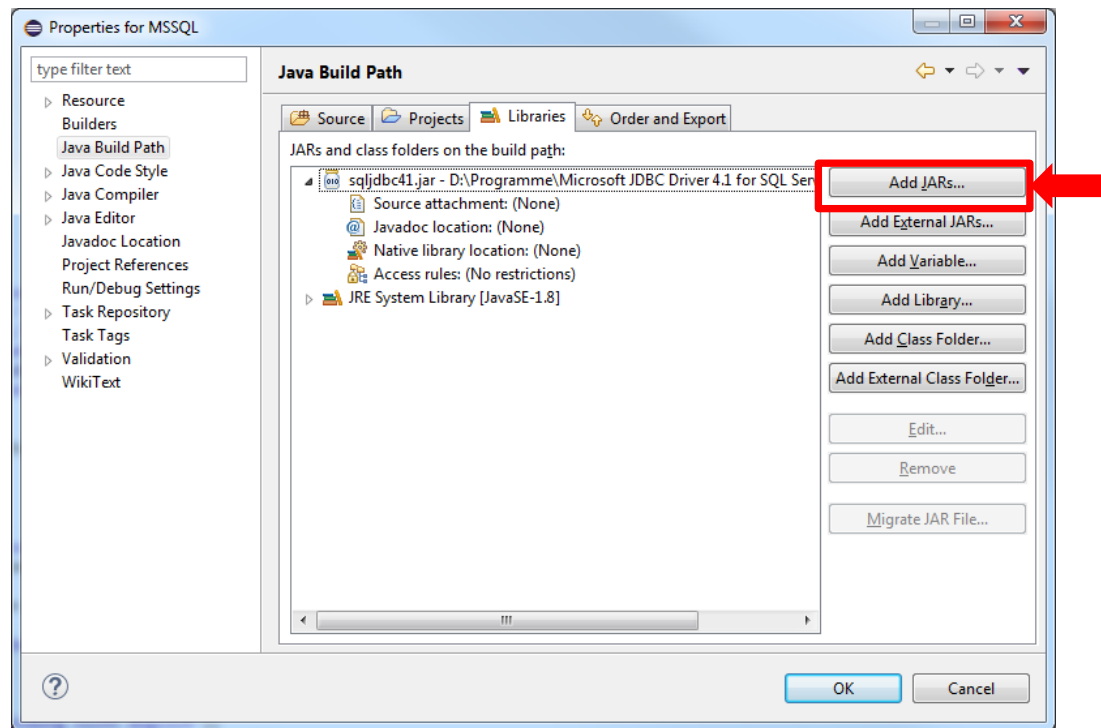
Beispiel mit

- Eclipse-Projekt (verfügbar auf Moodle)
- Microsoft JDBC-Treiber (verfügbar auf Moodle)

Datenbankverbindung mit einer Java-Applikation herstellen

„sqljdbc“-Bibliothek (Connector von Microsoft) muss in Projekt eingebunden werden, damit eine Verbindung zur DB hergestellt werden kann.

Einbinden in spezielles Eclipse-Projekt: Project → Properties



Example.java

```
//=====
//
// File:      Example.java
// Summary:   This sample application demonstrates how to connect to a
//            SQL Server database by using a connection URL. It also
//            demonstrates how to retrieve data from a SQL Server
//            database by using an SQL statement.
//
//=====
```

```
import java.sql.*;
```

```
public class Example {
```

```
    public static void main(String[] args) {
```

```
        // Hostname or IP
```

```
        String dbHost = "itnt0005";
```

```
        // Port
```

```
        String dbPort = "1433";
```

```
        // Database name
```

```
        String dbName = "Spieler2_LAB";
```

```
        // User
```

```
        String dbUser = "wkb4";
```

```
        // Password
```

```
        String dbPass = "wkb4";
```

```
        // Create a variable for the connection string.
```

```
        String connectionUrl = "jdbc:sqlserver://" +
```

```
            dbHost + ":" +
```

```
            dbPort + ";" +
```

```
            "databaseName=" + dbName + ";" +
```

```
            "user=" + dbUser + ";" +
```

```
            "password=" + dbPass + ";" ;
```

```
        // Declare the JDBC objects.
```

```
        Connection con = null;
```

```
        PreparedStatement p_stmt = null;
```

```
        ResultSet rs = null;
```

Example.java

```

try {
    // Establish the connection.
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
    con = DriverManager.getConnection(connectionUrl);

    // Create, prepare and execute an SQL statement that returns some data.
    String SQL = "SELECT * FROM spieler";
    p_stmt = con.prepareStatement(SQL);
    rs = p_stmt.executeQuery();

    // Iterate through the data in the result set and display it.
    while (rs.next()) {
        System.out.println(rs.getString("vorname") + " " + rs.getString("name"));
        //You can also use the column index; 4 = vorname, 3 = name
        //System.out.println(rs.getString(4) + " " + rs.getString(3));
    }
}

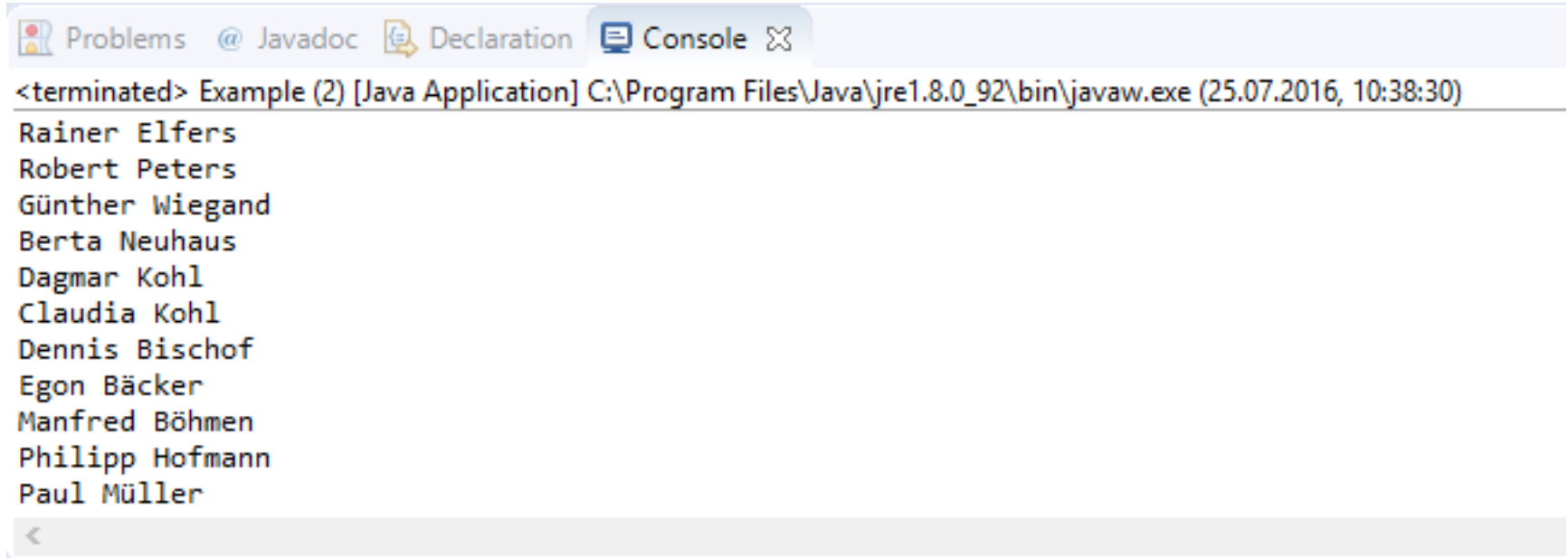
// Handle any errors that may have occurred.
catch (Exception e) {
    e.printStackTrace();
}

//Freeing up resources and close connection
finally {
    if (rs != null) try { rs.close(); } catch(Exception e) {}
    if (p_stmt != null) try { p_stmt.close(); } catch(Exception e) {}
    if (con != null) try { con.close(); } catch(Exception e) {}
}
}

```

Ausgabe

Konsolenausgabe in Eclipse:



```
<terminated> Example (2) [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (25.07.2016, 10:38:30)
Rainer Elfers
Robert Peters
Günther Wiegand
Berta Neuhaus
Dagmar Kohl
Claudia Kohl
Dennis Bischof
Egon Bäcker
Manfred Böhmen
Philipp Hofmann
Paul Müller
```